

# Introducción a R y RStudio

**Leticia Vega Alvarado**\*<sup>1</sup>

<sup>1</sup>Instituto de Ciencias Aplicadas y Tecnología - UNAM

\*leticia.vega@icat.unam.mx

14 Dec 2021

## Índice

Objetivo . . . . .	6
<b>1 Introducción . . . . .</b>	<b>6</b>
1.1 ¿Qué es R? . . . . .	6
1.1.1 Historia . . . . .	6
1.1.2 Distribuciones de R . . . . .	7
1.1.3 Instalando R . . . . .	7
1.2 ¿Qué es RStudio . . . . .	7
1.2.1 Distribuciones . . . . .	7
1.2.2 Instalación . . . . .	7
1.3 Iniciando RStudio desde el servidor del curso . . . . .	8
<b>2 Primeras nociones, un tour por RStudio . . . . .</b>	<b>8</b>
2.1 Ambiente de trabajo . . . . .	8
2.2 Paquetes de R . . . . .	8
2.2.1 Visualizando los paquetes instalados . . . . .	9
2.2.2 Cargando un paquete . . . . .	9
2.2.3 Visualizando las funciones de un paquete determinado . . . . .	10
2.2.4 Instalando paquetes . . . . .	10
2.3 Consultando la ayuda . . . . .	11
2.3.1 Ejercicios 1 . . . . .	11
2.4 Moviéndose entre los directorios . . . . .	12
2.4.1 ¿Cuál es mi directorio de trabajo? . . . . .	12
2.4.2 ¿Cómo cambio de directorio? . . . . .	12
2.4.3 Consultando el contenido de un directorio . . . . .	13
2.5 Saliendo de R . . . . .	13
2.5.1 Almacenando en un archivo los objetos creados en una sesión . . . . .	14
2.5.2 Almacenando, en un archivo, los comandos creados en una sesión . . . . .	14
2.5.3 Ejercicios 2 . . . . .	15
2.6 Manejo de proyectos (creación de subcarpetas data, doc, src, results) . . . . .	15

## Introducción a R y RStudio

<b>3</b>	<b>Tipos de datos y objetos . . . . .</b>	<b>16</b>
3.1	<b>Variables . . . . .</b>	<b>16</b>
3.1.1	<b>Reglas para nombres de variables . . . . .</b>	<b>16</b>
3.1.2	<b>R es sensible a mayúsculas y minúsculas en los nombres de objetos . . . . .</b>	<b>17</b>
3.2	<b>Tipos de datos . . . . .</b>	<b>17</b>
3.3	<b>Valores perdidos . . . . .</b>	<b>18</b>
3.3.1	<b>Ejercicios 3 . . . . .</b>	<b>18</b>
<b>4</b>	<b>Operadores aritméticos, relaciones y lógicos . . . . .</b>	<b>19</b>
4.1	<b>Operadores aritméticos . . . . .</b>	<b>19</b>
4.1.1	<b>¿Qué es el módulo? . . . . .</b>	<b>19</b>
4.1.2	<b>Misma precedencia . . . . .</b>	<b>20</b>
4.1.3	<b>Ejercicios 4 . . . . .</b>	<b>21</b>
4.2	<b>Funciones matemáticas . . . . .</b>	<b>21</b>
4.3	<b>Operadores relationales y lógicos . . . . .</b>	<b>22</b>
4.3.1	<b>Operadores relationales . . . . .</b>	<b>22</b>
4.3.2	<b>Operadores lógicos . . . . .</b>	<b>23</b>
4.3.2.1	<b>Operador <i>and</i> . . . . .</b>	<b>24</b>
4.3.2.2	<b>Operador <i>or</i> . . . . .</b>	<b>24</b>
4.3.2.3	<b>Operador lógico <i>not</i> . . . . .</b>	<b>25</b>
4.3.2.4	<b>Casos particulares NA y NaN . . . . .</b>	<b>25</b>
4.4	<b>Precedencia de operadores . . . . .</b>	<b>26</b>
4.5	<b>Ejercicios 5 . . . . .</b>	<b>26</b>
<b>5</b>	<b>Vectores . . . . .</b>	<b>27</b>
5.1	<b>Creación de un vector . . . . .</b>	<b>27</b>
5.2	<b>Adicionando elementos a un vector . . . . .</b>	<b>27</b>
5.3	<b>Combinando vectores . . . . .</b>	<b>28</b>
5.4	<b>Creación de vectores con secuencias numéricas usando <code>:</code> . . . . .</b>	<b>28</b>
5.5	<b>Accediendo a los elementos de un vector . . . . .</b>	<b>28</b>
5.6	<b>La función <code>names</code> . . . . .</b>	<b>29</b>
5.7	<b>Aritmética vectorial . . . . .</b>	<b>30</b>
5.8	<b>Operaciones relationales en un vector . . . . .</b>	<b>30</b>
5.9	<b>Ejercicios 6 . . . . .</b>	<b>31</b>
<b>6</b>	<b>Factores . . . . .</b>	<b>32</b>
6.1	<b>La función <code>summary</code> en factores . . . . .</b>	<b>32</b>
<b>7</b>	<b>Dataframes . . . . .</b>	<b>32</b>
7.1	<b>Manejo de dataframes . . . . .</b>	<b>34</b>
7.2	<b>Algunas funciones aplicables a <code>dataframe</code> . . . . .</b>	<b>35</b>
7.3	<b>Operaciones relationales en <code>data.frames</code> . . . . .</b>	<b>36</b>

7.4	Ejercicios 7 . . . . .	37
8	Lectura de archivos . . . . .	37
8.1	La función <code>read.table</code> . . . . .	37
8.2	Parámetros más comunes de <code>read.table</code> . . . . .	38
8.3	Algunas funciones para visualizar los datos . . . . .	39
8.4	Variantes de <code>read.table</code> . . . . .	40
8.5	La función <code>scan</code> . . . . .	40
9	Escritura de archivos . . . . .	40
9.1	La función <code>write.table</code> . . . . .	40
9.2	La función <code>write</code> . . . . .	41
9.3	Ejercicios 8 . . . . .	41
10	Gráficas con <code>ggplot2</code> . . . . .	41
10.1	¿Qué es <code>ggplot2</code> ? . . . . .	42
10.2	¿Cómo se generan las gráficas en R? . . . . .	42
10.3	Elementos principales de las gráficas de <code>ggplot2</code> . . . . .	43
10.4	Instalando <code>ggplot2</code> . . . . .	43
10.5	Conjunto de datos <code>msleep</code> . . . . .	44
10.6	Cargando <code>ggplot2</code> y visualizando los datos <code>msleep</code> . . . . .	44
10.7	Primeros pasos en graficación . . . . .	45
10.7.1	Mi primera gráfica . . . . .	46
10.8	Ejercicios 9 . . . . .	47
10.9	La estética ( <code>aes</code> ) . . . . .	48
10.9.1	Color . . . . .	48
10.9.2	Tamaño, transparencia y forma . . . . .	50
10.9.3	Cambiando los colores y formas manualmente. . . . .	52
10.10	Ejercicios 10 . . . . .	54
10.11	Más de una geometría en la gráfica . . . . .	54
10.12	Agregando la geometría de tendencia . . . . .	55
10.13	Ejercicios 11 . . . . .	56
10.14	Gráficas en múltiples paneles . . . . .	56
10.14.1	<code>facet_grid()</code> . . . . .	57
10.15	Ejercicios 12 . . . . .	58
10.16	Modificando los títulos y su estilo . . . . .	58
10.16.1	Agregando títulos . . . . .	58
10.16.2	Modificando el estilo de los textos . . . . .	60
10.17	Consultando las <code>fonts</code> disponibles . . . . .	63
10.17.1	Ejercicios 13 . . . . .	63
10.18	Modificando el fondo de la gráfica . . . . .	63

## Introducción a R y RStudio

10.18.1	Modificando el fondo de la gráfica con un fondo por defecto . . . . .	63
10.19	Ejercicios 14 . . . . .	64
10.20	Otras geometrías . . . . .	64
10.20.1	Histogramas . . . . .	65
10.21	Gráficas de cajas y bigotes ( <code>boxplots</code> ) . . . . .	67
10.22	Gráficas de barras . . . . .	69
10.23	Ejercicios 15 . . . . .	71
10.24	Guardando la imagen en un archivo . . . . .	71
10.24.1	Guardando la imagen en un archivo utilizando <code>ggsave()</code> . . . . .	71
10.24.1.1	Cambiando el tamaño de la imagen y los dpi's . . . . .	72
10.24.2	Guardando la imagen con instrucciones de R base y <code>dev.off()</code> . . . . .	72
10.25	Ejercicios 16 . . . . .	73
10.26	Ligas de información de <code>ggplot2</code> . . . . .	73
11	Respuestas de Ejercicios . . . . .	73
11.1	Solución Ejercicios 1 . . . . .	73
11.2	Solución Ejercicios 2 . . . . .	74
11.3	Solución Ejercicios 3 . . . . .	74
11.4	Solución Ejercicios 4 . . . . .	75
11.5	Solución Ejercicios 5 . . . . .	76
11.6	Solución Ejercicios 6 . . . . .	76
11.7	Solución Ejercicios 7 . . . . .	78
11.8	Solución Ejercicios 8 . . . . .	79
11.9	Solución Ejercicios 9 . . . . .	79
11.10	Solución Ejercicios 10 . . . . .	80
11.11	Solución Ejercicios 11 . . . . .	80
11.12	Solución Ejercicios 12 . . . . .	80
11.13	Solución Ejercicios 13 . . . . .	81
11.14	Solución Ejercicios 14 . . . . .	81
11.15	Solución Ejercicios 15 . . . . .	82
11.16	Solución Ejercicios 16 . . . . .	82
12	Apéndices . . . . .	83
12.1	Apéndice 1 . . . . .	83
12.1.1	Iniciando R desde línea de comandos . . . . .	83
12.1.2	Manejo de paquetes desde línea de comandos . . . . .	83
12.1.3	Consultando la ayuda desde línea de comandos . . . . .	84
12.1.3.1	Ayuda en general . . . . .	85
12.1.3.2	Ayuda relacionada a un término . . . . .	85
12.1.3.3	Ayuda sobre ejemplos de uso de una función . . . . .	86
12.1.3.4	Ayuda sobre argumentos de una función . . . . .	87
12.1.3.5	Búsquedas aproximadas . . . . .	87

12.2	Guardando y Cargando el Espacio de Trabajo desde línea de comandos . . . . .	87
12.2.1	Almacenando, en un archivo, objetos específicos creados en una sesión . . . . .	88
12.2.2	Guardando y Cargando el Historial de Comandos desde línea de comandos. . . . .	88
12.3	Apeéndice 2 . . . . .	88
12.3.1	Manipulación de Data frames con <code>dplyr</code> . . . . .	88
12.3.2	¿Qué es <code>dplyr</code> ? . . . . .	88
12.3.3	Gramática de <code>dplyr</code> . . . . .	89
12.3.4	La función <code>select()</code> . . . . .	89
12.3.5	La función <code>filter()</code> . . . . .	90
12.4	Apéndice 3 . . . . .	90
12.5	<code>facet_wrap()</code> . . . . .	90
12.6	Apéndice 4 . . . . .	92
12.6.1	Cambiando el fondo de mi gráfica de manera personalizada . . . . .	92
12.6.2	Asignando el diseño a un objeto . . . . .	93
12.7	Apéndice 5 . . . . .	94
12.7.1	Gráfica de Pie . . . . .	94

## Objetivo

Tener una visión general de qué es R, para qué sirve, cómo instalarlo y brindar los conceptos básicos de R para que los alumnos puedan utilizarlos para el manejo y análisis de datos. Así como conocer la gramática básica (estética y capas geométricas) para la creación de gráficas diferentes tipos de gráficas con el paquete `ggplot2`.

# 1 Introducción

## 1.1 ¿Qué es R?

R es un entorno integrado para el manejo de datos, cálculo y procedimientos gráficos y estadísticos. Los principales aspectos que ofrece son:

1. Facilidad para el manejo y el almacenamiento de datos.
2. Un conjunto de operadores para cálculo con vectores y matrices.
3. Una colección extensa e integrada de herramientas intermedias para el análisis estadístico de datos.
4. Multitud de facilidades gráficas.
5. Un lenguaje de programación simple.

### 1.1.1 Historia

R fue desarrollado en los 90's por Ross Ihaka y Robert Gentleman, del departamento de Estadística de la Universidad de Auckland. R es una implementación "open-source" del lenguaje S ( Creado por John Chambers a principios de los 70's ), que también es la base del sistema S-Plus (entorno comercial). El desarrollo actual de R es responsabilidad del *R Development Core Team*. Además, R dispone de una comunidad de desarrolladores/usuarios detrás que se dedican constantemente a la mejora y a la ampliación de las funcionalidades y capacidades del programa. Nosotros mismos podemos ser desarrolladores de R!!



### R: A Language for Data Analysis and Graphics

ROSS IHAKA and Robert GENTLEMAN

In this article we discuss our experience designing and implementing a statistical computing language. In developing this new language, we sought to combine what we felt were useful features from two existing computer languages. We feel that the new language provides advantages in the areas of portability, computational efficiency, memory management, and scoping.

**Key Words:** Computer language; Statistical computing.

Ross Ihaka is Senior Lecturer, and Robert Gentleman is Senior Lecturer, Department of Statistics, University of Auckland. Private Bag 92019, Auckland, New Zealand, e-mail: ihaka@stat.auckland.ac.nz.  
© 1996 American Statistical Association, Institute of Mathematical Statistics,  
*Journal of Computational and Graphical Statistics*, Volume 5, Number 3, Pages 299-314

**Figura 1: Creadores de R** [[@ihaka1996](#)]

Foto obtenida de (<https://www.stat.auckland.ac.nz/~ihaka/downloads/the-r-project.pdf>)

## Introducción a R y RStudio

### 1.1.2 Distribuciones de R

R se distribuye para Windows ,Linux, MacOSX y Unix. Los programas fuentes, binarios y la documentación de R, así como una lista de las preguntas más frecuentes acerca de R, se encuentran en el sitio de CRAN [Comprehensive R Archive Network](http://CRAN.R-project.org).

### 1.1.3 Instalando R

Los pasos para instalar R son:

1. Entrar a <http://cran.r-project.org/mirrors.html>
2. Elejir uno de los servidores
3. En la sección de “*Download and Install R*”, entrar a la liga de Linux, MacOS X o Windows según sea el caso.
4. Descargar
5. Iniciar instalación

Nota: Para fines prácticos de este curso, no haremos la instalación de R.

## 1.2 ¿Qué es RStudio

RStudio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para R, que facilita la tarea de uso interactivo de R y la programación de scripts. En este sentido, RStudio nos ofrece:

1. Visualización de los gráficos integrada en el mismo entorno
2. Acceso cómodo a la ayuda electrónica sobre R
3. Un editor que nos permite guardar nuestro trabajo
4. Una consola para la ejecución de código.
5. Editor de sintaxis que apoya la ejecución de código
6. Auto completado de código y sangría inteligente
7. Ejecución de código de R directamente desde el editor de código fuente.

### 1.2.1 Distribuciones

RStudio esta disponible en versiones *open source* y comercial, para escritorio (Windows, Mac y Linux) o desde web usando RStudio Server o RStudio Server Pro.

RStudio dispone de [hojas de consulta rápida](#), que facilitan el aprendizaje y el uso de algunos de sus paquetes.

### 1.2.2 Instalación

1. Instalar R, si no se encuentra previamente instalado. Ver la sección [Instalando R](#).
2. Ir a <https://www.rstudio.com>.
3. Seleccionar Download RStudio.
4. Descargar RStudio, dependiendo de nuestro sistema operativo (Windows, Mac o Linux).
5. Iniciar instalación.



Figura 2: Icono de RStudio

Una vez instalado, puedes iniciar RStudio, dando doble clic en el ícono de RStudio.

Nota: Para fines prácticos de este curso, no haremos la instalación de RStudio. Por otra parte, en este curso trabajaremos R desde el ambiente de trabajo de RStudio. Sin embargo, en el [Apéndice 1](#), se muestran una breve descripción de R desde línea de comandos, por ejemplo: [Iniciando R desde línea de comandos](#).

## 1.3 Iniciando RStudio desde el servidor del curso

Para nuestro taller trabajaremos en RStudio desde un servidor. Para acceder a RStudio debemos:

1. Ingresar a <http://132.248.32.180:8787>
2. Proporcionar nuestro usuario y password, y de manera automática se abrirá el ambiente de trabajo de RStudio.

## 2 Primeras nociones, un tour por RStudio

---

### 2.1 Ambiente de trabajo

RStudio por defecto tiene cuatro páneles de trabajo (Figura 3):

1. El panel inferior izquierdo es una consola de R en la que se puede escribir, ejecutar código, así como mostrar resultados del código que se va ejecutando.
2. El panel superior izquierdo es un editor de código, en este panel se abren los archivos en pestañas.
3. El panel superior derecho es para mostrar las variables en el entorno y un histórico de comandos ejecutados.
4. El panel inferior derecho contiene varias pestañas, entre ellas destacan files (muestra el directorio de archivos), plots (muestra las gráficas según se van ejecutando) y help (muestra la página de ayuda de las funciones cuando se soliciten).

### 2.2 Paquetes de R

Las funciones de R se agrupan en paquetes (*packages*). Los paquetes que contienen las funciones más habituales se incluyen por defecto en la distribución de R, y el resto se encuentran disponibles en [CRAN](#).

## Introducción a R y RStudio

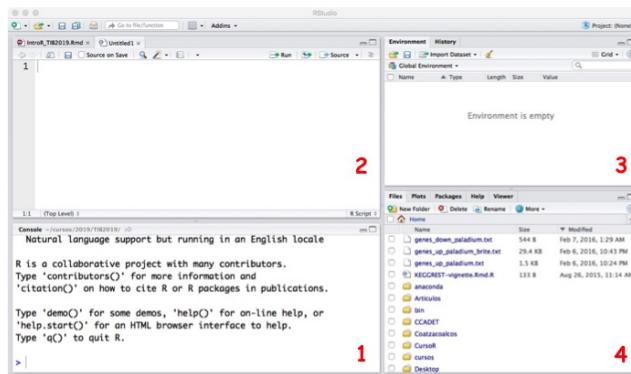


Figura 3: Ambiente de trabajo de RStudio

### 2.2.1 Visualizando los paquetes instalados

Para visualizar los paquetes que tenemos instalados seleccionamos la pestaña de **Packages**, que se encuentra en el panel inferior derecho de nuestro ambiente de trabajo (Figura 3, panel 4). Los paquetes que se encuentran seleccionados (Figura 4), son aquellos que están cargados y listos para usarse. Para utilizar un paquete instalado, es necesario cargarlo. Al iniciar R se cargan por defecto los paquetes básicos, por ejemplo: `graphics` y `stats`, entre otros.

Name	Description	Version
<input type="checkbox"/> glue	Interpreted String Literals	1.3.0
<input type="checkbox"/> GO.db	A set of annotation maps describing the entire Gene Ontology	3.7.0
<input type="checkbox"/> goseq	Gene Ontology analyser for RNA-seq and other length biased data	1.34.1
<input type="checkbox"/> gplots	Various R Programming Tools for Plotting Data	3.0.1.1
<input type="checkbox"/> graph	graph: A package to handle graph data structures	1.60.0
<input checked="" type="checkbox"/> graphics	The R Graphics Package	3.5.1
<input checked="" type="checkbox"/> grDevices	The R Graphics Devices and Support for Colours and Fonts	3.5.1
<input type="checkbox"/> grid	The Grid Graphics Package	3.5.1

Figura 4: Paquetes instalados

### 2.2.2 Cargando un paquete

Para cargar un paquete en particular se selecciona la casilla del paquete que queremos utilizar. Por ejemplo, si queremos usar el paquete `ggplot2`, marcamos la casilla correspondiente a este paquete (Figura 5).

Name	Description	Version
<input type="checkbox"/> ggcormplot	Visualization of a Correlation Matrix using 'ggplot2'	0.1.2
<input checked="" type="checkbox"/> ggplot2	Create Elegant Data Visualisations Using the Grammar of Graphics	3.1.0
<input type="checkbox"/> ggpubr	'ggplot2' Based Publication Ready Plots	0.2

Figura 5: Cargando un paquete

## Introducción a R y RStudio

### 2.2.3 Visualizando las funciones de un paquete determinado

Para visualizar las funciones contenidas en un paquete, daremos clic en el paquete que queremos consultar y posteriormente nos mostrará la documentación del paquete. Por ejemplo si damos clic en `ggplot2` nos mostrará sus funciones (Figura 6).

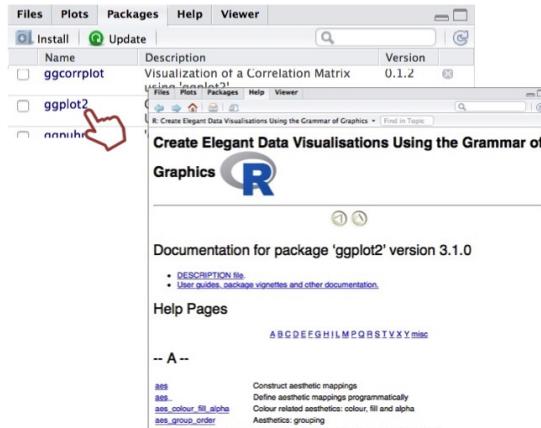


Figura 6: Visualización de funciones de un paquete

### 2.2.4 Instalando paquetes

Para instalar un paquete, seleccionamos **Install** que se encuentra en la pestaña de paquetes. En automático se abrirá una ventana en la cual escribiremos el nombre del paquete que queremos instalar, por ejemplo: `knitr`. La casilla donde escribimos el nombre del paquete, cuenta con la función de autocompletado, por lo tanto si hay algún valor coincidente (que comience por los caracteres escritos) te los mostrará (Figura 7). Es importante marcar la casilla **Install dependencies**, para que se instalen todas las dependencias, es decir, programas o paquetes que requiere el paquete que estamos instalando. Posteriormente, daremos clic en el botón **Install** e iniciará el proceso de instalación.

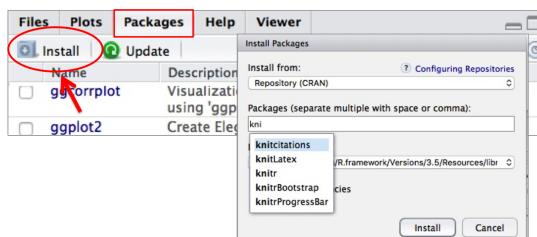


Figura 7: Instalación del paquete knitr

Nota: En el [Apéndice 1](#) se muestran algunos comandos para el [Manejo de paquetes desde Línea de comandos](#).

### 2.3 Consultando la ayuda

Para consultar la ayuda seleccionamos la pestaña **Help**, que se encuentra en el panel inferior derecho de nuestro ambiente de trabajo (Figura 3, panel 4). En esa pestaña encontraremos varios botones y campos para llenado (Figura 8).

1. Campo de búsqueda
2. Ayuda de R. Documentación general de R y RStudio
3. Mostrar Ayuda en una ventana emergente

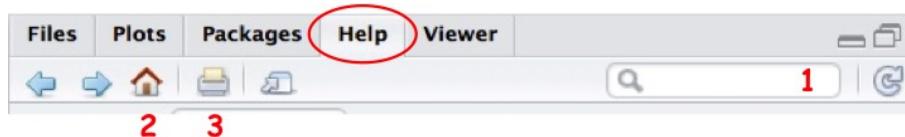


Figura 8: Menú de Ayuda

Para solicitar ayuda de un comando en particular, escibimos el comando en el campo de búsqueda del, damos enter y en automático apaecerá la ayuda. Por ejemplo, podemos solicitar ayuda de `matrix` (Figura 9)

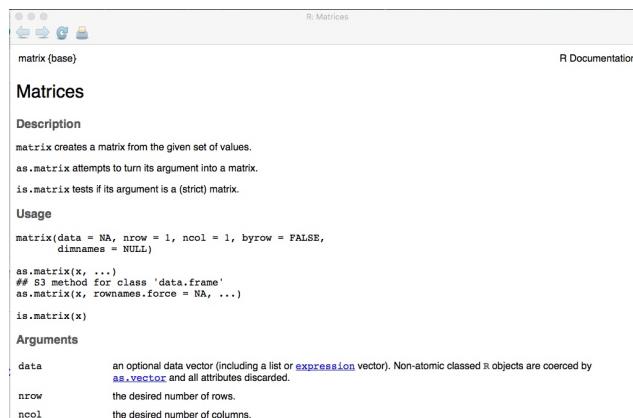


Figura 9: Menú de Ayuda

Como podemos observar en la Figura 9. La ayuda nos proporciona la descripción del comando consultado, sus argumentos y ejemplos, entre otra información.

Además, podemos realizar busquedas por aproximación, es decir, poniendo sólo una parte del término que buscamos o por un tema. Así, se desplegarán todas las páginas de ayuda en donde aparece dicho término. Por ejemplo, si buscamos el término “clustering” veremos, los paquetes en los que aparece ese término.

Nota: En el [Apéndice 1](#) se muestran algunos comandos para el [Consultando la ayuda desde Línea de comandos](#).

#### 2.3.1 Ejercicios 1

1. Busca qué hace la función `rm`
2. ¿Cuáles son los argumentos de la función `rep`?
3. Proporciona tres paquetes que contengan el término `prediction`.

Las respuestas se encuentran disponibles en: [Solución Ejercicios 1](#)

### 2.4 Moviéndose entre los directorios

El directorio de trabajo de R, corresponde a la ruta desde donde se inicia R. Cuando queremos trabajar con archivos, R los busca por defecto en el directorio de trabajo. Para visualizar archivos, cambiar o consultar el directorio de trabajo, seleccionamos la pestaña **Files**, que se encuentra en el panel inferior derecho de nuestro ambiente de trabajo (Figura 3, panel 4). En esa pestaña encontraremos diversos botones y un administrador de archivos (Figura 10A).

1. **More.** Funciones relacionadas con archivos o el directorio de trabajo.
2. **Home.** Para moverse a nuestro directorio raíz, que no necesariamente tiene que ser el mismo del directorio de trabajo.

La Figura 10B, corresponde al menú que se despliega con **More**

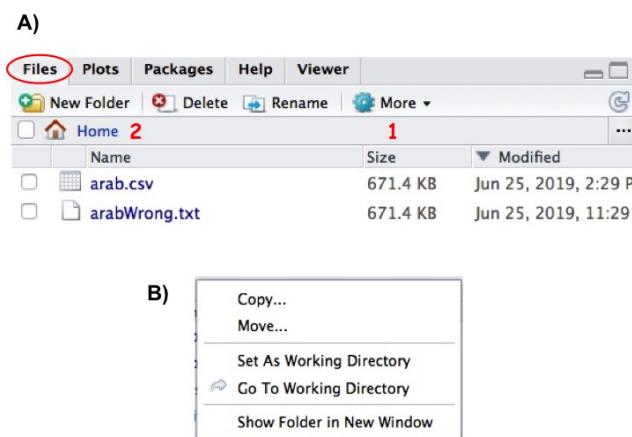


Figura 10: Menú de File

#### 2.4.1 ¿Cuál es mi directorio de trabajo?

Para movernos al directorio de trabajo, seleccionamos **Go To Working Directory** del menú **More** (Figura 10B) y en automático nos llevará al directorio de trabajo. También desde línea de comandos podemos escribir:

```
getwd()
```

```
[1] "/home/alumnoX"
```

#### 2.4.2 ¿Cómo cambio de directorio?

Para cambiar de directorio de trabajo:

1. Nos movemos a la carpeta que queremos definir como el directorio de trabajo, por medio del administrador de archivos.
2. Seleccionamos **Set As Working Directory** del menú **More**.

## Introducción a R y RStudio

Desde línea de comandos podemos usar la instrucción `setwd( )` y en los parentesis ponemos la dirección hacia donde queremos movernos.

```
setwd("/home/alumnoX/R")
```

Si consultamos nuevamente cuál es el directorio de trabajo, veremos que cambió.

```
getwd()
```

```
[1] "/home/alumnoX/R"
```

### 2.4.3 Consultando el contenido de un directorio

Para consultar el contenido de un directorio, nos movemos al directorio de interés, por medio del administrador de archivos o desde línea de comandos usamos la instrucción `dir( )` y en los parentesis ponemos la ruta del directorio que queremos consultar. Si dejamos los parentesis vacíos, nos mostrará el contenido del directorio de trabajo. por ejemplo:

```
dir()
```

## 2.5 Saliendo de R

Para salir de RStudio seleccionamos **Quit RStudio** de **RStudio** o **Quit Session** en **File** en el menú principal (Figura 11).

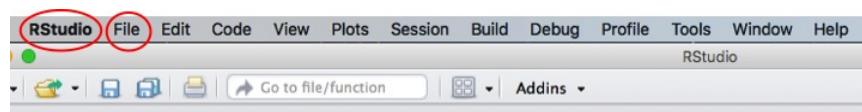


Figura 11: Menú Principal RStudio

También podemos escribir en la consola:

```
q()
```

```
# O podemos escribir la palabra completa
```

```
quit()
```

```
## Save workspace image? [y/n/c]:
```

Antes de cerrar la sesión, R nos preguntará si queremos almacenar el ambiente de trabajo, es decir, objetos y comandos usados en la sesión. Si le decimos que si, entonces los objetos se guardarán en un archivo llamado **.RData**, que es de tipo binario y los comandos se almacenarán en el archivo **.Rhistory**, que es de tipo texto. Estos dos archivos se guardan en el directorio de trabajo y quedan como archivos ocultos. La siguiente vez que iniciemos una sesión de R en el mismo directorio de trabajo, el ambiente de trabajo y todos los objetos se restablecerán.

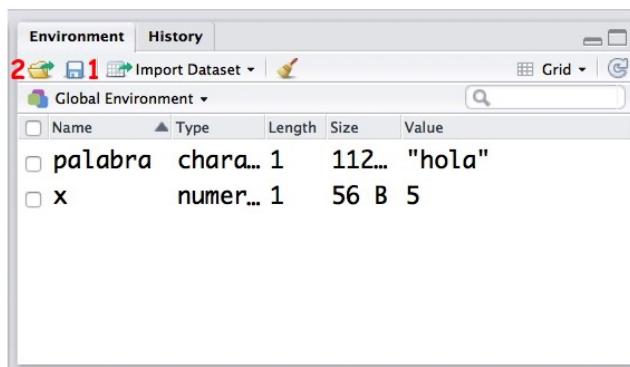
### 2.5.1 Almacenando en un archivo los objetos creados en una sesión

Puedes almacenar los objetos que se crearon en tu sesión en un archivo y después llamarlos, sin tener que cerrar R. Para almacenar todos los objetos utilizamos los botones y menús que se encuentran en la pestaña **Enviroment** en el panel superior derecho de nuestro ambiente de trabajo (Figura 3, panel 3).

Para ver cómo funciona, crearemos dos objetos y los almacenaremos

```
x<-5  
palabra<- "hola"
```

Como podemos observar en la sección **Enviroment** se muestran los objetos que acabamos de crear (Figura 12). Para guardar todos los objetos en un archivo, con el nombre y ruta que le especifiquemos, seleccionamos el ícono marcado con 1 en la Figura 12. Por ejemplo, guardaremos en el archivo “MisObjetos” (en el directorio de trabajo) los objetos creados hasta el momento.



Name	Type	Length	Size	Value
palabra	chara...	1	112...	"hola"
x	numer...	1	56	B 5

Figura 12: Ambiente de los objetos

Si nos movemos al directorio donde guardamos el archivo “MisObjetos”, veremos que por defecto R coloca la extención “.RData” al archivo.

Para cargar los objetos almacenados en el archivo MisObjetos, utilizamos el ícono marcado con 2 en la Figura 12 y seleccionamos el archivo que queremos cargar.

En el [Apéndice 1](#) se describe cómo guardar los objetos desde línea de comandos en la sección [Guardando y Cargando el Espacio de Trabajo desde línea de comandos](#)

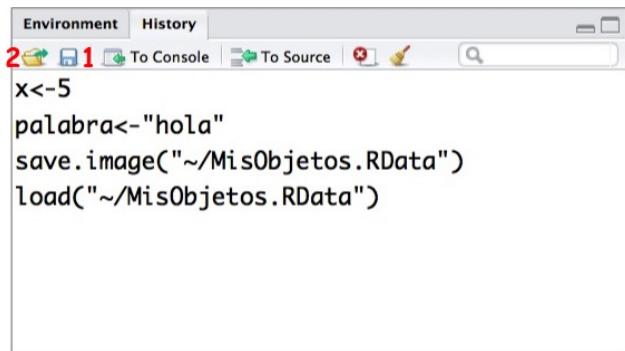
### 2.5.2 Almacenando, en un archivo, los comandos creados en una sesión

También podemos guardar el historial de los comandos utilizados en una sesión. Para ello usaremos los botones de la pestaña **History**, que se encuentra en el panel superior derecho de nuestro ambiente de trabajo (Figura 3, panel 3). En la ventana de la pestaña **History**, se muestran todos los comandos que hemos escrito en la sesión (Figura 13).

Para guardar los comandos en un archivo, con el nombre y ruta que le especifiquemos, seleccionamos el ícono marcado con 1 en la Figura 13. Por ejemplo, guardaremos en el archivo “MisComandos” (en el directorio de trabajo) los comandos utilizados hasta el momento.

Para cargar los comandos almacenados en el archivo MisComandos, utilizamos el ícono marcado con 2 en la Figura 13 y seleccionamos el archivo que queremos cargar.

## Introducción a R y RStudio



The screenshot shows the RStudio interface with the 'History' tab selected in the top navigation bar. Below the tabs, there are two entries in the command history:

```
x<-5
palabra<-"holo"
```

Figura 13: Historial de comndos

En el [Apéndice 1](#) se describe cómo guardar el historial de comandos desde línea de comandos en la sección [Guardando y Cargando el Historial de Comandos desde línea de comandos](#).

### 2.5.3 Ejercicios 2

1. Busca los archivos **.RData** y **.Rhistory**.
2. ¿Cómo veo el contenido del archivo **.Rhistory**?
3. ¿En qué directorio fueron creados los archivos **.RData** y **.Rhistory**?
4. ¿Por qué se crearon ahí?

Las respuestas se encuentran disponibles en: [Solución Ejercicios 2](#)

## 2.6 Manejo de proyectos (creación de subcarpetas data, doc, src, results)

Existen diversas maneras de organizar nuestros proyectos, con la finalidad de hacer más fácil el manejo de los mismos. Una de las herramientas útiles de RStudio es la de gestión de proyectos, que nos permite crear un proyecto autocontenido y reproducible. Para este curso crearemos un proyecto en RStudio, donde se almacenará todo lo que desarrollemos en el curso.

Para crear un proyecto:

1. Hacer **clic** en el menú **File**, luego en **New Project**.
2. Hacer **clic** en **New Directory**
3. Hacer **clic** en **Empty Project**
4. Introducir el nombre del directorio para guardar tu proyecto, por ejemplo: "ModuloR"
5. Hacer **clic** en el botón **Create Project**

En el directorio "ModuloR" se crea un objeto de extensión **.RProj**. Cuando abrimos este objeto, con un doble **clic**, se abre una sesión de RStudio y se establece, de manera automática, a "ModuloR", como el directorio de trabajo.

Ahora crearemos dentro de nuestro proyecto las siguientes carpetas:

- doc: para guardar toda la documentación pertinente al proyecto
- dat: que contiene los datos iniciales, es decir, sin procesar.
- scr: para almacenar programas o código en R
- res: donde se almacenarán los resultados finales o parciales (gráficos, informes, etc.)

## 3 Tipos de datos y objetos

En términos generales, todos los elementos que maneja R son objetos: un valor numérico es un objeto, una variable es un objeto, un vector es un objeto, una función es un objeto, una base de datos es un objeto, un gráfico es un objeto, ...

### 3.1 Variables

Las variables son localidades de memoria a las que les asignamos un nombre para almacenar un valor que puede cambiar en el transcurso de un programa. La Figura, muestra de manera esquemática cómo se representan las variables en la memoria.

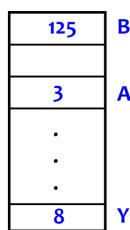


Figura 14: Representación de variables en la memoria

En R existen 3 maneras de hacer asignaciones de valores a las variables, utilizando los símbolos:  
`<-`, `->`, `=`. Por ejemplo:

```
A = 3
B <- 125
8 -> Y
```

Si una operación no es asignada a una variable, la operación se realizará y su resultado sólo se desplegará en la pantalla, pero no quedará almacenada. Por ejemplo:

```
3
## [1] 3
125 - 48
## [1] 77
2/5
## [1] 0.4
```

Nota: Podemos agregar comentarios a nuestro código utilizando el carácter `#`. Es muy útil documentar el código, ya que lo hace más entendible. Por ejemplo:

```
# La variable num_gene almacena el numero de genes del genoma
num_gene<-3600
genoma<- "Gsulf" # genoma guarda el nombre del genoma
```

#### 3.1.1 Reglas para nombres de variables

Al momento de definir el nombre de una variable debemos tomar en consideración las siguientes reglas:

## Introducción a R y RStudio

1. No deben comenzar con un número
2. No deben contener espacios en blanco
3. No deben contener símbolos especiales, excepto guión bajo ( \_ ) o punto( . ).

Ejemplos de nombres inválidos son:

```
Factor de cambio = 2  
1x <- 32  
mes-actual = "enero"
```

Algunos ejemplos de nombres válidos son:

```
Factor.de.cambio = 2  
x1 <- 32  
mes_actual = "enero"  
sexo <- "fem"; curso2017 <- "Estadistica"; x<-25  
b <- 3 -  
+ 8
```

Notas:

- 1) Como vimos en el ejemplo anterior, cada comando puede estar en una línea o varios comandos por línea separados por ";".
- 2) Si al escribir un comando damos **enter** y el comando queda incompleto, entonces, de manera automática R colocará un símbolo de más (+) indicando que el comando se encuentra incompleto y permitiendo completarlo. Mientras no se complete el comando, seguirá apareciendo el símbolo +, cada vez que demos **enter**.

### 3.1.2 R es sensible a mayúsculas y minúsculas en los nombres de objetos

Si bien es cierto que los nombres de variables pueden contener mayúsculas y minúsculas, es importante mencionar que, R distingue entre mayúsculas y minúsculas. Por ejemplo:

```
edad <- 3  
Edad <- 6  
EDAD <- 25  
EdAd <- 10
```

Todas las definiciones anteriores son variables diferentes.

## 3.2 Tipos de datos

En R (como en otros lenguajes de programación) hay diferentes tipos de datos; dentro de los más utilizados están los numéricos, lógicos y alfanuméricos (cadenas, strings en inglés). Por ejemplo:

```
Edad <- 18      # Valor numérico  
LogFoldC <- 2.0075 # Valor numérico  
resultado <- TRUE    # Valor lógico o booleano  
geneName = "CRP"   # Valor alfanumérico (caracter)
```

## Introducción a R y RStudio

Nota: Es importante resaltar que los valores de tipo carácter deben ser delimitados por comillas simples o dobles, de lo contrario R considera que es el nombre de un objeto.

### 3.3 Valores perdidos

Los valores perdidos constituyen una clase particular de dato, que se codifica siempre como **NA** (*Not Available*), independientemente de que la variable sea numérica o alfanumérica. Estos valores sirven, por ejemplo, cuando se registran datos y no es posible obtener la información de algunas variables para determinados casos. Con carácter general cualquier operación que involucre a un valor perdido dará como resultado también **NA**. Por ejemplo:

```
escolaridad<-NA  
estado_civil<-NA
```

Una segunda clase de **valores “perdidos”** en R corresponden a cálculos numéricos cuyos resultados son indefinidos o infinitos, como por ejemplo,  $\log(-1)$  o  $\frac{5}{0}$ . En el caso de los valores indefinidos, el resultado que devuelve R es **NaN** (*Not a Number*), y para los valores infinitos puede devolver **Inf** o **-Inf**. Por ejemplo:

```
5/0          # Valor Infinito  
## [1] Inf  
log(-1)      # Valor indefinido  
## Warning in log(-1): NaNs produced  
## [1] NaN  
-3/0          # Valor Infinito negativo  
## [1] -Inf  
  
# Los valores se pueden asignar a variables  
  
x<-5/0  
y<-log(-1)  
## Warning in log(-1): NaNs produced
```

#### 3.3.1 Ejercicios 3

1. ¿Cuáles de las siguientes definiciones son correctas?

```
variable.Uno <- 10  
25 <- "curso R"  
Sqrt25 <- 5  
nombre-25 <- 50  
resultado <- 10/0  
cer0<-3/-Inf  
Inf<-48
```

Las respuestas se encuentran disponibles en: [Solución Ejercicios 3](#)

## 4 Operadores aritméticos, relationales y lógicos

R, entre otras cosas puede realizar operaciones aritméticas básicas, así como operaciones relationales y lógicas. En esta sección veremos los operadores aritméticos, relationales y lógicos, así como las funciones matemáticas más usuales ya predefinidas en R listas para ser usadas.

### 4.1 Operadores aritméticos

Los operadores aritméticos, más habituales en R se detallan en la Tabla.

Cuadro 1: Operadores aritméticos

Operador	Símbolo	Ejemplos
<b>Suma</b>	+	9 + 25
<b>Resta</b>	-	80 - 9.65
<b>Multiplicación</b>	*	12.2 * 15
<b>División</b>	/	2 / 3
<b>Exponenciación</b>	** o ^	3 ** 4
<b>Módulo/Residuo</b>	% %	2 % % 3

#### 4.1.1 ¿Qué es el módulo?

El módulo es el residuo de una división entera, por ejemplo, el resultado de 7 módulo 2 es 1, como podemos observar en la siguiente figura.

```
knitr::include_graphics("imagenes/Modulo.png")
```

A diagram illustrating integer division. A horizontal line with a vertical tick on its left side separates the dividend (7) from the divisor (2). Above the line is the quotient (3). Below the line is the remainder (1). A red arrow points to the number 1 with the label "Residuo o módulo".

Los resultados de los ejemplos se muestran a continuación:

```
9 + 25
## [1] 34
80 - 9.65
## [1] 70.35
12.2 * 15
## [1] 183
2 / 3
## [1] 0.6666667
3 ** 4
```

## Introducción a R y RStudio

```
## [1] 81  
2 %% 3  
## [1] 2
```

En R, al igual que en matemáticas, las operaciones tienen un orden de evaluación definido (precedencia). Cuando tenemos diversos operadores en una misma expresión, algunos de estos operadores se aplican antes que otros y el resultado final de la expresión dependerá de la precedencia de los operadores.

El orden de operaciones incluye a los operadores aritméticos, relacionales, lógicos y de asignación. Sin embargo, en principio veremos el orden de las operaciones aritméticas y asignación.

```
knitr::include_graphics("imagenes/PrioridadOperadores_1.png")
```

Operadores <chr>	Significado <chr>	Orden <dbl>
( )	Parentesis	1
^	Potencia	2
-X, +X	Menos y más unario	3
%%	Módulo	4
* /	Multiplicación y división	5
+ -	Suma y resta	6
=, <-, ->	Asignación	7

Por ejemplo:  $6 ^ 2 - 20 * 2$  se evaluará como -4.

```
# En este caso se evalua primero la exponenciación, posterior  
# la multiplicación y finalmente la resta  
6 ^ 2 - 20 * 2  
## [1] -4  
  
# En este caso se realiza primero la exponenciación, posterior  
# la resta y finalmente la multiplicación  
(6 ^ 2 - 20) * 2  
## [1] 32  
  
# En este caso se realiza primero la resta, posteriormente la  
# exponenciación y finalmente la multiplicación  
6 ^ (2 - 20) * 2  
## [1] 1.96928e-14
```

### 4.1.2 Misma precedencia

Cuando los operadores tienen la misma precedencia, entonces la evaluación se realiza de izquierda a derecha, por ejemplo:

```
# Si evaluamos la expresión
```

## Introducción a R y RStudio

```
3/4/5  
## [1] 0.15
```

Si queremos cambiar el orden en el que se realiza la operación, podemos utilizar los paréntesis. Es decir, por ejemplo:

```
# CAMbiando el orden deen que se realizan las operaciones  
3/(4/5)  
## [1] 3.75
```

### 4.1.3 Ejercicios 4

1. Predice el resultado, luego corrobóralo:

- a.  $8 + 2$
- b.  $13 / 4$
- c.  $6 + 5 * 10 - 3$
- d.  $6 + 5 * (10 - 3)$
- e.  $9 (8 - 4)$
- f.  $A + b$
- g.  $A <- 5; a <- 11; A + a$
- h.  $c <- \text{"prueba"}$
- i.  $a + c$

2. Calcula algunos datos sobre una circunferencia a partir de su radio.

- a. Guarda en la variable `radio` el radio de la circunferencia con la que vamos a trabajar.
- b. Calcula el perímetro de la circunferencia ( $2\pi \times radio$ )
- c. Calcula el área del círculo delimitado por dicha circunferencia ( $\pi \times radio^2$ )

3. Escribe las siguientes expresiones aritméticas en R:

$$\frac{a}{b} + 1$$

$$\frac{a+b}{c+d}$$

$$(a+b)\frac{c}{d}$$

$$\frac{c+\frac{b}{a}}{d+\frac{a}{c}}$$

Las respuestas se encuentran disponibles en: [Solución Ejercicios 4](#)

## 4.2 Funciones matemáticas

Además de las operaciones aritméticas básicas, R cuenta con una serie de funciones matemáticas, así como diversas funciones útiles en estadística. Algunas de estas funciones se detallan en la Tabla 2.

Los resultados de los ejemplos de funciones se muestran a continuación.

## Introducción a R y RStudio

Cuadro 2: Algunas funciones matemáticas

Función	Símbolo	Ejemplos
Raíz cuadrada	<code>sqrt()</code>	<code>sqrt(25)</code>
coseno	<code>cos()</code>	<code>cos(0)</code>
logaritmo	<code>log()</code>	<code>log(2)</code>
Redondeo	<code>round()</code>	<code>round(3.85)</code>
valor absoluto	<code>abs()</code>	<code>abs(48 - 90)</code>

```
sqrt(25)
## [1] 5
cos(0)
## [1] 1
log(2)
## [1] 0.6931472
round(3.85)
## [1] 4
abs(48 - 90)
## [1] 42
```

## 4.3 Operadores relacionales y lógicos

### 4.3.1 Operadores relacionales

Los operadores relacionales se utilizan para comparar dos valores. El resultado de la comparación es de tipo lógico, es decir, toma un valor de Verdadero o Falso. Los operadores relacionales pueden aplicarse sobre todos los tipos de datos. En la Tabla 3 se muestran los operadores relacionales.

Cuadro 3: Operadores relacionales

Operador	Símbolo	Ejemplo
Igual	<code>==</code>	<code>5 == 3</code>
Menor que	<code>&lt;</code>	<code>5 &lt; 3</code>
Menor o igual que	<code>&lt;=</code>	<code>5 &lt;= 3</code>
Mayor que	<code>&gt;</code>	<code>5 &gt; 3</code>
Mayor o igual que	<code>&gt;=</code>	<code>5 &gt;= 3</code>
Diferente	<code>!=</code>	<code>5 != 3</code>

Veamos cuáles son los resultados de los ejemplos anteriores.

```
5 == 3
## [1] FALSE
5 < 3
## [1] FALSE
5 <= 3
## [1] FALSE
```

## Introducción a R y RStudio

```
5 > 3
## [1] TRUE
5 >= 3
## [1] TRUE
5 != 3
## [1] TRUE
```

Veamos otros ejemplos de estos operadores:

```
5 == 10/2
## [1] TRUE

x<- 10
y <- 25
automovil<- "Toyota"
nombre<- "Julia"

x < 50
## [1] TRUE
y >= 32
## [1] FALSE
x == y
## [1] FALSE
nombre != automovil
## [1] TRUE
nombre <= automovil
## [1] TRUE
```

### 4.3.2 Operadores lógicos

Los operadores lógicos son usados para evaluar proposiciones lógicas. Las proposiciones son enunciados aseverativos que pueden ser evaluados en términos de verdadero (*TRUE*) o falso (*FALSE*). Los operadores lógicos más utilizados son los que se describen en la Tabla 4..

Cuadro 4: Operadores lógicos

Operador	Símbolo	Ejemplo
Y	&	(5 > 3) & (8 == 25)
O		(5 > 3)   (8 == 25)
Negación	!	!(5 > 3)

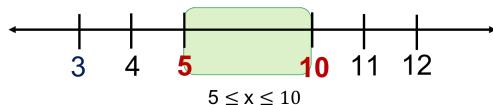
```
(5 > 3) & (8 == 25)
## [1] FALSE
(5 > 3) | (8 == 25)
## [1] TRUE
!(5 > 3)
## [1] FALSE
```

Las modalidades son: ( & ) cumplimiento en forma simultanea de las dos expresiones, para que el resultado sea verdadero, de lo contrario será falso, ( | ) cumplimiento alternativo, es decir por lo menos una de las expresiones debe ser verdadera, para que el resultado sea verdadero, de los contrario será falso. ( ! ) cumplimiento negado, si la expresión se evalua en verdadera al negarla el resultado será falso y viceversa.

**4.3.2.1 Operador and** Veamos el uso del operador **and** con e le siguiente ejemplo. Para determinar si un valor  $x$  se encuentra en el intervalo  $[5,10]$ , utilizamos el operador ( $\&$ ). Es decir, para establecer que el valor  $x$  se encuentra en el intervalo, éste debe ser mayor o igual a 5 y menor o igual a 10.

```
knitr:::include_graphics("imagenes/IntervaloAnd.png")
```

¿ $x$  pertenece al  
intervalo  $[5, 10]$  ?



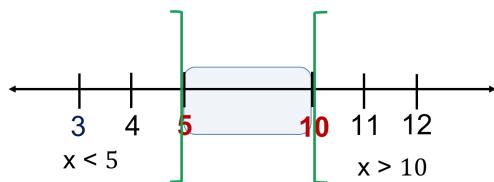
```
# Verificar si un número x se encuentra en el intervalo [5, 10]
x<-7
(x >= 5) & (x <= 10)
## [1] TRUE

# Ahora si cambiamos el valor de x
x<-12
(x >= 7) & (x <= 10)
## [1] FALSE
```

**4.3.2.2 Operador or** Ahora veamos un ejemplo del operador **or**. En este caso queremos determinar si el valor  $x$  está fuera del intervalo  $[5,10]$ . Para saberlo se requiere que el valor de  $x$  sea menor que 5 o mayor que 10.

```
knitr:::include_graphics("imagenes/IntervaloOr.png")
```

¿ $x$  no pertenece al  
intervalo  $[5, 10]$  ?



## Introducción a R y RStudio

```
# Verificar si un número x no se encuentra en el intervalo [5, 10]
x <- 15
(x < 5) | (x > 10)
## [1] TRUE

# Ahora si cambiamos el valor de x
x <- 8
(x < 5) | (x > 10)
## [1] FALSE
```

```
Casado<- TRUE
```

```
!Casado
## [1] FALSE
```

**4.3.2.3 Operador lógico `not`** Retomando el ejercicio anterior, donde queremos saber si un valor x está fuera del intervalo [5, 10], es decir, que **no** pertenece a dicho intervalo. En este caso podemos utilizar el operador negación.

```
# Verificar si un número x no se encuentra en el intervalo [5, 10]
x<-15
!((x >= 5) & (x <= 10))
## [1] TRUE

# Ahora si cambiamos el valor de x
x<-8
!((x >= 7) & (x <= 10))
## [1] FALSE
```

**4.3.2.4 Casos particulares NA y NaN** Para saber si un objeto contiene el valor de NA o NaN, no utilizamos los operadores relacional (`==` o `!=`), porque el resultado será nuevamente el valor perdido. Por ejemplo:

```
x <- NA
y<-log(-10)
## Warning in log(-10): NaNs produced

x
## [1] NA
y
## [1] NaN

x == NA
## [1] NA
y != NA
## [1] NA
```

Para identificar si los objeto x o y contienen un valor perdido, utilizamos las funciones `is.na` o `is.nan`. Por ejemplo:

```
is.na(x)
## [1] TRUE
is.nan(y)
## [1] TRUE
is.nan(x)
## [1] FALSE
is.na(5)
## [1] FALSE
```

## 4.4 Precedencia de operadores

En la siguiente tabla podemos observar la precedencia de los operadores aritméticos, relacionales, lógicos y de asignación.

Operadores <chr>	Significado <chr>	Orden <dbl>
()	Parentesis	1
^	Potencia	2
-X, +X	Menos y más unario	3
%%	Módulo	4
* /	Multiplicación y división	5
+ -	Suma y resta	6
<,>, <=, >=, ==, !=	Relacionales	7
!	Not lógico	8
&, &&	And	9
,	Or	10
=, <-,->	Asignación	11

## 4.5 Ejercicios 5

1. Escribir la expresión necesaria para evaluar si el valor almacenado en la variable x, coincide con alguno de una serie de valores, por ejemplo 2, 7 y 9.
2. Escribir la expresión necesaria para evaluar si el valor almacenado en la variable no coincide con alguno de una serie de valores, por ejemplo 2, 7 y 9
3. Escribe una expresión lógica que compruebe si una variable contiene un valor par. Nota: el operador % % devuelve el resto de la división entera.
4. Intenta predecir el resultado de evaluar las siguientes expresiones lógicas. Piensa en el orden en que se evalúan las subexpresiones dentro de cada expresión. Los valores de las variables son: a <- TRUE, b <- TRUE, c <- FALSE, d <- FALSE.
  - a. c || !a && b
  - b. !(a || c) || b && !c
  - c. !( ! (a && c || d))
  - d. !(5<3) && a || !(d || c)

Las respuestas se encuentran disponibles en: [Solución Ejercicios 5](#)

## 5 Vectores

Un vector en R, es esencialmente una lista ordenada de elementos, donde todos los elementos son del mismo tipo (enteros, caracteres, ...) y se encuentran agrupados bajo un mismo nombre. La función más utilizada para generar un vector, es `c()` (de *combine*). **Datos simples como un valor numérico son en realidad un vector.** Por ejemplo:

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

En este caso, un vector de tipo numérico y de longitud igual a 1. Para verificar si un valor es un vector, utilizamos la función `is.vector()`.

```
is.vector(3)  
## [1] TRUE
```

### 5.1 Creación de un vector

La función `c()` nos permite crear un vector. Esta función recibe como argumento los elementos, que queremos combinar en el vector, separados por coma. Por ejemplo:

```
# Vector numerico  
edades<-c(25,8,52,18,0,93)  
  
# Vector de cadenas  
nombres<-c("Ana","Luis","Juan","Paty","Jair","Mary")  
  
# Vector logico  
casado<-c(TRUE,FALSE,TRUE,FALSE)
```

Para ver los valores de los vectores que acabamos de crear, tecleamos el nombre del vector, por ejemplo:

```
edades  
## [1] 25 8 52 18 0 93
```

### 5.2 Adicionando elementos a un vector

Existen diversas formas para agregar un elemento a un vector, una de ellas es volver a utilizar el operador compose (`c`), por ejemplo:

```
# Vector numerico  
casado <- c(casado, FALSE, FALSE)  
casado  
## [1] TRUE FALSE TRUE FALSE FALSE FALSE
```

En este caso reasignamos el resultado al mismo objeto `casado`.

### 5.3 Combinando vectores

Para combinar vectores utilizaremos nuevamente la función `c()`, por ejemplo, podemos crear dos vectores y posteriormente combinarlos en un tercero.

```
vec_1<-c(1,2,3)
vec_2<-c(9,8,7)

vec_3<-c(vec_2,vec_2)
vec_3
## [1] 9 8 7 9 8 7
```

Si intentamos **combinar datos de diferentes tipos** en un mismo vector, R realizará **coerción automáticamente**. El vector resultante será del tipo más flexible entre los datos que contenga, siguiendo las reglas de coerción. En principio el tipo más flexible es el carácter. Por ejemplo

```
vec_tip_mezclado <- c(33,"casa",TRUE,8.03)
vec_tip_mezclado
## [1] "33"    "casa"   "TRUE"   "8.03"
```

En este caso todos los datos se convierten a tipo carácter.

### 5.4 Creación de vectores con secuencias numéricas usando `:`

Se pueden crear vectores de secuencias numéricas usando `::`. Antes de los dos puntos escribimos el número de inicio de la secuencia y después de los dos puntos, el número final. Por ejemplo:

```
# Secuencia del 1 al 8
1:8
## [1] 1 2 3 4 5 6 7 8

# Secuencia de -20 a 10
-5:3
## [1] -5 -4 -3 -2 -1  0  1  2  3
```

### 5.5 Accediendo a los elementos de un vector

Además de las funciones que operan sobre todos los elementos del vector, R permite manipular solo una parte del vector. Para extraer partes de un vector, usamos corchetes cuadrados. Como podemos observar en la siguiente figura, los elementos de un vector están ordenados y constan de un índice. Por lo tanto para hacer referencia a algún elemento, colocamos en el corchete el índice del elemento al que queremos acceder.

## Introducción a R y RStudio

edades	25	8	52	18	0	93	indice
	1	2	3	4	5	6	

**edades[ indice ]**

**edades[ c( indices ) ]**

```
# Primer elemento del vector
edades[1]
## [1] 25

# Quinto elemento del vector
edades[5]
## [1] 0

# Elementos no consecutivos
edades[c(1,3,6)]
## [1] 25 52 93

# Elementos consecutivos
edades[2:5]
## [1] 8 52 18 0
```

Para excluir algunos elementos del vector, utilizamos el signo menos ( - ). Por ejemplo:

```
# Para excluir algún elemento
edades[-c(3)]
## [1] 25 8 18 0 93
edades[-c(1,3)]
## [1] 8 18 0 93
```

## 5.6 La función **names**

Es posible asignarle etiquetas (nombres) a un vector por medio de la función **names()**. Por ejemplo, al vector de edades, le podemos poner como etiquetas los valores de vector nombres.

edades	Ana	Luis	Juan	Paty	Jair	Mary	Etiqueta (names)
	25	8	52	18	0	93	indice
	1	2	3	4	5	6	

**edades[ etiqueta ]**

**edades[ c( etiquetas ) ]**

## Introducción a R y RStudio

```
# Asignando etiquetas al vector edades
names(edades)<-nombres

# Visualizando nuevamente el vector edades
edades
## Ana Luis Juan Paty Jair Mary
## 25     8    52   18     0    93

# Accediendo a los elementos del vector edades por la etiqueta
edades["Ana"]
## Ana
## 25
edades[c("Mary", "Juan")]
## Mary Juan
## 93    52

# Accediendo a los elementos del vector edades por los indices
edades[1]
## Ana
## 25
edades[c(6,3)]
## Mary Juan
## 93    52
```

## 5.7 Aritmética vectorial

R tiene aritmética vectorial, por lo que los vectores pueden aparecer en operaciones aritmáticas, afectando o modificando con la operación a cada uno de los elementos del vector. Por ejemplo:

```
# Visualizando el contenido de vec_3
vec_3
## [1] 9 8 7 9 8 7

# Multiplicando cada uno de los elementos de vec_3 por 10
vec_3 * 10
## [1] 90 80 70 90 80 70

# Sumando 2 a cada uno de los elementos de vec_3
vec_3 + 2
## [1] 11 10  9 11 10  9
```

## 5.8 Operaciones relacionales en un vector

Al aplicar operaciones relacionales, obtenemos un vector de TRUE y FALSE. Por ejemplo:

```
# Visualizando el contenido de edades
edades
## Ana Luis Juan Paty Jair Mary
```

## Introducción a R y RStudio

```
## 25 8 52 18 0 93

# Obteniendo las edades mayores o iguales a 18
edades >= 18
## Ana Luis Juan Paty Jair Mary
## TRUE FALSE TRUE TRUE FALSE TRUE
```

Si queremos visualizar, solamente los elementos que cumplen la condición entonces utilizamos la siguiente expresión:

```
# Visualizando el contenido de edades
edades
## Ana Luis Juan Paty Jair Mary
## 25 8 52 18 0 93

# Obteniendo las edades mayores o iguales a 18
edades[edades >= 18]
## Ana Juan Paty Mary
## 25 52 18 93
```

Algunas funciones que se pueden aplicar a los vectores son:

```
mean(edades) # promedio de los valores del vector
## [1] 32.66667
max(edades) # máximo de los valores del vector
## [1] 93
min(edades) # mínimo de los valores del vector
## [1] 0
sum(edades) # suma de los elementos del vector
## [1] 196
length(edades) # longitud del vector
## [1] 6
```

Nota: Las matrices son objetos de dos dimensiones que contienen renglones y columnas. Al igual que en el caso de los vectores en las matrices sólo podemos almacenar datos del mismo tipo (por ejemplo numéricos o caracteres).

## 5.9 Ejercicios 6

1. Para cada una de las siguientes asignaciones, indica, de qué tipo es el vector resultante:
  - a. N<-c(5,"cinco")
  - b. Solucion<- c(TRUE,FALSE,TRUE)
  - c. Res<-(uno,1)
2. Genere un vector con las mediciones del tamaño (superficie), de por lo menos 5 planetas diferentes.
  - a. Agregue los nombres a cada planeta usando la función names.
  - b. Calcule del tamaño el: promedio, mínimo y máximo.
  - c. Agregue dos planetas más, junto con sus datos de superficie.

3. Escribe una expresión que simule el lanzamiento de una moneda 10 veces. Obtén como resultado de la expresión un vector de valores “cara” y “cruz”. Cuenta las ocurrencias de cara y cruz, y la proporción de cada valor. Tip, revise la función sample.
4. Crea un vector aleatorio de 10 elementos y selecciona aquellos elementos del vector que ocupan índices impares. Usa seq para expresar el vector de índices impares y la función rnorm para generar los números aleatorios.

Las respuestas se encuentran disponibles en: [Solución Ejercicios 6](#)

## 6 Factores

Los factores son variables categóricas que pueden ser numéricas o de tipo carácter. Un ejemplo clásico de una variable categórica es el “genero”, ya que sólo toma dos valores: “femenino” o “masculino”. Para crear un objeto de tipo factor se utiliza la función `factor()` sobre un vector. Por ejemplo:

```
Tejidos <- factor(c("higado", "pancreas", "higado", "pancreas", "higado"))
Tejidos
## [1] higado pancreas higado pancreas higado
## Levels: higado pancreas
Experimentos <- factor(c("WT", "Control", "Control", "WT", "WT", "Control"))
Experimentos
## [1] WT      Control Control WT      WT      Control
## Levels: Control WT
```

Como podemos observar `levels` contiene las diferentes categorías del factor. A cada categoría R le asigna un número identificador por orden alfabético. Para ver las categorías de un objeto tipo factor se utiliza el comando `levels()`. Por ejemplo:

```
levels(Tejidos)
## [1] "higado"   "pancreas"
```

### 6.1 La función `summary` en factores

En el caso de los factores, la función `summary` cuantifica el número de elementos por categoría.

```
summary(Tejidos)
##    higado pancreas
##          3         2
```

## 7 Dataframes

Los `dataframe` son objetos de dos dimensiones que contienen renglones y columnas. Este tipo de objeto es muy similar a las matrices, pero su principal diferencia es que los `dataframe` pueden almacenar diferentes tipos de datos, es decir, cada columna puede ser de un tipo distinto. En

## Introducción a R y RStudio

general un **dataframe** se crea al hacer uso de la función **read.table()**, que se encarga de leer un archivo. Sin embargo, también se puede generar con la función **data.frame()**. Por ejemplo:

```
# Podemos hacer un data.frame con los
# vectores definidos en la sección de vectores

asegurados<-data.frame(edades=c(25,8,52,18,0,93),
                        nombres=c("Ana","Luis","Juan","Paty","Jair","Mary"),
                        casado=c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE))

asegurados
##   edades nombres casado
## 1      25     Ana  TRUE
## 2       8    Luis FALSE
## 3      52    Juan  TRUE
## 4      18    Paty FALSE
## 5       0     Jair FALSE
## 6      93    Mary FALSE
```

También podemos generar primero los vectores y posteriormente utilizarlos en el dataframe

```
ed<-c(25,8,52,18,0,93)
nom<-c("Ana","Luis","Juan","Paty","Jair","Mary")
cas<-c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)
asegurados<-data.frame(edades=ed,nombres=nom,casado=cas)
```

Como podemos observar, la primera columna del **dataframe** contiene valores numéricos consecutivos, estos valores representan los nombres de cada uno de los renglones (los cuales deben ser valores únicos, es decir, no puede haber nombres repetidos). Si al momento de generar el **dataframe**, los nombres de los renglones no son proporcionados, por default se asignaran valores numéricos. Para poner nombres al momento de generar el **dataframe**, agregamos el parámetro **row.names**, Por ejemplo:

```
asegurados<-data.frame(edades=c(25,8,52,18,0,93),
                        nombres=c("Ana","Luis","Juan","Paty","Jair","Mary"),
                        casado=c(TRUE,FALSE,TRUE,FALSE,FALSE,FALSE),
                        row.names=c("10-20","15-30","10-18","10-09","15-12","12-32"))

asegurados
##   edades nombres casado
## 10-20      25     Ana  TRUE
## 15-30       8    Luis FALSE
## 10-18      52    Juan  TRUE
## 10-09      18    Paty FALSE
## 15-12       0     Jair FALSE
## 12-32      93    Mary FALSE
```

## 7.1 Manejo de dataframes

Para acceder a los elementos de un data.frame, usamos corchetes cuadrados. Como podemos observar en la siguiente figura, los elementos de un vector están ordenados y constan de un índice para los renglones y otro para las columnas. Por lo tanto para hacer referencia a algún elemento, colocamos en el corchete los índices del elemento(s) al que queremos acceder.

**asegurados =**

		edades	nombres	casado		
		10-20	25	Ana	TRUE	1
		15-30	8	Luis	FALSE	2
		10-18	52	Juan	TRUE	3
		10-09	18	Paty	FALSE	4
		15-12	0	Jair	FALSE	5
		12-32	93	Mary	FALSE	6
			1	2	3	

*nombres de columnas*

*índices de columnas*

**asegurados [índice renglones , índice columnas ]**  
**asegurados [c(índice renglones) , c(índice columnas)]**

```

asegurados[1,1]    # Obtiene el elemento de la columna 1 y renglón 1
## [1] 25
asegurados[5,1]    # Obtiene el elemento del renglón 5 columna 1
## [1] 0
asegurados[,c(1,3)] # Obteniendo elementos no consecutivos
##      edades casado
## 10-20     25   TRUE
## 15-30      8 FALSE
## 10-18     52  TRUE
## 10-09     18 FALSE
## 15-12      0 FALSE
## 12-32     93 FALSE
asegurados[2:5,]   # Obteniendo elementos consecutivos
##      edades nombres casado
## 15-30      8    Luis FALSE
## 10-18     52   Juan  TRUE
## 10-09     18   Paty FALSE
## 15-12      0   Jair FALSE
asegurados[,-c(3)] # Excluyendo la columna 3
##      edades nombres
## 10-20     25     Ana
## 15-30      8    Luis
## 10-18     52   Juan
## 10-09     18   Paty
## 15-12      0   Jair
## 12-32     93   Mary

```

## Introducción a R y RStudio

```
asegurados[-c(1,3),] # Excluyendo el renglón 1 y 3
##      edades nombres casado
## 15-30      8    Luis FALSE
## 10-09     18    Paty FALSE
## 15-12      0    Jair FALSE
## 12-32     93   Mary FALSE
```

Las columnas del `dataframe` pueden ser accedidas de manera individual de tres formas diferentes.

- 1) Utilizando el símbolo `$` y el nombre de la columna que queremos extraer, si nosotros no le asignamos de manera explícita nombres a las columnas, R les asigna V1, V2, ..., de manera automática. Pero para estar seguros cómo se llaman las columnas utilizamos el comando `names(nombre_del_dataframe)` y no aparecerá el nombre de cada columna. Consultemos los nombres de nuestro `dataframe` `asegurados`.

```
names(asegurados)
## [1] "edades" "nombres" "casado"

asegurados$nombres # Extrayendo la columna nombres, por medio del operador $
## [1] "Ana"  "Luis" "Juan" "Paty" "Jair" "Mary"
```

- 2) Poniendo entre corchetes el nombre de la columna que queremos consultar, el nombre de la columna debe ir entre comillas.

```
asegurados ["nombres"]
##      nombres
## 10-20    Ana
## 15-30   Luis
## 10-18   Juan
## 10-09  Paty
## 15-12  Jair
## 12-32  Mary
```

- 3) Poniendo entre cohete el número o índice de la columna que queremos consultar.

```
asegurados[,2]
## [1] "Ana"  "Luis" "Juan" "Paty" "Jair" "Mary"
```

## 7.2 Algunas funciones aplicables a `dataframe`

Podemos utilizar algunas de las funciones que vimos para vectores y/o matrices (`mean`, `max`, `dim`, `nrow` y `ncol`, `names`, entre otras).

```
dim(asegurados) # Obtiene el número de renglones y columnas de la matriz
## [1] 6 3
head(asegurados) # Muestra los primeros 6 renglones
##      edades nombres casado
## 10-20      25    Ana  TRUE
## 15-30      8    Luis FALSE
## 10-18      52   Juan  TRUE
```

## Introducción a R y RStudio

```
## 10-09    18   Paty  FALSE
## 15-12     0   Jair  FALSE
## 12-32    93   Mary  FALSE
tail(asegurados) # Muestra los últimos 6 renglones
##      edades nombres casado
## 10-20    25   Ana   TRUE
## 15-30     8   Luis  FALSE
## 10-18    52   Juan  TRUE
## 10-09    18   Paty  FALSE
## 15-12     0   Jair  FALSE
## 12-32    93   Mary  FALSE
max(asegurados) # Calculando el máximo de los elementos
## Error in FUN(X[[i]], ...): only defined on a data frame with all numeric variables
rowSums(asegurados) # Suma de los elementos por renglon
## Error in rowSums(asegurados): 'x' must be numeric
colSums(asegurados) # Suma de los elementos por columna
## Error in colSums(asegurados): 'x' must be numeric
```

Como podemos observar al querer realizar operaciones numéricas (`max`, `min`, `rowSums`, `colSums`) sobre todo el `dataframe` nos marca un error, debido a que algunas de sus columnas no son de tipo numérico. En ese caso se pueden seleccionar sólo las columnas numéricas donde se aplicarán dichos operadores.

### 7.3 Operaciones relacionales en `data.frames`

De la misma forma que con los vectores, con los `dataframes` podemos hacer algunas consultas directas sobre todos los elementos o un conjunto de ellos, utilizando los **Operadores relacionales** (mayor que, igual que, etc.) y los Operadores lógicos (and, or, not). Al aplicar estas operaciones obtenemos los resultados en términos lógicos, es decir, TRUE y FALSE por ejemplo:

```
# Obteniendo del data.frame asegurados los asegurados,
# mayores de 18 años
asegurados[,1] >= 18
## [1] TRUE FALSE TRUE TRUE FALSE TRUE
```

Como podemos observar los resultados de la comparación aparecen en términos de verdadero o falso, es decir, son valores lógicos. Si queremos acceder a los valores de aquellos elementos que se evalúan de manera verdadera, lo hacemos de la siguiente manera:

```
asegurados[asegurados[,1] >= 18, ]
##      edades nombres casado
## 10-20    25   Ana   TRUE
## 10-18    52   Juan  TRUE
## 10-09    18   Paty  FALSE
## 12-32    93   Mary  FALSE
```

Es decir, colocamos en corchetes la condición, indicando que queremos los valores de los elementos que cumplen esa condición.

### 7.4 Ejercicios 7

1. Con los siguientes vectores, genere un **dataframe**, llamado planetas.

```
name = c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")
type = c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
        "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant")
diameter = c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
rotation = c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
rings = c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE)
```

2. Obenga el promedio de los valores de las columnas **diameter** y **rotation**
3. Aplique la función **summary** al objeto planetas y vea que sucede.
4. obtenga:
  - a. Todos los planetas con un diámetro (diameter) mayor a 1.
  - b. Los planetas que tengan anillos (rings) y que el valor de rotación (rotation) sea menor o igual 0.5

Las respuestas se encuentran disponibles en: [Solución Ejercicios 7](#)

## 8 Lectura de archivos

Para lectura y escritura de archivos R utiliza por defecto el directorio de trabajo, a menos que le digamos de manera explícita la dirección donde se leerán o escribirán los archivos. Recordemos que si queremos saber en qué directorio estamos, utilizamos el comando **getwd**, que vimos en la sección [¿Cuál es mi directorio de trabajo?](#) y para cambiar de directorio, lo haremos con el comando **setwd**, que vimos en la sección [¿Cómo cambio de directorio?](#).

Con R podemos leer archivos de texto (ASCII) utilizando las siguientes funciones:

- **read.table** (que tiene algunas variantes)
- **scan**

Se pueden leer otro tipo de formatos como son Excel, SPSS, entre otros, pero las funciones que se necesitan para la lectura de estos archivos no vienen dentro del paquete básico de R, se requiere la instalación de otros [Paquetes de R](#).

### 8.1 La función **read.table**

Por medio de la función **read.table** leemos archivos en formato tabular y se crea un **dataframe** para su almacenamiento. Los argumentos de la función y sus valores por defecto los podemos ver [Consultando la ayuda](#).

```
?read.table
```

```
## read.table                  package:utils          R Documentation
## 
## Data Input
##
```

## Introducción a R y RStudio

```
## Description:  
##  
##   Reads a file in table format and creates a data frame from it,  
##   with cases corresponding to lines and variables to fields in the  
##   file.  
##  
## Usage:  
##  
##   read.table(file, header = FALSE, sep = "", quote = "\'",  
##               dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),  
##               row.names, col.names, as.is = !stringsAsFactors,  
##               na.strings = "NA", colClasses = NA, nrows = -1,  
##               skip = 0, check.names = TRUE, fill = !blank.lines.skip,  
##               strip.white = FALSE, blank.lines.skip = TRUE,  
##               comment.char = "#",  
##               allowEscapes = FALSE, flush = FALSE,  
##               stringsAsFactors = default.stringsAsFactors(),  
##               fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

## 8.2 Parámetros más comunes de `read.table`

Por ejemplo, tenemos el archivo `conteos.txt` y vamos a ver qué características tiene, en términos de 1) si tiene nombres de columnas, 2) cómo están separados sus campos (tabulador, espacio en blanco, comas, etc.), entre otras, con la finalidad de seleccionar los parámetros necesarios para leer el archivo.

```
external_gene_id    Mutant2_1    Mutant2_2    Mutant2_3    wildType_1    wildType_2    wildType_n_3  
# Este es un comentario  
2L52.1    422    465    441    278    262    256  
2RSSE.1   1644   1849   1841   922    998    928  
2RSSE.2   82     74     75     30     42     59  
pot-3     154    139    161    190    181    174  
nas-6     305    297    333    266    248    275  
4R79.2    59     44     58     32     41     44  
6R55.2    0      0      1      2      3      2  
sri-20    7      4      5      4      6      3  
spe-10    13     8      3      11    10     3  
# Este es otro comentario  
AC3.12    0      0      0      0      0      0  
ugt-49    758    755    732    835    872    837  
abu-1     117    146    164    143    118    144
```

Una vez que visualizamos el archivo podemos seleccionar los parámetros necesarios para leerlo, para leer el archivo utilizaríamos la siguiente instrucción:

```
countTable<-read.table("/Data/conteos.txt",sep = "\t", header = T, row.names = 1)
```

Aunque nuestro archivo contiene nombres de columnas y nombres de renglones, la función `read.table` no los identifica por defecto, para ello necesitamos definir algunos parámetros, los cuales podemos ver en la ayuda. En general los parámetros más utilizados son:

```
header = T      # Para indicar que el primer renglón del archivo contienen  
los nombres de las columnas  
sep            # Para indicar qué tipo de separador tenemos, por lo general es tabulador  
row.names      # Para indicar en qué columna se encuentran los nombres de los renglones  
comment.char   # Para indicar el símbolo que se usará para poner comentarios.  
stringsAsFactors # Para indicar si los valores de tipo carácter son considerados como  
factores o no.
```

Nota: Cuando el parámetro `header=FALSE`, las columnas son etiquetadas por defecto como V1, V2, ... Vn. Por otra parte, si no se especifica el parámetro `row.names`, los renglones son etiquetados con valores secuenciales que inician en "1".

### 8.3 Algunas funciones para visualizar los datos

Para visualizar los datos tenemos funciones como:

```
# Visualizando los primeros renglones  
head(countTable)  
##           Mutant2_1 Mutant2_2 Mutant2_3 wildType_1 wildType_2 wildTypen_3  
## 2L52.1        422       465       441       278       262       256  
## 2RSSE.1       1644      1849      1841       922       998       928  
## 2RSSE.2        82        74        75        30        42        59  
## pot-3         154       139       161       190       181       174  
## nas-6         305       297       333       266       248       275  
## 4R79.2         59        44        58        32        41        44  
# Visualizando el numero de renglones y columnas  
dim(countTable)  
## [1] 20259      6  
# Visualizando los ultimos renglones  
tail(countTable)  
##           Mutant2_1 Mutant2_2 Mutant2_3 wildType_1 wildType_2 wildTypen_3  
## ZK973.9        212       241       248       251       227       272  
## ceh-45         27        40        32        25        17        15  
## ZK993.2        38        25        53        21        29        21  
## ZK994.1         1         0         0         1         1         1  
## pxn-1         2149      2254      1925      1398      1429      1325  
## ZK994.6        21        11        19        20        11        14  
# Visualizando los nombres de las columnas  
names(countTable)  
## [1] "Mutant2_1"    "Mutant2_2"    "Mutant2_3"    "wildType_1"   "wildType_2"  
## [6] "wildTypen_3"  
# Visualizando los primeros nombres de renglones  
head(row.names(countTable))  
## [1] "2L52.1"     "2RSSE.1"     "2RSSE.2"     "pot-3"      "nas-6"      "4R79.2"
```

### 8.4 Variantes de `read.table`

Las variantes de `read.table` que pueden ser de utilidad, ya que contienen algunos parámetros por defecto diferentes, son:

```
read.csv(file, header = TRUE, sep = ",", quote = "", dec = ".", fill = TRUE, ...)  
read.csv2(file, header = TRUE, sep = ";", quote = "", dec = ",", fill = TRUE, ...)  
read.delim(file, header = TRUE, sep = "\t", quote = "", dec = ".", fill = TRUE, ...)  
read.delim2(file, header = TRUE, sep = "\t", quote = "", dec = ",", fill = TRUE, ...)
```

### 8.5 La función `scan`

La función `scan` permite leer archivos especificando el tipo de datos que se van a leer, por lo tanto sólo puede leer archivos con valores del mismo tipo, es decir, sólo numéricos o sólo de tipo carácter. El resultado de la lectura del archivo queda en formato de vector o lista, en este sentido, no se recomienda utilizar esta función para hacer lectura de datos que se encuentre en formato de tabla. Dado que para los fines de este curso sólo haremos lecturas de tablas no profundizaremos en la función `scan`. Sin embargo, pueden consultar la descripción de la misma en la ayuda.

## 9 Escritura de archivos

La escritura de datos a un archivo se realiza por medio de dos funciones:

- `write.table` y las variantes `write.csv` y `write.csv2`
- `write`

### 9.1 La función `write.table`

La función `write.table` nos permite guardar los datos de una `dataframe` en un archivo, con un formato de tabla. `write.table` tiene una serie de argumentos para indicar, cómo queremos que se almacenen los datos. Por ejemplo, si se incluyen los nombres de renglón, de columna, el tipo de separador, entre otros. Los argumentos de la función y sus valores por default los podemos consultar en la ayuda.

```
?write.table
```

```
Usage:
```

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", * eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape", "dou  
ble"), fileEncoding = "")
```

Los parámetros más utilizados son los que se encuentran en letras itálicas:

```
sep: Para indicar el tipo de separador
```

```
row.names: Para indicar si la tabla se guarda con nombres de renglones o no
```

## Introducción a R y RStudio

`col.names: Para indicar se la tabla se guarda con nombres de columnas o no`

Por ejemplo, utilizando el objeto `countTable`, que leímos anteriormente, eliminaremos los renglones de ceros y guardaremos los resultados en un archivo llamado `conteosSinCeros.txt`, utilizando la función `write.table`.

Primero verificaremos cuántos renglones tiene el objeto `countTable`, posteriormente eliminaremos los renglones con ceros y veremos con cuántos renglones nos quedamos. Finalmente, guardaremos los resultados en el archivo.

```
# Verificando el número de renglones
nrow(countTable)
## [1] 20259

# Eliminando los renglones de ceros
countTableSinCeros<- countTable [rowSums(countTable)>0,]

# Verificando el número de renglones resultantes
nrow(countTableSinCeros)
## [1] 18878

# Guardando la tabla de conteos, donde se eliminaron
# los renglones con solo ceros

write.table(countTableSinCeros, "conteosSinCeros.txt", quote=FALSE, sep="\t")
```

El archivo `conteosSinCeros.txt`, debe tener 18879 renglones y 6 columnas, considerando que guardamos los nombres de las columnas.

## 9.2 La función `write`

La función `write`, se utiliza generalmente para almacenar objetos de tipo vector o matriz. Para efectos de este curso, no haremos uso de esta función, dado que trabajaremos en general con tablas y por lo tanto utilizaremos `write.table`. Sin embargo, se puede consultar la descripción de `write` en la ayuda.

## 9.3 Ejercicios 8

1. Utilizando la función `write.table`, almacene en un archivo el objeto `planetas`, creado en [Ejercicios 7](#).

La respuesta se encuentran disponibles en: [Solución Ejercicios 8](#)

## 10 Gráficas con `ggplot2`

La visualización es una tarea fundamental durante el análisis de datos, sobre todo cuando se tienen grandes cantidades de información. Una buena representación gráfica, puede mostrar cosas que no se esperaba o permitir plantear nuevas preguntas sobre los datos. Pero, también

## Introducción a R y RStudio

podría indicar que se está haciendo la pregunta incorrecta o que se requiere una nueva recolección de datos. Las representaciones gráficas pueden ser sorprendentes, sin embargo, siempre requieren a un ser humano para su interpretación.

R es un herramienta muy potente para el análisis estadístico de datos y la graficación. Existen diversos paquetes para graficar, tres de los más utilizados son:

- Graphics
- lattice
- ggplot2

En este curso, nos centraremos en **ggplot2**, que es muy potente y versátil para realizar gráficos con calidad para publicación.

### 10.1 ¿Qué es ggplot2?

**ggplot2** es un paquete para crear gráficas y forma parte del paquete **tidyverse**. **ggplot2** fue creado por **Hadley Wickham** en 2009, basado en el libro **Grammar of graphics** de **Leland Wilkinson**, de hecho el término **gg**, viene justamente de “grammar of graphics”\*\*. Este libro presenta una serie de ideas sobre cómo debería hacerse la representación gráfica de datos estadísticos.

La idea principal de esta propuesta es que todos los gráficos pueden generarse mediante un lenguaje regular con una determinada sintaxis. Es decir, que podemos tener una serie de reglas comunes para crear una representación visual.

### 10.2 ¿Cómo se generan las gráficas en R?

Las gráficas en R operan sobre lo que se conoce como un “*modelo de pintor*”, es decir, no se dibujan en una sola operación, sino que requieren varias “*capas*” de trazado diferentes, colocadas una encima de la otra, como se muestra en la Figura 1.

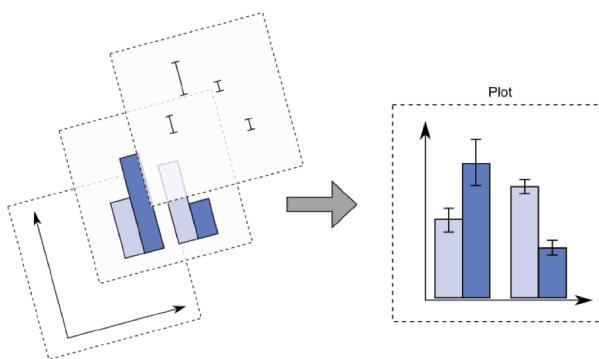


Figura 15: **Modelo de pintor** (*Introduction to ggplot2*, Babraham bioinformatics, 2015.)

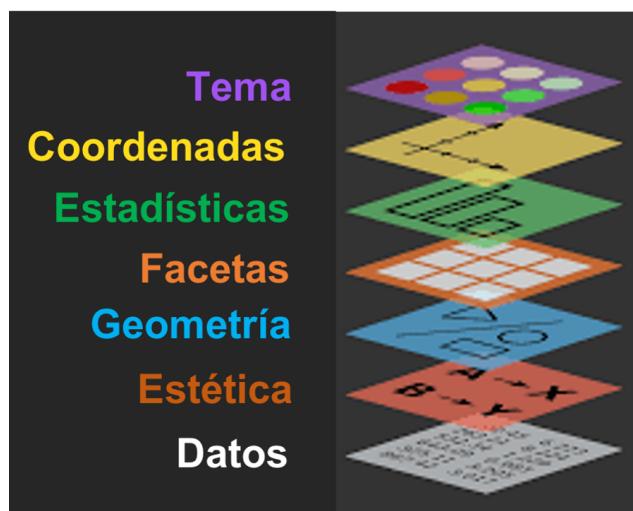
En este sentido, las capas inferiores están obscurecidas por las capas superiores, permitiendo mezclar, enmascarar y superponer objetos. Es importante mencionar que no se pueden deshacer las operaciones que se realizan a la gráfica, es necesario volver a ejecutar el código desde cero.

### 10.3 Elementos principales de las gráficas de ggplot2

Los componentes de una gráfica en ggplot son los siguientes:

- **Datos:** conjunto de datos representados en un data.frame
- **Estética:** la estética que determina, cómo se representarán visualmente los datos en términos ejes (x,y), color, tamaño, forma, etc.
- **Geometría:** geometría que se utilizará para la representación de los datos a graficar (puntos, líneas, polígonos, etc.)
- **Facetas:** para generar gráficas en multipaneles.
- **Estadísticas:** para colocar estadíticos, como barras de ajuste.
- **Coordenadas:** para cambiar transformar el sistema de coordenadas, por ejemplo a coordenadas polares.
- **Temas:** características estéticas de la gráfica, por ejemplo, el fondo, las líneas de la cuadrícula, entre otras.

Las tres primeras capas (Datos, estética y geometría) son las más importantes y son necesarias y las capas restantes se utilizan solo si se requieren.



Con ggplot se pueden generar diversos tipos de gráficos, como son los gráficos de barras, de puntos, de líneas, entre otros. Para hacer una consulta de estos gráficos se puede revisar las hojas de consulta rápida (“*cheatsheet*”), que se encuentran en el sitio de RStudio en la sección Recursos (<https://www.rstudio.com/resources/cheatsheets/>)

### 10.4 Instalando ggplot2

La instalación del paquete `ggplot2` se puede realizar, desde R, de dos maneras:

1. Instalando el paquete `tidyverse`, el cual incluye al paquete `ggplot2`.  
`install.package("tidyverse", dependencies=TRUE)`
2. Instalando solo `ggplot2`  
`install.package("ggplot2", dependencies=TRUE)`

### 10.5 Conjunto de datos `msleep`

Existen diversos conjuntos de datos en R con los que se puede explorar el tema de graficación, los cuales se pueden consultar utilizando el comando:

```
data()
```

Sin embargo, para este curso utilizaremos el conjunto de datos [`msleep`], el cual se encuentra por defecto en el paquete de `ggplot2`.

El `data.frame` contiene un conjunto de datos con los **habitos de sueño de diversos mamíferos**, los cuales están basados en artículo “**A quantitative, theoretical framework for understanding mammalian sleep**”, de V. M. Savage y G. B. West. *Proceedings of the National Academy of Sciences*, 104 (3):1051-1056, 2007.

### 10.6 Cargando `ggplot2` y visualizando los datos `msleep`

Para poder utilizar las funciones del paquete `ggplot2`, primero, debemos cargar la biblioteca `ggplot2`, utilizando la función `library`.

```
library(ggplot2)
```

Una vez cargado el paquete de `ggplot2`, podemos visualizar el conjunto de datos `msleep` y verificar los nombres de las columnas y el número de renglones y columnas.

- ¿Cuántas columnas y renglones tiene?
- ¿Cuáles son los nombres de las columnas?

```
# Obtener número de renglones y columnas.  
dim(msleep)  
## [1] 83 11  
  
# Obtener nombres de las columnas  
names(msleep)  
## [1] "name"          "genus"         "vore"          "order"         "conservation"  
## [6] "sleep_total"    "sleep_rem"     "sleep_cycle"   "awake"        "brainwt"  
## [11] "bodywt"  
  
# Visualizando los primeros renglones  
head(msleep)  
## # A tibble: 6 x 11  
##   name   genus vore order conservation sleep_total sleep_rem sleep_cycle awake  
##   <chr>  <chr> <chr> <chr> <dbl>      <dbl>      <dbl>      <dbl>  
## 1 Cheetah Acin~ carni Carn~ lc      12.1       NA       NA      11.9  
## 2 Owl mo~ Aotus omni Prim~ <NA>     17        1.8       NA       7  
## 3 Mounta~ Aplo~ herbi Rode~ nt     14.4       2.4       NA      9.6  
## 4 Greate~ Blar~ omni Sori~ lc     14.9       2.3      0.133     9.1  
## 5 Cow     Bos   herbi Arti~ domesticated 4        0.7      0.667    20  
## 6 Three~~ Brad~ herbi Pilo~ <NA>    14.4       2.2      0.767    9.6  
## # ... with 2 more variables: brainwt <dbl>, bodywt <dbl>
```

## Introducción a R y RStudio

Para visualizar el significado de cada una de las columnas del conjunto de datos `msleep`, podemos utilizar la ayuda.

```
# Obtener la descripción del conjunto msleep
help(msleep)

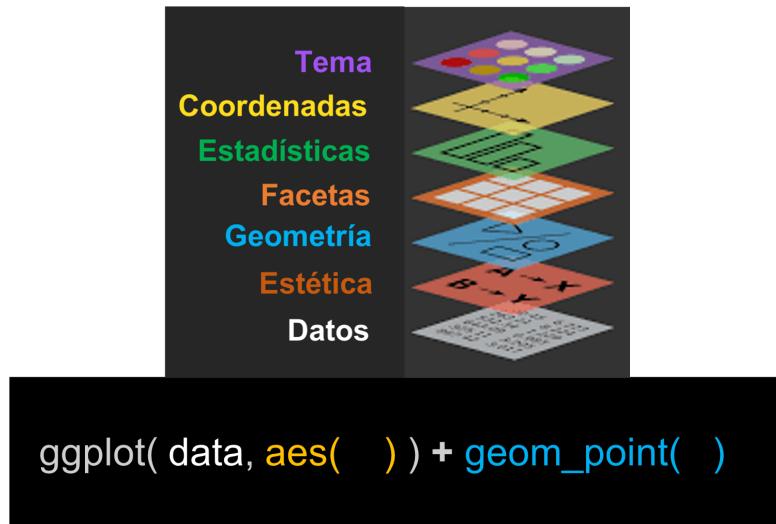
## Usage
##
## msleep
##
## Format
##
## A data frame with 83 rows and 11 variables:
##
##   name
##   common name
##   genus
##   vore
##   carnivore, omnivore or herbivore?
##   order
##   conservation
##   the conservation status of the animal
##   sleep_total
##   total amount of sleep, in hours
##   sleep_rem
##   rem sleep, in hours
##   sleep_cycle
##   length of sleep cycle, in hours
##   awake
##   amount of time spent awake, in hours
##   brainwt
##   brain weight in kilograms
##   bodywt
##   body weight in kilograms
```

## 10.7 Primeros pasos en graficación

Supongamos que queremos **visualizar**, cuál es la **relación** que existe entre el **número de horas dormidas por día** (`sleep_total`) del animal y su **peso** (`bodywt`).. Para la graficación `ggplot2` tiene 2 comandos:

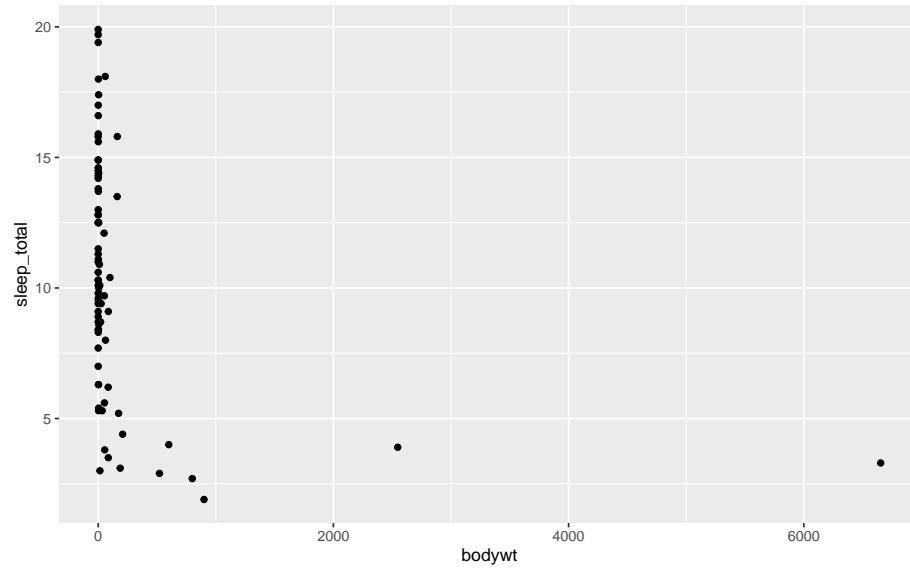
```
qplot()
ggplot()
```

Para este taller solo utilizaremos el comando `ggplot()`. Recoredemos cómo es la gramática de las gráficas.



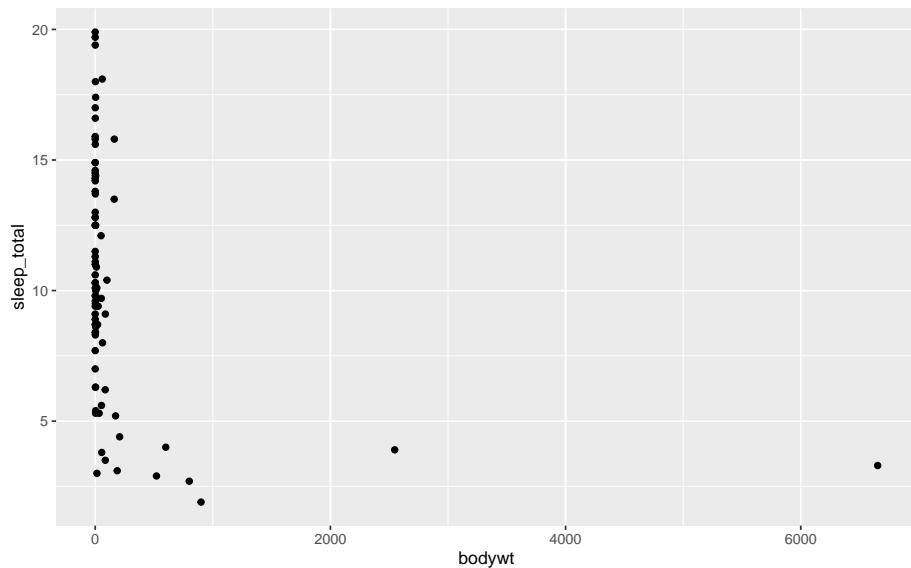
### 10.7.1 Mi primera gráfica

```
ggplot(data = msleep,  
       aes(x = bodywt, y = sleep_total)) +  
       geom_point()
```



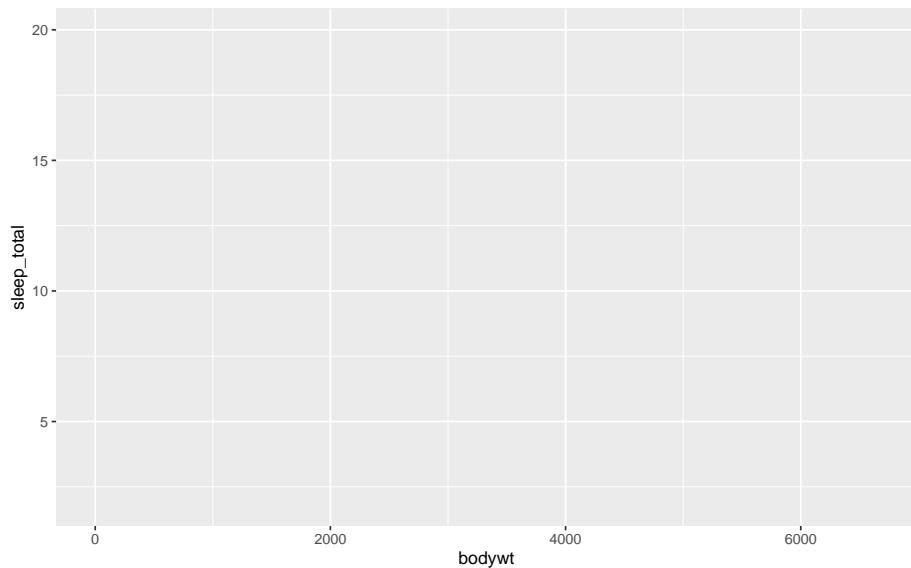
podemos omitir el `data=` y solo poner el nombre del data.frame donde se encuentran los datos.

```
ggplot(msleep,  
       aes(x = bodywt, y = sleep_total)) +  
       geom_point()
```



Por otra parte, ¿Qué sucede si quitamos la geometría de alguna de las instrucciones anteriores?

```
ggplot(msleep,  
       aes(x = bodywt, y = sleep_total))
```



No graficaría los datos, dado que no le dijimos cómo hacerlo, en este sentido, es importante mencionar que llamar sólo a la función `ggplot` no es suficiente para dibujar una figura. Necesitamos decirle a `ggplot` cómo queremos representar visualmente los datos, por eso agregamos la capa `geom`.

## 10.8 Ejercicios 9

1. De los datos DNase, que vienen en las bases de datos de R:

## Introducción a R y RStudio

- Explore los datos del dataframe, ¿cuántas columnas y renglones tiene?, ¿qué describe cada dato?
- Grafique la densidad (en el eje de las y) vs. la concentración.

La respuesta se encuentran disponibles en: [Solución Ejercicios 9](#)

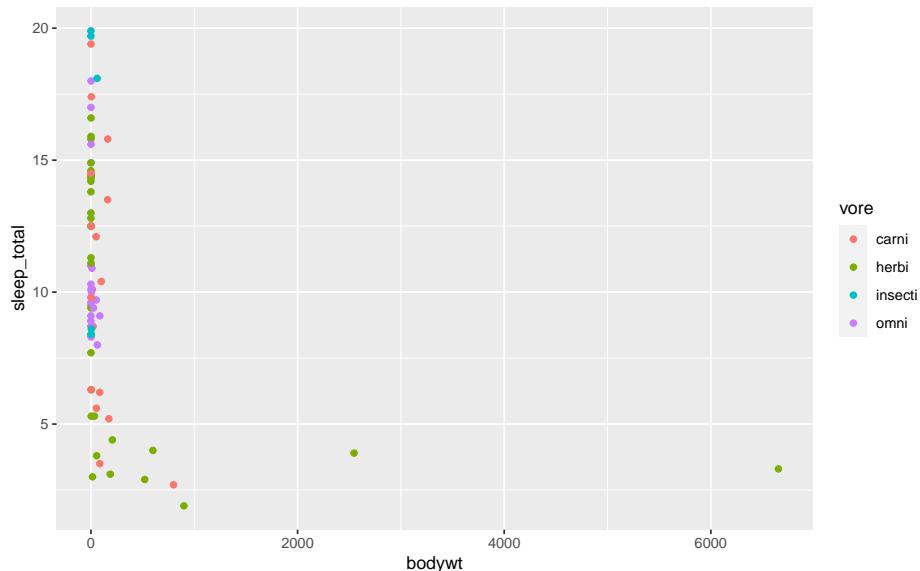
## 10.9 La estética (aes)

La estética (**aesthetic**) es una propiedad visual de los objetos en tu gráfica e incluye **características como el tamaño, la forma o el color de los mismos**. Usamos la palabra “*level*” para describir las propiedades estéticas. Color, tamaño y forma son ejemplos de *levels* que describen la estética.

### 10.9.1 Color

Por ejemplo, podemos **cambiar la estética de los puntos en términos del color**. Modificaremos el código de ejercicio anterior para colorear los puntos de acuerdo al tipo de alimentación (**vore**) de los animales.

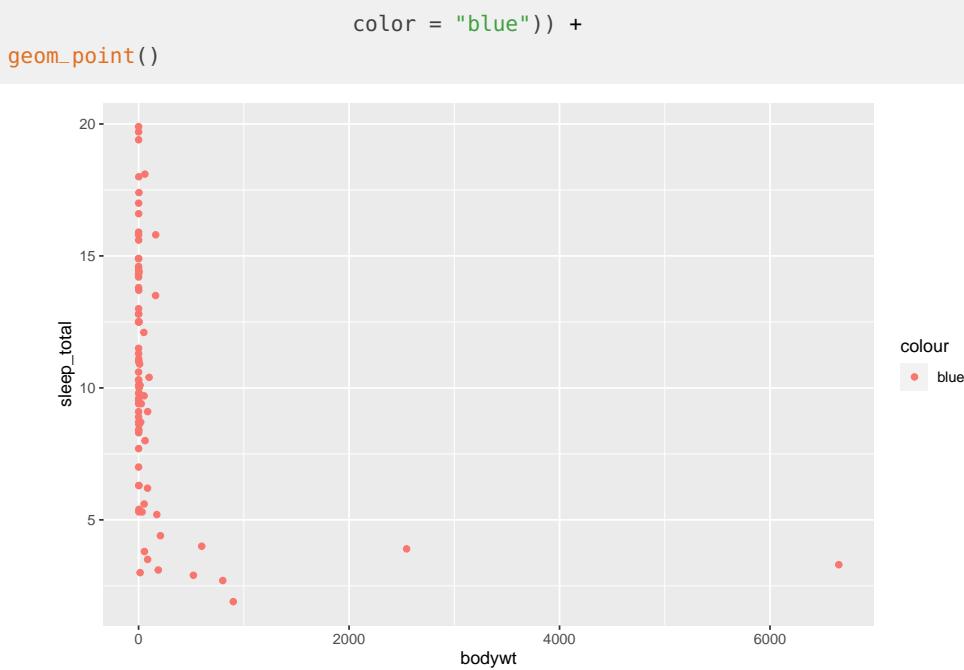
```
# Como vore contiene NA's, filtramos primero los renglones con NA's
keep_rows <- !(is.na(msleep$vore))
# Graficando en colores los puntos
ggplot(msleep[keep_rows, ], aes(x = bodywt, y = sleep_total,
                                 color = vore)) +
  geom_point()
```



¿Qué sucede si queremos cambiar el color de todos los puntos a azul? ¿Cómo lo haríamos? Podríamos pensar en agregar `color = "blue"` en la estética de los puntos. ¿Funcionaría?

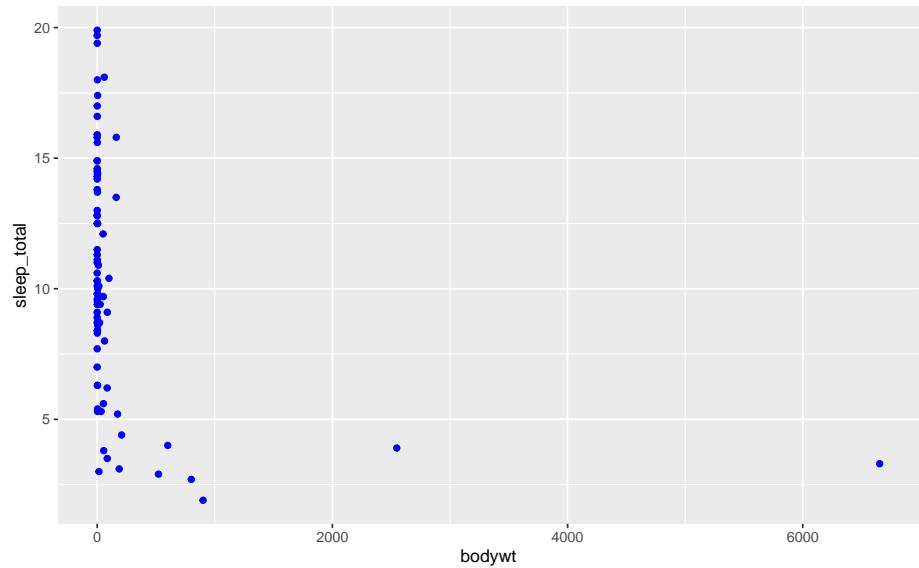
```
# Graficando en colores los puntos
ggplot(msleep, aes(x = bodywt, y = sleep_total,
```

## Introducción a R y RStudio



¿Por qué no funciona? Porque el color no está asociado a ninguna variable, es decir es un valor constante, por lo tanto simplemente movemos la especificación de color fuera de cualquier función `aes` y colocamos la especificación de color de manera local en `geom_point`, como se muestra a continuación:

```
# Graficando en colores los puntos  
ggplot(msleep, aes(x = bodywt, y = sleep_total)) +  
  geom_point(color="blue")
```



Para consultar los códigos de los colores, podemos poner:

```
colors()
```

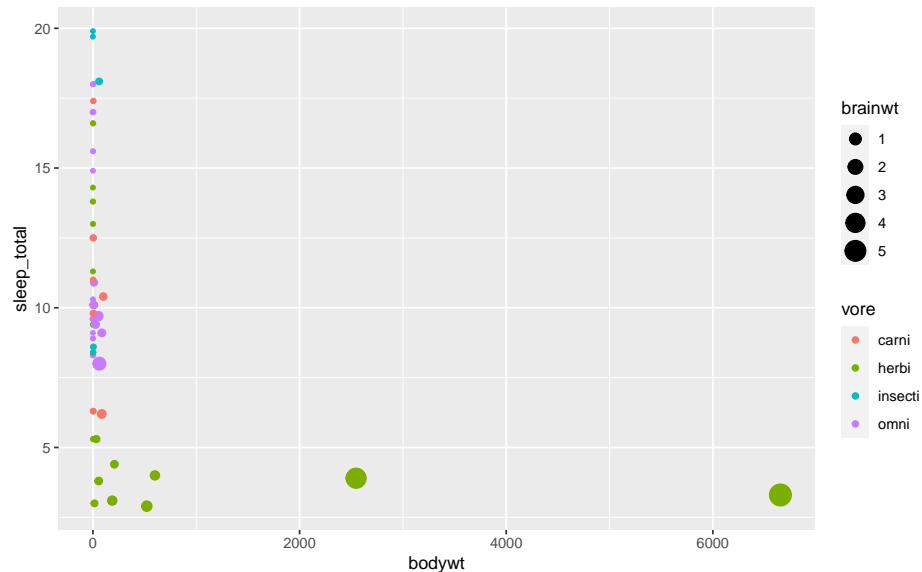
### 10.9.2 Tamaño, transparencia y forma

Otros niveles (“*levels*”) que podemos modificar en la estética son el **tamaño**, **la transparencia** y **la forma**. Para modificar el tamaño usamos `size`, para modificar la transparencia utilizamos `alpha` y para modificar la forma utilizamos `shape`. Las modificaciones las podemos realizar:

- De manera constante
- Asociado a los valores de una variable

Cambiemos ahora el tamaño de los puntos en función del peso del cerebro de los animales (`brainwt`). Dado que es un valor que está asociado a una variable, colocamos el parámetro `size` dentro de la función `aes`.

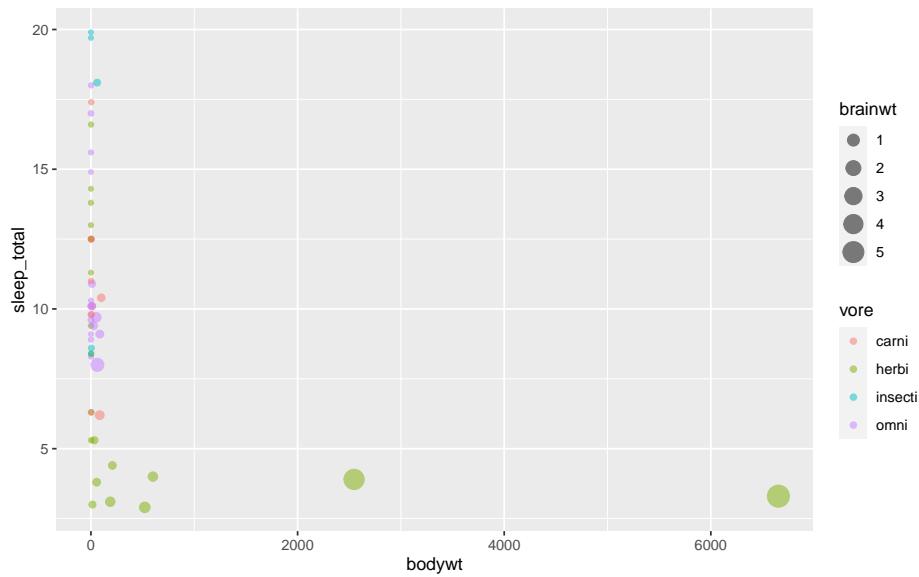
```
keep_rows <- !is.na(msleep$vore) & !is.na(msleep$brainwt)
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color=vore, size = brainwt)) +
  geom_point( )
```



Ahora cambiemos la transparencia de los puntos de una forma constante. Dado que es un valor constante, no está asociado a alguna variable, por lo tanto este nivel queda de manera local dentro de la geometría.

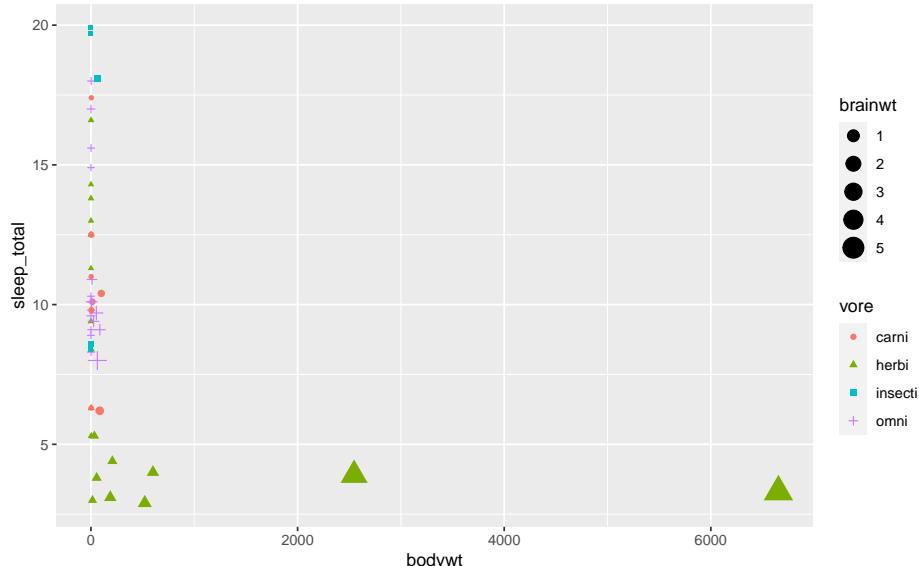
```
keep_rows <- !is.na(msleep$vore) & !is.na(msleep$brainwt)
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore, size = brainwt)) +
  geom_point(alpha=0.5)
```

## Introducción a R y RStudio

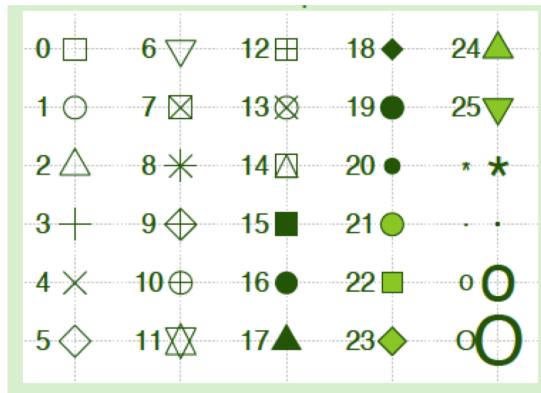


Finalmente, realicemos modificaciones en términos de la forma. Cambiemos ahora la forma de los puntos en función del tipo de alimentación (`vore`) de los animales. Dado que es un valor que está asociado a una variable, colocamos el parámetro `shape` dentro de la función `aes`.

```
keep_rows <- !is.na(msleep$vore) & !is.na(msleep$brainwt)
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore, size = brainwt,
            shape = vore)) +
  geom_point( )
```



Los símbolos de las formas tienen asociado un valor numérico. En la siguiente Figura se muestra los diversos símbolos. En este sentido podemos cambiar el valor de la forma a un valor constante asociado a un símbolo determinado.



Nota La la **transparencia y el tamaño**, se utilizan con variables **numéricas**, mientras el **color y la forma** se utilizan con variables **categóricas**.

### 10.9.3 Cambiando los colores y formas manualmente

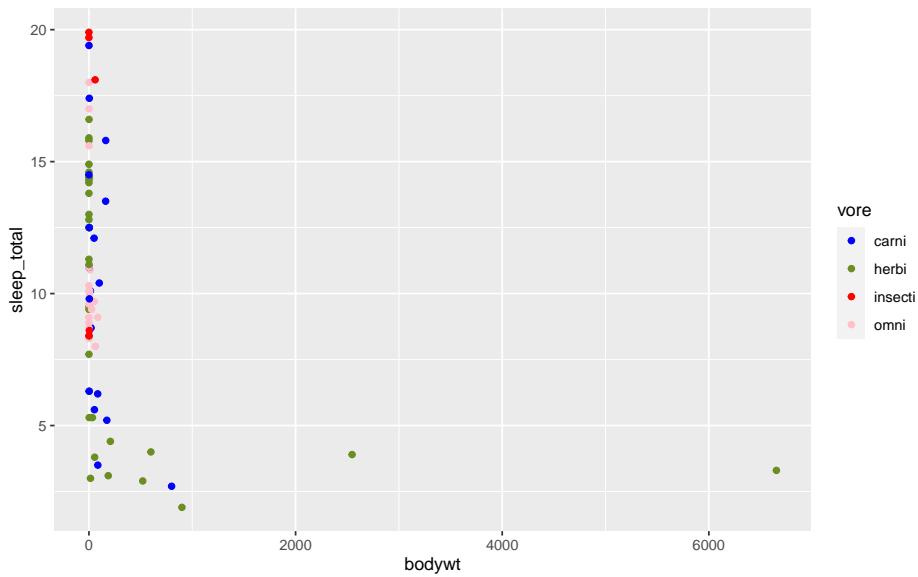
Por defecto cuando agregamos color a la gráfica los colores que se ponen ya están establecidos, lo mismo pasa con la forma, ya está definida. Sin embargo, podemos modificar estos valores, por medio de las siguientes capas:

1. `scale_color_manual` y
2. `scale_shape_manual`

Por ejemplo:

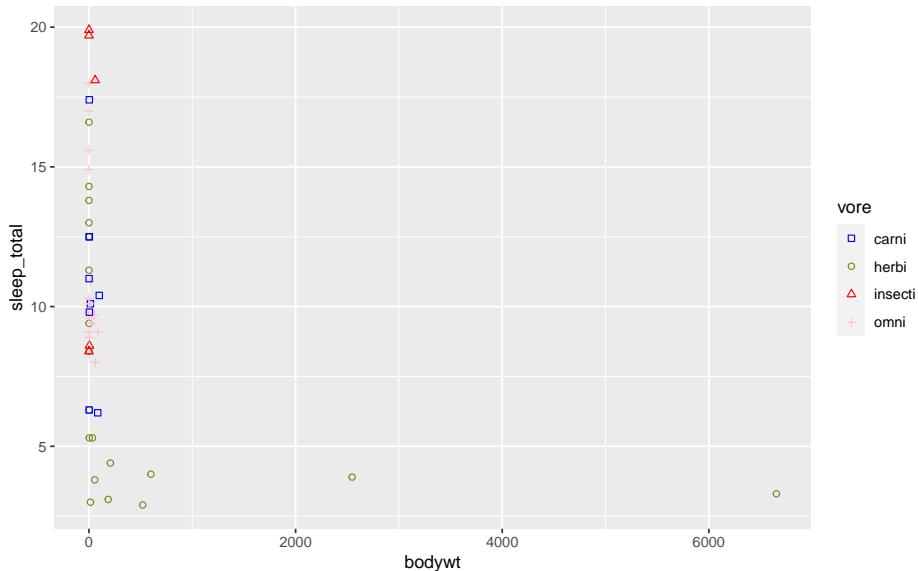
```
# Como vore contiene NA's, filtramos primero los renglones con NA's
keep_rows <- !(is.na(msleep$vore))
# Graficando en colores los puntos
ggplot(msleep[keep_rows, ], aes(x = bodywt, y = sleep_total,
                                 color = vore)) +
  geom_point() +
  scale_color_manual(values=c("blue","olivedrab","red","pink"))
```

## Introducción a R y RStudio



Recordemos que cada forma tiene un valor numérico asociado. Por lo tanto podemos establecer los símbolos de acuerdo a dicho valor numérico. Por ejemplo:

```
keep_rows <- !is.na(msleep$vore) & !is.na(msleep$brainwt)
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore, shape = vore)) +
  geom_point( ) +
  scale_color_manual(values=c("blue","olivedrab","red","pink")) +
  scale_shape_manual(values=c(0,1,2,3))
```



### 10.10 Ejercicios 10

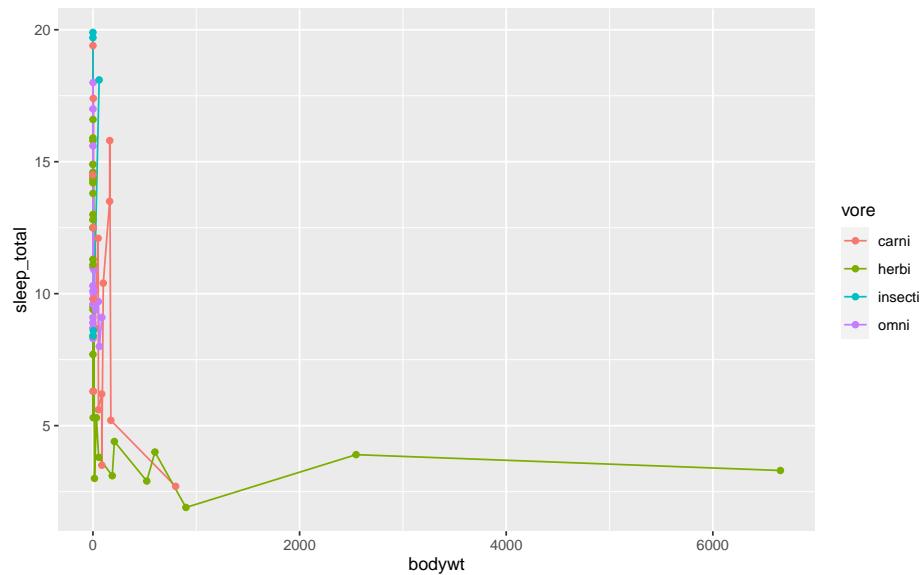
1. De la gráfica que realizamos en [Ejercicios 9](#), cambiar el color de los puntos con base en la columna `vore`. Además, colocar un símbolo distinto al punto y cambiar la transparencia a 0.4.

[Solución Ejercicios 10](#)

### 10.11 Más de una geometría en la gráfica

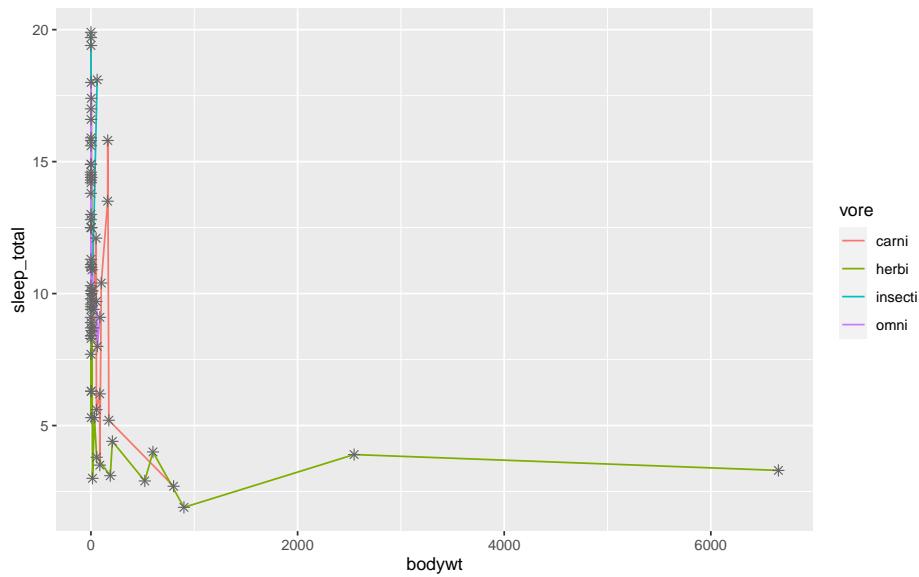
¿Cómo visualizaríamos puntos y líneas en la misma gráfica? Agregando dos geometrías al mismo tiempo, una para graficar puntos y otra para líneas. Por ejemplo:

```
keep_rows <- !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore)) +
  geom_line() +
  geom_point()
```



Recordemos que cada capa se dibuja encima de la capa anterior. En este ejemplo, los puntos se han dibujado sobre las líneas. Para visualizar mejor el orden de pintado de las capas, sacaremos de las opciones globales el color.

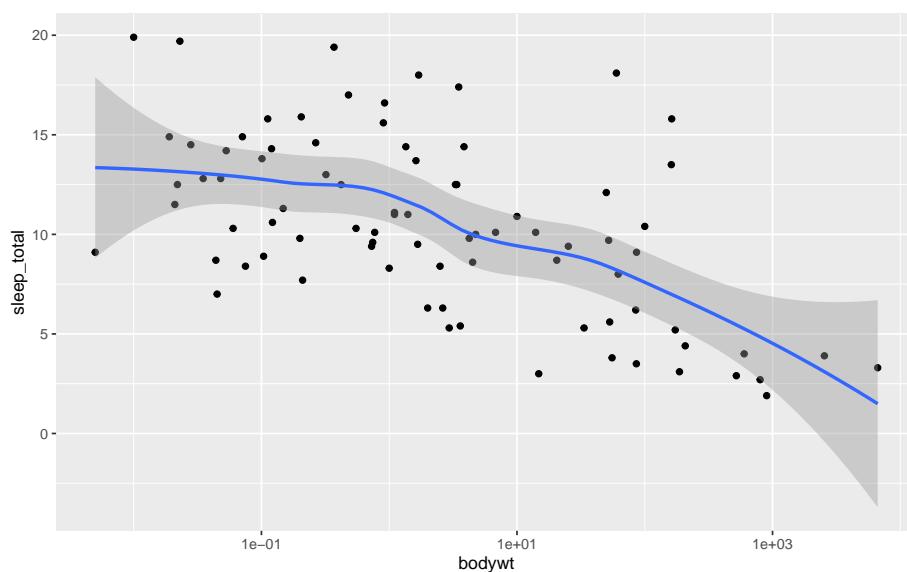
```
keep_rows <- !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total)) +
  geom_line(aes(color=vore)) +
  geom_point(color="grey40", shape=8, size=2)
```



## 10.12 Agregando la geometría de tendencia

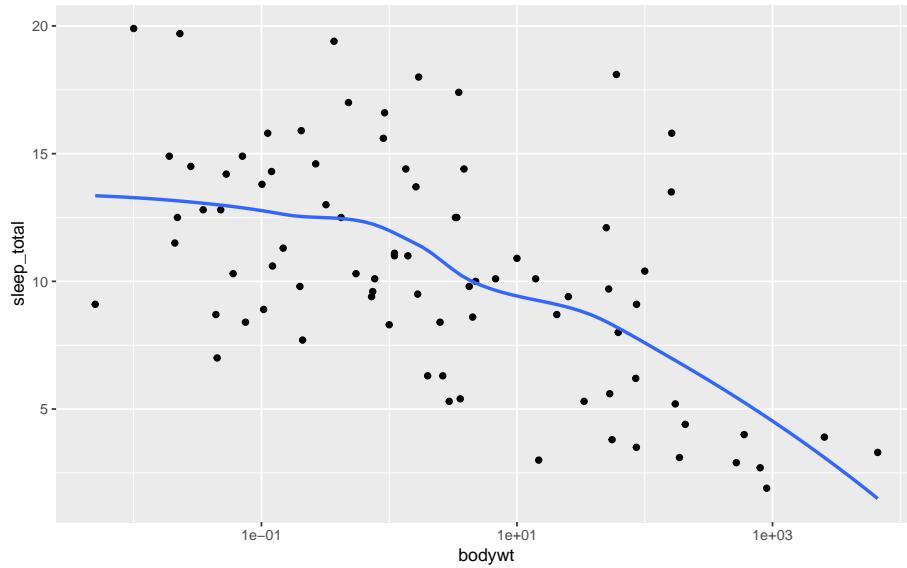
Otra tipo de graficas en las cuales podemos tener más de una geometría, son aquellas en las cuales incorporamos una línea de tendencia, para ver como se correlacionan los datos. Para agregar la línea de tendencia utilizamos la geometría `geom_smooth()`. Por ejemplo vamos a graficar la relación que existe entre el peso del animal y las horas de sueño.

```
ggplot(data = msleep,
       aes(x = bodywt, y = sleep_total)) +
  geom_point() +
  scale_x_log10() +
  geom_smooth()
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



También podemos cambiar el modelo de ajuste, por ejemplo podemos poner un ajuste lineal y eleminar el sombreado gris que representa el error estándar. Por ejemplo:

```
ggplot(data = msleep,  
       aes(x = bodywt, y = sleep_total)) +  
  geom_point() +  
  scale_x_log10() +  
  geom_smooth(method="lm", se=FALSE)  
## Warning: Ignoring unknown parameters: method  
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



### 10.13 Ejercicios 11

1. De los datos de `msleep`, graficar la relación que existe entre las características `bodywt` y `awake`, colorear los puntos con base en la característica `vore` y agregar la línea de tendencia (`geom_smooth`) general, es decir para todos los puntos.
2. Definir el tamaño de los puntos, con base a la característica `sleep_cycle`.

La respuesta se encuentran disponibles en: [Solución Ejercicios 3](#)

### 10.14 Gráficas en múltiples paneles

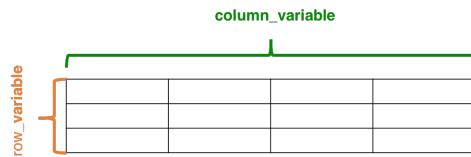
Hasta ahora hemos graficado todo en una sola gráfica, pero podemos dividir la gráfica en múltiple paneles. Para subdividir nuestras gráficas en paneles, utilizamos `facet`. Esta capa tiene dos tipos de funciones: `facet_grid( )` y `facet_wrap( )`, especificando las variables con las cuales se dividirán los datos. Para este cuso sólo veremos las gráficas de multipáneles con `facet_grid( )`, pero en el [Apéndice 3](#) puedes consultar la sintaxis para `facet_wrap( )`.

## Introducción a R y RStudio

### 10.14.1 `facet_grid()`

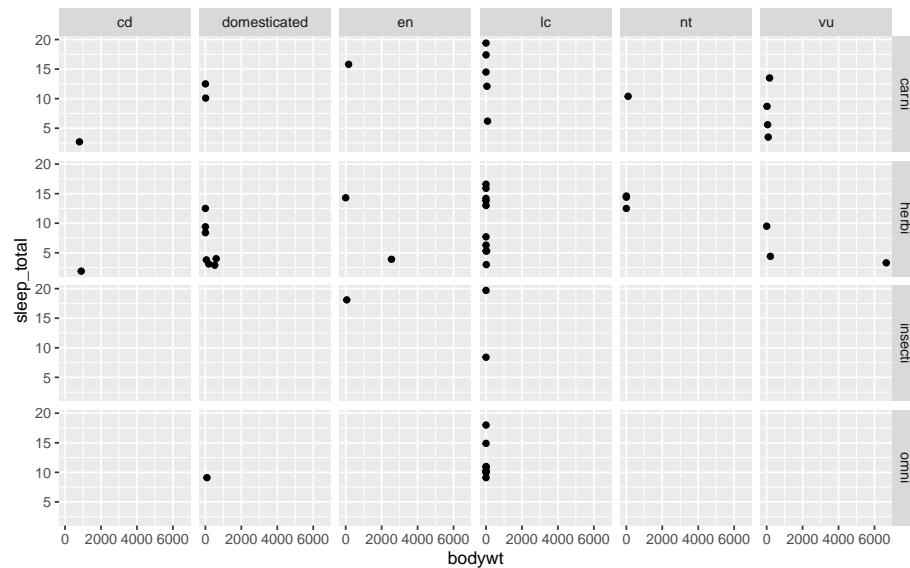
`facet_grid()` nos permite organizar los diversos paneles en renglones(vertical) y/o en columnas(horizontal) y tiene la siguiente sintaxis de `facet_grid(row_variable ~ column_variable)`. Para excluir una de las variables, la remplazamos con un punto(.)

`facet_grid(row_variable ~ column_variable)`



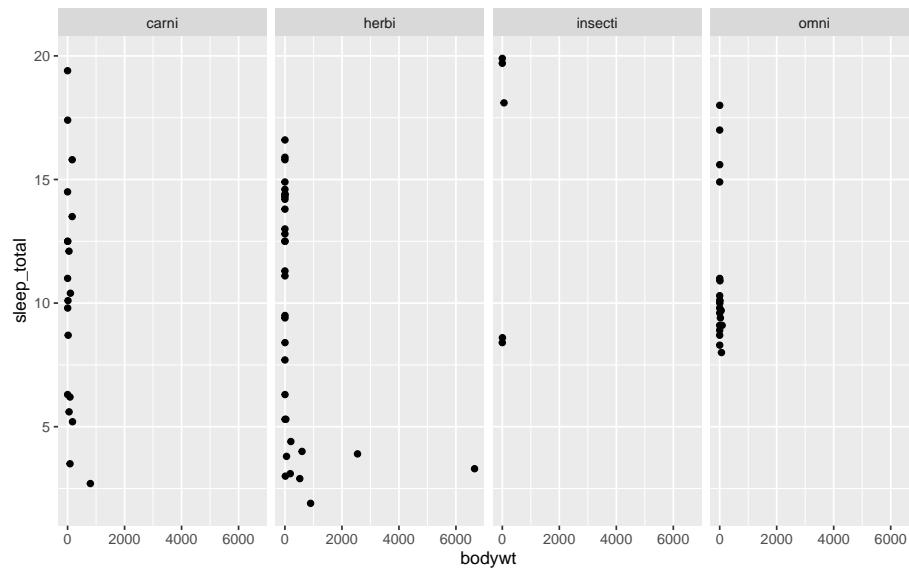
Es decir, en los renglones colocamos una característica, por ejemplo, el tipo de alimentación `vore` de los animales, mientras que en las columnas colocamos la conservación `conservation`. Y graficaremos en cada panel la relación que hay entre `bodywt` y la `sleep_total`.

```
keep_rows <- !(is.na(msleep$vore)) & !(is.na(msleep$conservation))
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total)) +
       geom_point() +
       facet_grid(vore ~ conservation)
```



Para dividir la gráfica en relación a una sola variable eliminamos uno de los primeros argumentos de la función `facet_grid`, dependiendo de cómo queremos la gráfica: en renglones o columnas. Para excluir una de las variables, la remplazamos con un punto(.). Por ejemplo, dejemos solo la característica `vore`, en la sección de las columnas.

```
keep_rows <- !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total)) +
       geom_point() +
       facet_grid(. ~ vore)
```



## 10.15 Ejercicios 12

- De los datos `DNase`, que vienen en las bases de datos de R:
  - Hacer una grafica en múltiples paneles para visualizar la relación que existe entre la densidad (en el eje de las y) vs.la concentración (eje de las x), para las corridas 1 a 4.

La respuesta se encuentran disponibles en: [Solución Ejercicios 12](#)

## 10.16 Modificando los títulos y su estilo

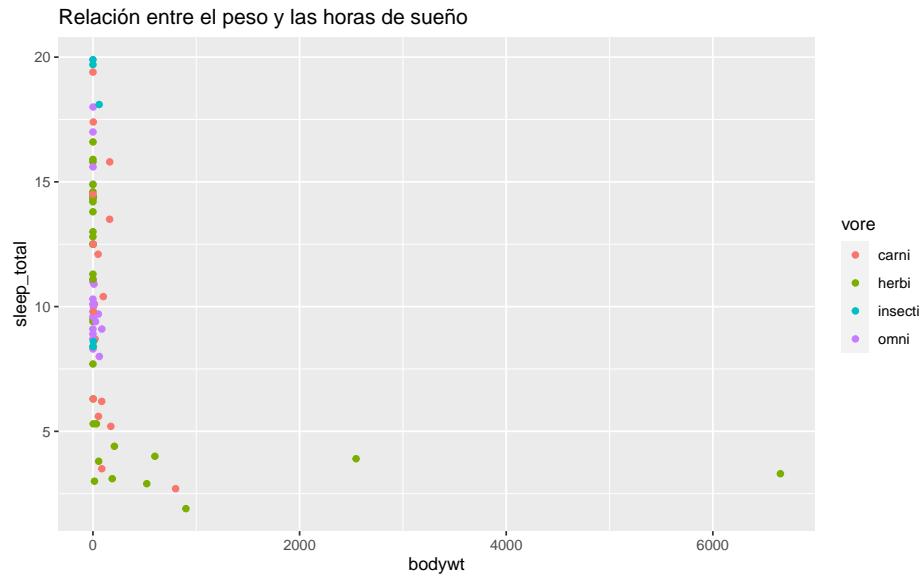
Ahora que hemos dibujado las partes principales del gráfico. Es posible que queramos agregar el título principal de la gráfica y quizás cambiar los títulos de los ejes X e Y. Además de manipular características de las etiquetas, como el tamaño, color, entre otras. Esto se puede lograr utilizando las capas `labs` y `theme`.

### 10.16.1 Agregando títulos

`labs` es la capa que controla las etiquetas de los ejes, el título (`title`) del gráfico y el subtítulo (`subtitle`). Por ejemplo, colocaremos un título a la gráfica.

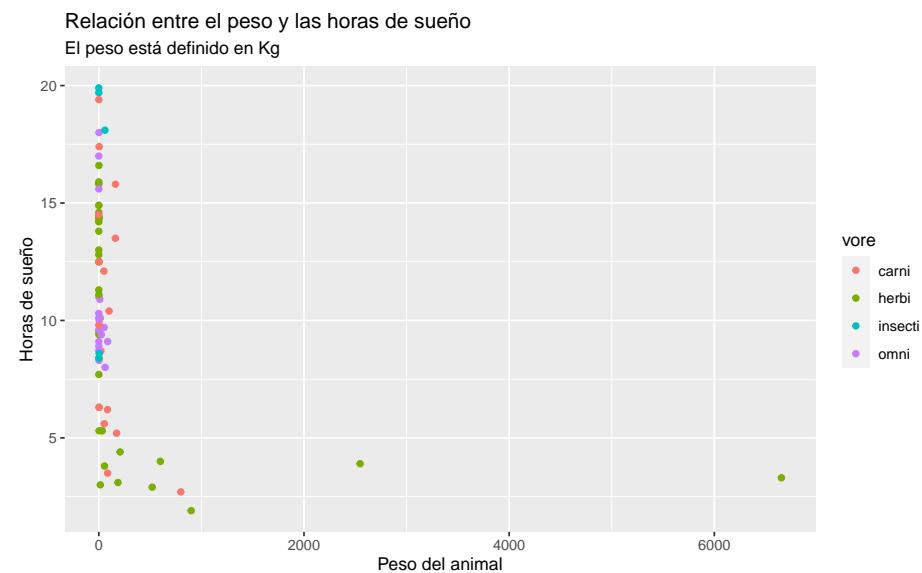
```
keep_rows <- !is.na(msleep$vore)
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore)) +
  geom_point( )
  labs(title="Relación entre el peso y las horas de sueño")
```

## Introducción a R y RStudio



Dentro de la misma instrucción podemos modificar los títulos de los ejes (x , y) o podemos agregar otra capa labs con esta información. Incluso podemos agregar un subtítulo (subtitle). Por ejemplo:

```
keep_rows <- !is.na(msleep$vore)
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore)) +
  geom_point() +
  labs(title="Relación entre el peso y las horas de sueño",
       subtitle="El peso está definido en Kg",
       x="Peso del animal", y="Horas de sueño")
```

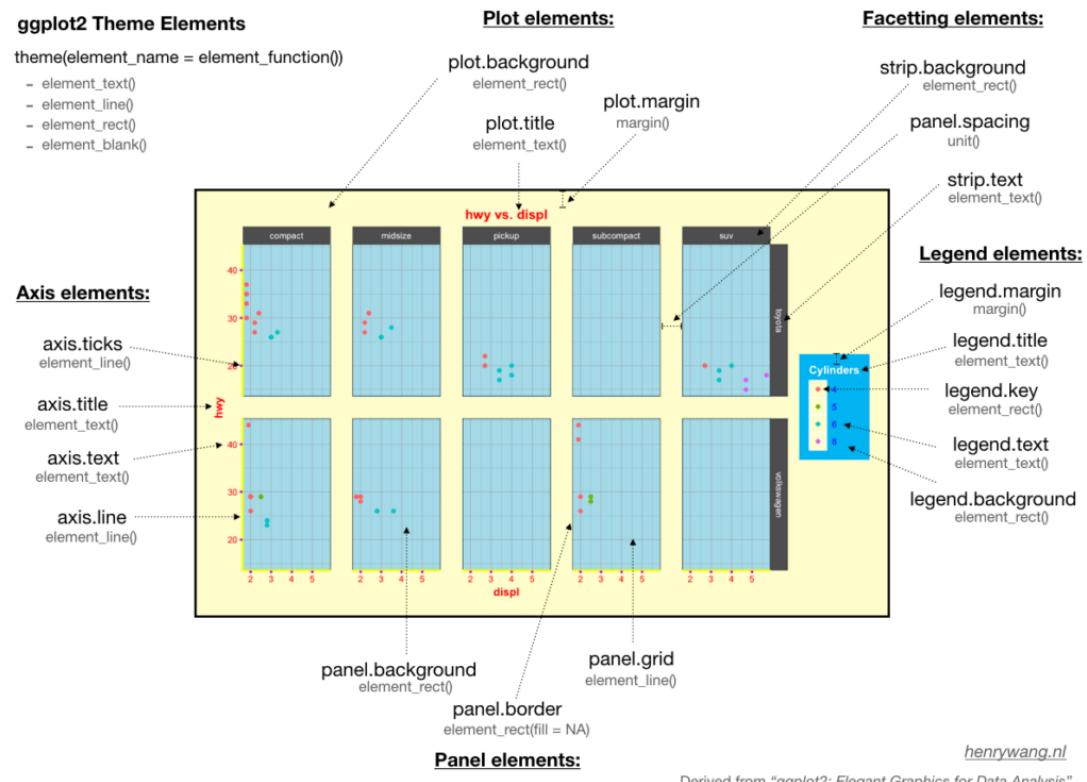


### 10.16.2 Modificando el estilo de los textos

`theme` permite controlar diversos aspectos de la gráfica como son fuentes (tamaño, color, estilo, entre otros), marcas, franjas de paneles, *grid* y fondo. `theme` ayuda a hacer que la trama sea estéticamente agradable o que coincida con una guía de estilo existente.

El `theme` tiene 2 componentes:

1. Los **elementos** del tema
2. La **función** del elemento



Element name	Description	Theme elements	Text geoms	Description
axis.title	Appearance of axis labels on both axes	family	family	<code>Helvetica, Times, Courier</code>
axis.title.x	Appearance of x-axis label	face	fontface	<code>plain, bold, italic, bold.italic</code>
axis.title.y	Appearance of y-axis label	colour	colour	Color (name or "#RRGGBB")
axis.ticks	Appearance of tick labels on both axes	size	size	Font size (in points for theme elements; in mm for geoms)
axis.ticks.x	Appearance of x tick labels	hjust	hjust	Horizontal alignment: 0=left, 0.5=center, 1=right
axis.ticks.y	Appearance of y tick labels	vjust	vjust	Vertical alignment: 0=bottom, 0.5=middle, 1=top
legend.title	Appearance of legend title	angle	angle	Angle in degrees
legend.text	Appearance of legend items	lineheight	lineheight	Line spacing multiplier
plot.title	Appearance of overall plot title			
strip.text	Appearance of facet labels in both directions			
strip.text.x	Appearance of horizontal facet labels			
strip.text.y	Appearance of vertical facet labels			

Por ejemplo, vamos a cambiar el tipo de letra, color y ajuste, del título principal de la gráfica, accediendo al elemento `plot.title`.

## Introducción a R y RStudio

```
keep_rows <- !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore)) +
  geom_point() +
  labs(title="Relación entre el peso y las horas de sueño",
       subtitle="El peso está definido en Kg",
       x="Peso del animal", y="Horas de sueño") +
  theme(plot.title=element_text(face="bold",family="Times",
                                colour="red",size=14,hjust = 0.5))
```



Como podemos observar, no se modificó el estilo del subtítulo. Para modificarlo accedemos al elemento `plot.subtitle`

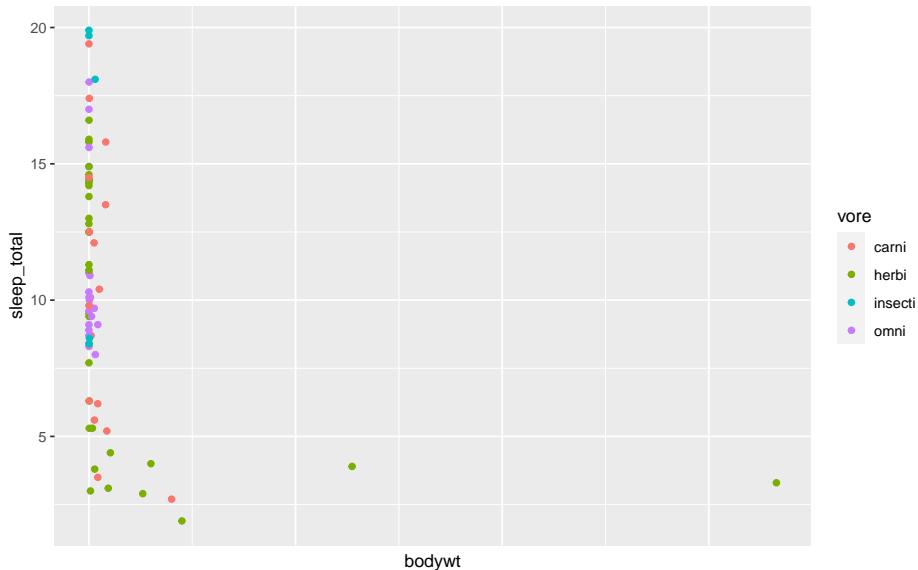
```
keep_rows <- !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore)) +
  geom_point() +
  labs(title="Relación entre el peso y las horas de sueño",
       subtitle="El peso está definido en Kg",
       x="Peso del animal", y="Horas de sueño") +
  theme(plot.title=element_text(face="bold",family="Times",
                                colour="red",size=14,hjust = 0.5),
        plot.subtitle=element_text(face="italic",color="blue",
                                  size=12,hjust = 0.5))
```

## Introducción a R y RStudio



Dentro de las funciones de los elementos, tenemos `element_blank`, que nos permite quitar algún elemento. Por ejemplo, podemos quitar los textos del eje x, accediendo al elemento `axis.text.x`.

```
keep_rows <- !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore)) +
  geom_point() +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank())
```



### 10.17 Consultando las *fonts* disponibles

Finalmente, es importante mencionar que no siempre podemos acceder a las diferentes familias de **fonts**. Para consultar las familias que tenemos disponibles, dependiendo de nuestro sistema operativo, podemos utilizar el paquete **extrafont**.

```
library(extrafont)

font_import()
fonts()
fonttable()[20:30, ]
```

#### 10.17.1 Ejercicios 13

1. Utiliza la gráfica realizada en [Ejercicios 11](#) y agrega un título, subtítulo, títulos a los ejes y dales formato, en términos, del estilo, tamaño, familia y color de la letra, así como de la posición.
2. Dar formato a la etiquetas de las leyendas (color, tipo de letra, etc.)

La respuesta se encuentran disponibles en: [Solución Ejercicios 13](#)

### 10.18 Modificando el fondo de la gráfica

Además de la apariencia de las etiquetas, podemos modificar otras características de las gráficas, por ejemplo, el área de graficación. En este sentido, podemos modificar el área de graficación utilizando algunos “temas” que ggplot tiene por defecto o de manera personalizada.

#### 10.18.1 Modificando el fondo de la gráfica con un fondo por defecto

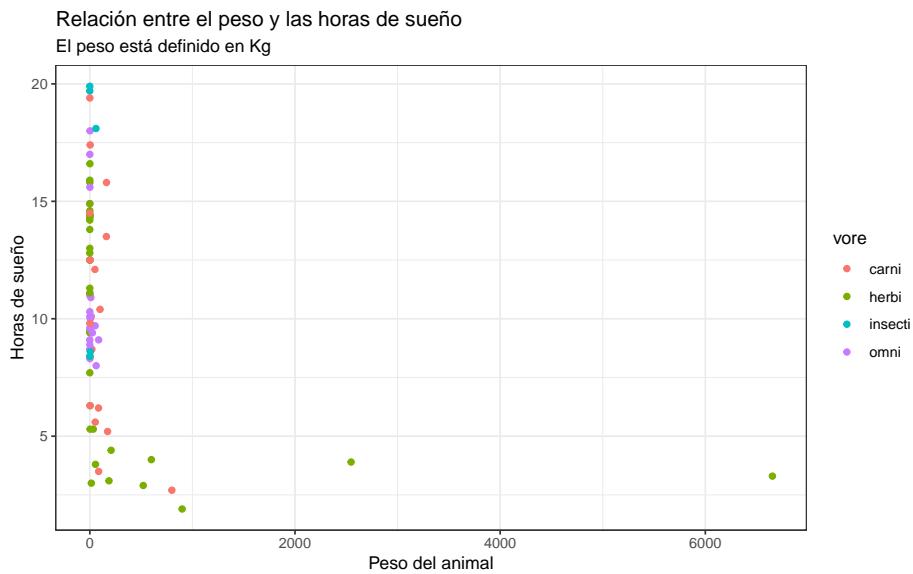
ggplot2 tiene por defecto diversos temas. El “tema” básico de fondo que, ggplot2, pone por defecto es el **theme\_gray**, sin embargo, se puede cambiar la apariencia de las tramas utilizando alguno de los siguientes temas definidos.

```
1. theme_gray()
2. theme_bw()
3. theme_minimal()
4. theme_classic()
5. theme_void()
```

Haremos una gráfica de puntos, con la relación del peso y las horas totales de sueño, y vamos a utilizar el ‘**theme\_bw**’, como una capa dentro de nuestra gráfica, para cambiar el fondo.

```
keep_rows <- !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total,
            color = vore)) +
  geom_point( ) +
  labs(title="Relación entre el peso y las horas de sueño",
       subtitle="El peso está definido en Kg",
```

```
x="Peso del animal", y="Horas de sueño") +  
theme_bw()
```



Por otra parte, podemos modificar las características de la gráfica, de forma personalizada, en términos del fondo y las líneas divisorias (*grid*), entre otras. Para consultar cómo realizar estos cambios, revisar la sección de [Apéndice 4](#)

## 10.19 Ejercicios 14

1. De la gráfica generada en [Ejercicios 13](#), cambiar el tema utilizando algún tema por defecto, para cambiar la apariencia de la gráfica.
2. Generar un tema personalizado, que puedes utilizar en tus gráficas. Puedes modificar, por ejemplo, el color de fondo el color y ancho del borde, entre otras. Aplicar este tema a alguna gráfica

La respuesta se encuentran disponibles en: [Solución Ejercicios 14](#)

## 10.20 Otras geometrías

Existen varias geometrías para graficar, la Figura 7 muestra un esquema de las diversas geometrías del paquete `ggplot2`. En esta lección veremos como hacer un histograma de frecuencias, una gráfica de barras y una gráfica de cajas y bigotes (*boxplot*)

# Introducción a R y RStudio

**Visualización de Datos usando ggplot2**

Guía Rápida

R Studio

**Conceptos Básicos**

ggplot2 se basa en la idea que cualquier gráfica se puede construir usando estos tres componentes: **datos**, **coordenadas** y **objetos geométricos (geoms)**. Este concepto se llama **gramática de las gráficas**.

Para visualizar resultados, asigne variables a las propiedades visuales o **estéticas**, como tamaño, color, posición x o y.

Para construir una gráfica completa este patrón

```
ggplot(data = mpg, aes(x = cty, y = hwy)) +  
  FUNCTION GEOM +  
  mapping aes(ESTÉTICAS),  
  stat = "STAT",  
  position = "POSICIÓN",  
  FUNCTION COORDINADA +  
  FUNCTION FICHA +  
  FUNCTION ESCALAS +  
  FUNCTION TEMA
```

No Requerido, se proveen valores iniciales

Este comando comienza una gráfica, completa la mediante agregando capas, un **geom** por capa.

geom\_point(): **cty**, **y** = **hwy**, **data** = **mpg**, **geom** = "point"

Este comando es una gráfica completa, tiene datos, las estéticas asignadas y por lo menos un **geom**.

last\_plot()

Devuelve la última gráfica

ggsave("plot.png", width = 5, height = 5)

La última gráfica es grabada como una imagen de 5 por 5 pulg., usa el mismo tipo de archivo que la extensión

**Geoms** – Funciones geom se utilizan para visualizar resultados. Asigne variables a las propiedades estéticas del geom. Cada geom forma una capa.

**Geométricas Elementales**

- a <- ggplot(economics, aes(date, unemployed))
- b <- ggplot(seals, aes(x = long, y = lat))
- c <- geom\_blank()
- d <- geom\_curve(aes(yend = lat + 1, xend = long + 1, curvature = 0), x, yend, y, alpha, angle, color, family, fontface, size, linetype, shape, stroke)
- e <- geom\_rect(xmin = 1, xmax = 10, ymin = 1, ymax = 10, alpha, color, group, linetype, size, fill, shape, stroke)
- f <- geom\_polygon(aes(group = group))
- g <- geom\_rect(xmin = 1, xmax = 10, ymin = 1, ymax = 10, alpha, color, group, linetype, size, fill, shape, stroke)
- h <- geom\_ribbon(aes(ymin = unemployed - 900, ymax = unemployed + 900, alpha, color, fill, group, linetype, size, weight))
- i <- geom\_smooth(method = lm)
- j <- geom\_text(aes(label = cyl, nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE))
- k <- geom\_jitter(height = 2, width = 2)
- l <- geom\_point()
- m <- geom\_quantile()
- n <- geom\_rug(sides = "bl")
- o <- geom\_smooth(method = lm)
- p <- geom\_text(aes(label = cyl, nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE))
- q <- geom\_smooth(method = lm)
- r <- geom\_text(aes(label = cyl, nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE))
- s <- geom\_smooth(method = lm)
- t <- geom\_text(aes(label = cyl, nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE))
- u <- geom\_smooth(method = lm)
- v <- geom\_text(aes(label = cyl, nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE))
- w <- geom\_smooth(method = lm)
- x <- geom\_text(aes(label = cyl, nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE))
- y <- geom\_smooth(method = lm)
- z <- geom\_text(aes(label = cyl, nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE))

**Dos Variables**

**X Continua, Y Continua**

- e + geom\_label(aes(label = cyl), nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE)
- x, y, alpha, angle, color, family, fontface, size, linetype, shape, stroke

**Distribución Bivariada Continua**

- h <- ggplot(diamonds, aes(carat, price))
- x, y, alpha, color, fill, linetype, size, weight

**Función Continuo**

- i <- ggplot(economics, aes(date, unemployed))
- x, y, alpha, color, group, linetype, size

**Visualizando el Error**

- df <- data.frame(grp = c("A", "B"), fit = 4:12)
- j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
- f + geom\_crossbar(fatten = 2)
- x, y, ymin, ymax, alpha, color, fill, group,
- g + geom\_errorbar()
- x, ymin, ymax, alpha, color, group, linetype, size, width
- h + geom\_errorbarh()
- i + geom\_linerange()
- x, y, ymin, ymax, alpha, color, fill, group, linetype, size
- j + geom\_pointrange()
- x, y, ymin, ymax, alpha, color, fill, group, linetype, size

**Mapas**

- data <- data.frame(murders = USArrests\$Murder, state = tolower(substr(USArrests\$State, 1, 2)))
- k <- ggplot(data, aes(state))
- l <- geom\_map(aes(fill = state), map = map) + expand\_limits(x = maplong, y = maplat)
- map\_id, alpha, color, fill, linetype, size

**Trés Variables**

- sealsSz <- with(seals, sqrt(delta\_long \* delta\_lat \* 2))
- l <- ggplot(seals, aes(long, lat))
- l + geom\_contour(aes(z = z))
- x, y, z, alpha, colour, group, linetype, size, weight

**Argumentos**

- l + geom\_raster(aes(z = z), blurborder = 5, blurborder\_alpha = 0.5, blurborder\_type = "solid", blurborder\_color = "#0000FF", blurborder\_size = 1)
- x, y, alpha, color, fill, linetype, size, weight
- label = etiqueta, angle = ángulo, size = tamaño, weight = peso, alpha = opacidad, type = tipo, fontface = tipo de letra, fontcolor = color, fontweight = fuente, ajusste = horizontal, lineheight = grosor de linea

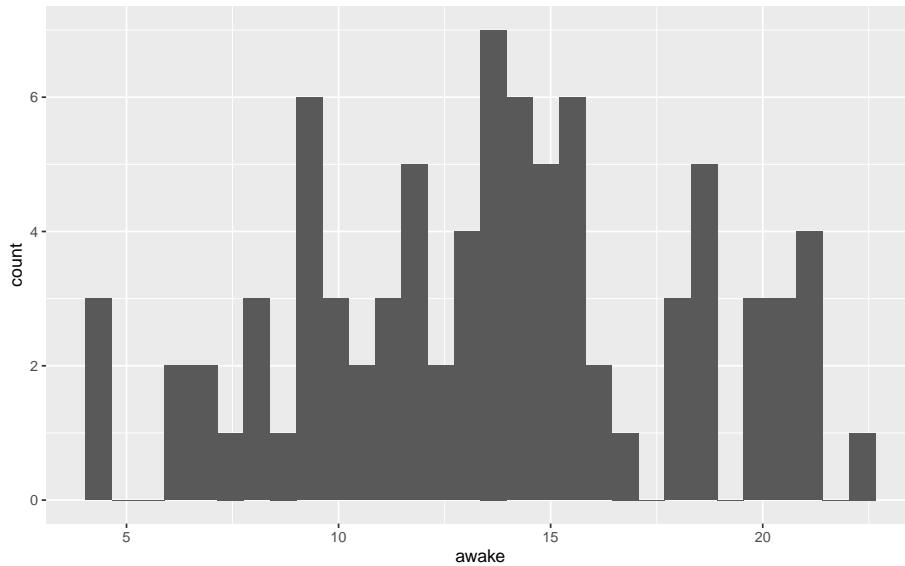
## 10.20.1 Histogramas

Los histogramas de frecuencia sirven para obtener una “**primera vista**” general, o panorama, de la distribución de los datos, respecto a una característica en particular. Para graficar datos en un histograma, utilizamos la geometría **geom\_histogram**. Por ejemplo, grafiquemos en un histograma, cómo se distribuye la variable **awake** de los animales.

Para graficar datos en un histograma, utilizamos la geometría **geom\_histogram**. Por ejemplo, grafiquemos en un histograma, cómo se distribuye la variable **awake** de los animales.

```
ggplot(data = msleep, aes(awake)) +  
  geom_histogram()  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

## Introducción a R y RStudio



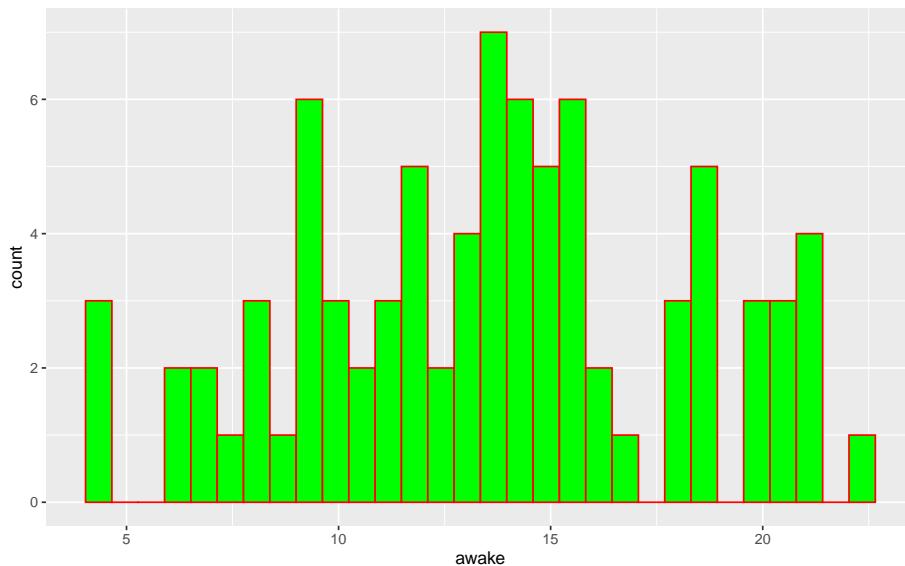
Podemos modificar el número de barras por medio de:

- **binwidth**: Con esta opción le damos el ancho de la barra
- **bins**: Con esta opción le decimos cuántas barras queremos
- **breaks**: Con esta opción le damos un rango de valores

Sino se especifica alguno de ellos, entonces se utiliza **bins** por default, con un valor de 30.

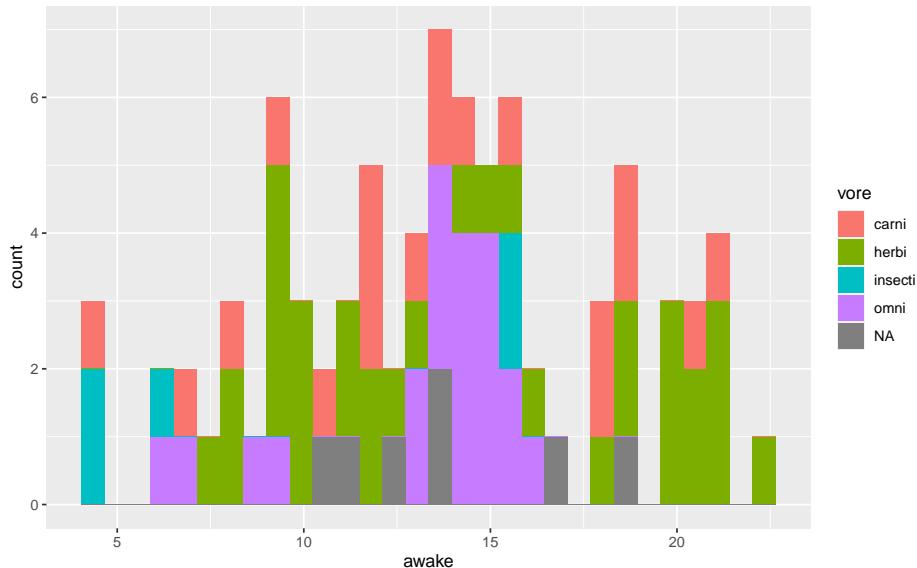
Podemos cambiar la estética de la gráfica. Por ejemplo el color de las barras, utilizando los parámetros **fill** y **color**.

```
ggplot(data = msleep, aes(awake)) +  
  geom_histogram(fill="green",color="red" )  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



También se podría asociar el color a la característica **vore**

```
ggplot(data = msleep, aes(awake, fill=vore)) +  
  geom_histogram()  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



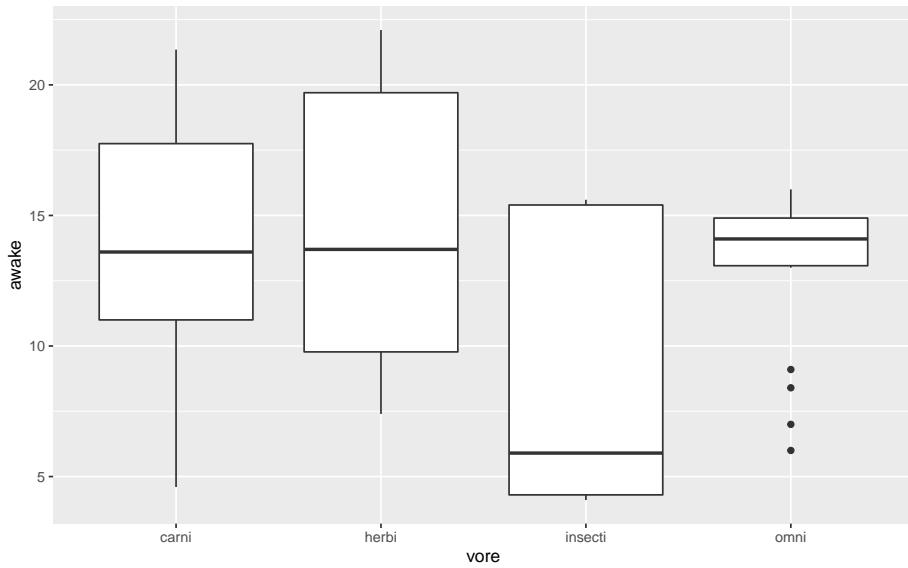
### 10.21 Gráficas de cajas y bigotes (boxplots)

La gráfica de cajas y bigotes (*boxplot*) permite visualizar de manera clara la distribución de los datos y sus características en términos de cuartiles. Como herramienta visual se puede utilizar para ilustrar los datos, para estudiar simetría, para estudiar las colas.

Para hacer una gráfica de cajas y bigotes utilizamos la geometría `geom_boxplot`. Por ejemplo, podemos graficar las horas que los animales permanecen despiertos (`awake`) vs su tipo de alimentación (`vore`).

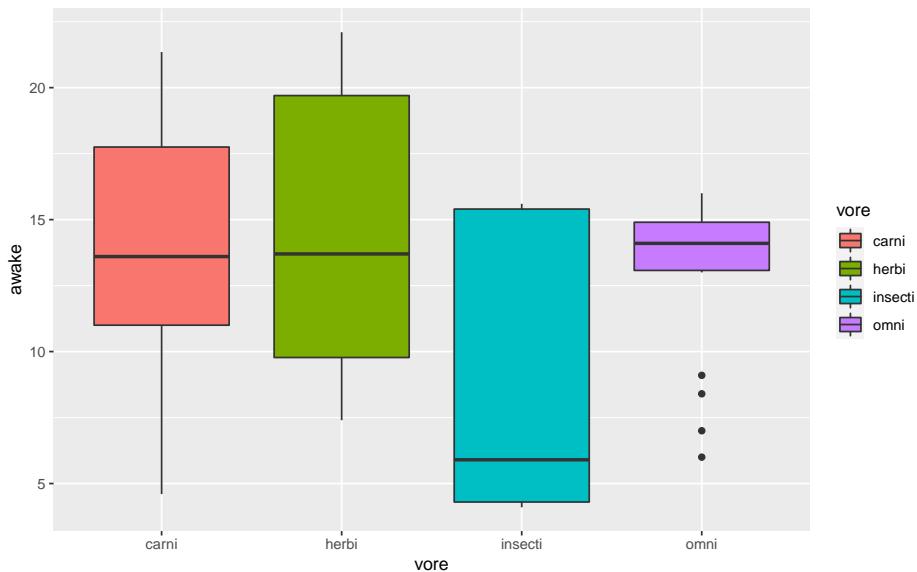
```
keep_rows <- !(is.na(msleep$vore))  
ggplot(data = msleep[keep_rows, ], aes(x=vore, y=awake)) +  
  geom_boxplot()
```

## Introducción a R y RStudio



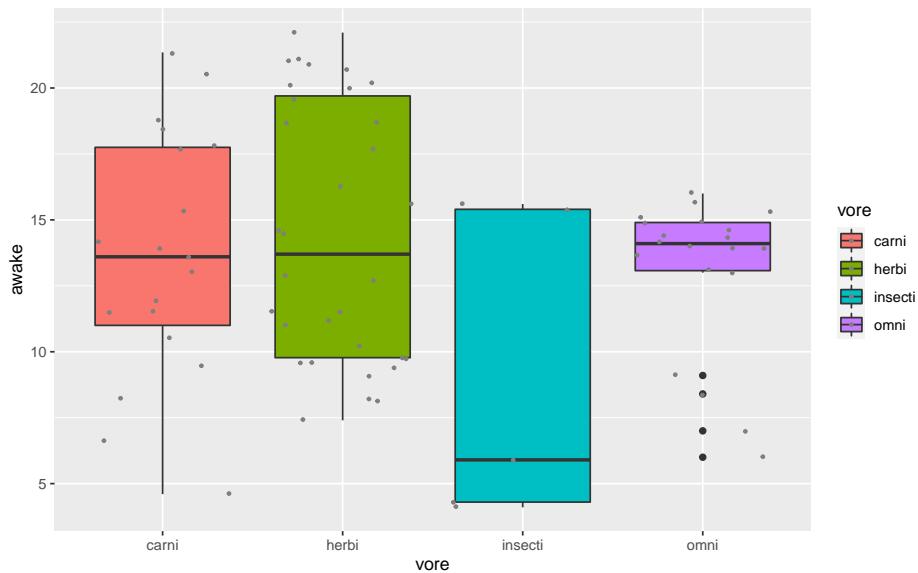
Podemos agregar `fill` a las cajas, asociada al la característica `vore`

```
keep_rows <- !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ], aes(x=vore, y=awake,
                                         fill=vore)) +
  geom_boxplot()
```



Ademas, podemos agregar una capa más con la distribución de los puntos, con `geom_jitter`.

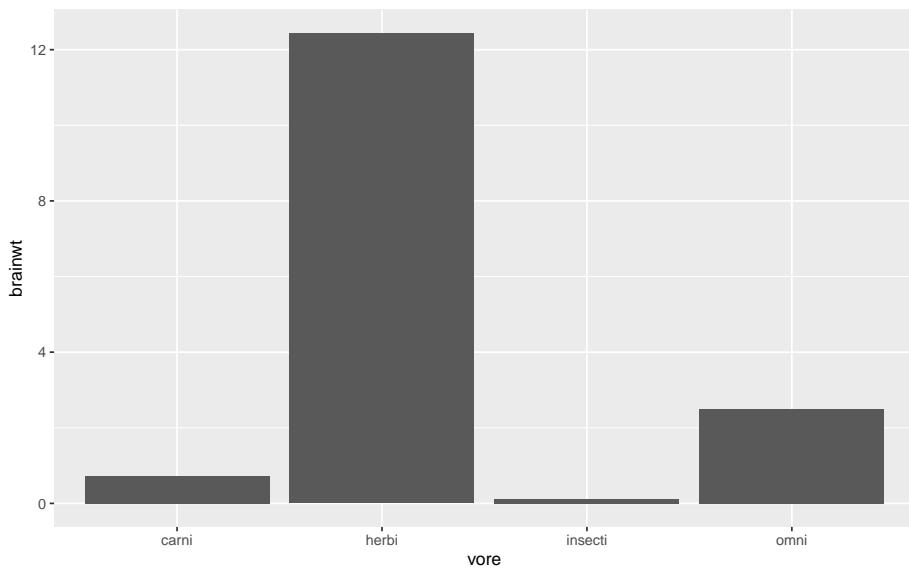
```
keep_rows <- !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ], aes(x=vore, y=awake,
                                         fill=vore)) +
  geom_boxplot() +
  geom_jitter(size=0.7, color="grey50")
```



## 10.22 Gráficas de barras

Con las gráficas de barras también podemos visualizar la distribución de los datos, por ejemplo, veamos cómo se distribuyen los tamaños de las aletas de los pingüinos en las diversas islas. Graficaremos en el eje de las **X** las característica **vore** y en el eje de las **Y** la característica **brainwt**

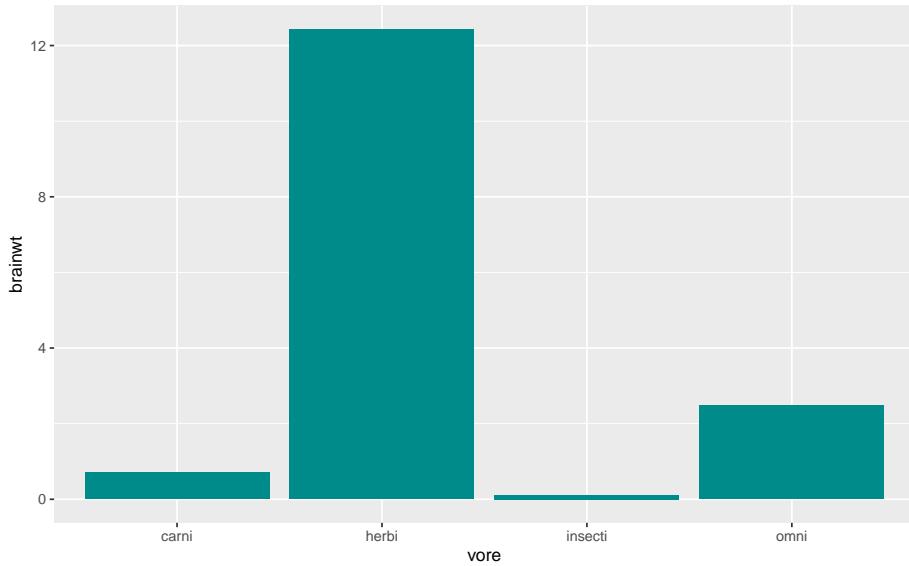
```
keep_rows <- !(is.na(msleep$brainwt)) & !(is.na(msleep$vore))
ggplot(data = msleep[keep_rows, ], aes(x=vore, y=brainwt)) +
  geom_bar(stat="identity")
```



Podemos agregar un **color constante** al **fill** de las barras, por ejemplo, le pondremos el color “**cyan4**”.

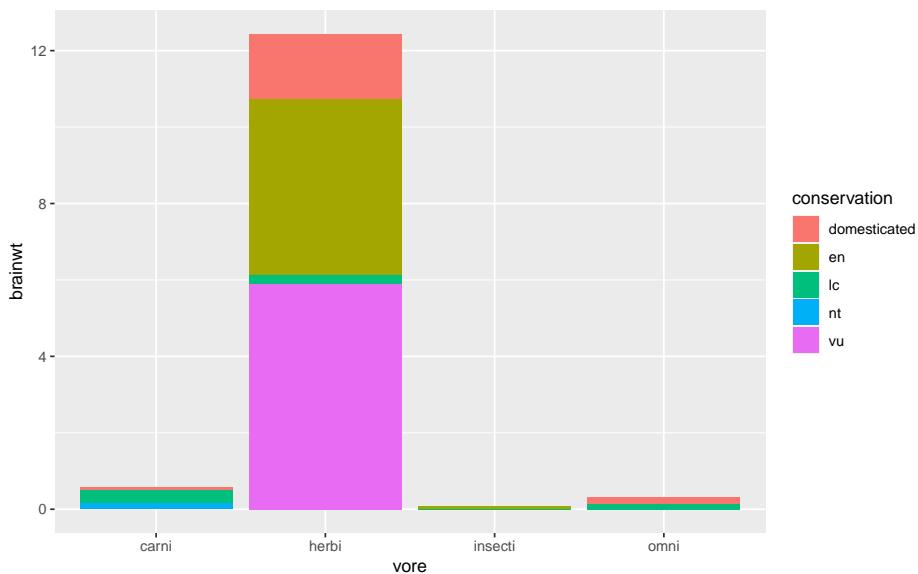
## Introducción a R y RStudio

```
keep_rows <- !(is.na(msleep$brainwt)) & !(is.na(msleep$vore))  
ggplot(data = msleep[keep_rows, ],aes(x=vore,y=brainwt)) +  
  geom_bar(stat="identity", fill="cyan4")
```



Por otra parte podemos **asociar el color a alguna característica**, por ejemplo, a la característica **vore**, para el **fill** de las barras.

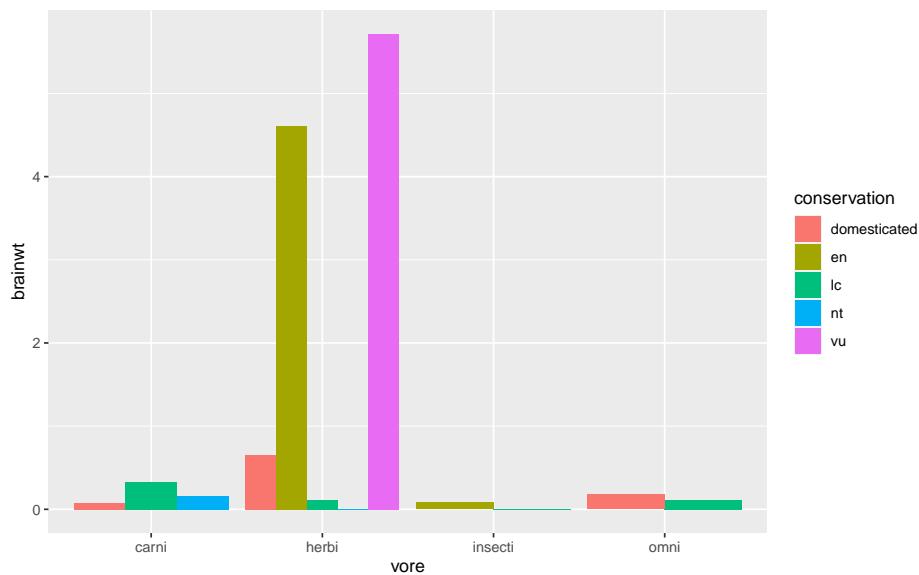
```
keep_rows <- !(is.na(msleep$brainwt)) & !(is.na(msleep$vore)) &  
  !(is.na(msleep$conservation))  
ggplot(data = msleep[keep_rows, ],aes(x=vore,y=brainwt,  
  fill=conservation)) +  
  geom_bar(stat="identity")
```



El parámetro **dodge** permite que **las barras queden una junto a otra**.

## Introducción a R y RStudio

```
keep_rows <- !(is.na(msleep$brainwt)) & !(is.na(msleep$vore)) &
  !(is.na(msleep$conservation))
ggplot(data = msleep[keep_rows, ], aes(x=vore, y=brainwt,
                                         fill=conservation)) +
  geom_bar(stat="identity", position="dodge")
```



## 10.23 Ejercicios 15

1. Con los datos de DNase, realiza un histograma de frecuencias, para visualizar la distribución de los valores de la característica density. Dale formato a las barras y a la gráfica en general, en términos del color de las barras, el tema de la gráfica y las leyendas de la misma.
2. Con los datos de msleep realiza una grafica de barras o un boxplot, para visualizar las horas que los animales permanecen despiertos (awake, en eje de las y) vs el tipo de alimentación(vore, eje de las x). Dale formato a la gráfica y en el caso del boxplot agrega la distribución de los puntos con la capa geom\_jitter.

La respuesta se encuentran disponibles en: [Solución Ejercicios 15](#)

## 10.24 Guardando la imagen en un archivo

Para Guardar la imagen podemos utilizar la función `ggsave()` que pertenece al paquete `gridExtra` o podemos utilizar las funciones basicas de R en conjunto con la función `dev.off()`.

### 10.24.1 Guardando la imagen en un archivo utilizando `ggsave()`

Por default `ggsave`, guarda la última gráfica generada, a menos que le indiquemos lo contrario.  
Por ejemplo, si sólo escribo:

## Introducción a R y RStudio

```
ggsave("results/miGrafica.pdf")
## Saving 8 x 5 in image
```

guardará la imagen de la última gráfica de barras que realizamos y por la extensión proporcionada del archivo, la guardará en formato pdf.

Podemos también especificar qué gráfica queremos guardar, siempre y cuando la gráfica se haya almacenado en una variable. Por ejemplo,

```
keep_rows <- !(is.na(msleep$vore))
gf1<-ggplot(data = msleep[keep_rows, ], aes(x=vore, y=awake,
                                              fill=vore)) +
  geom_boxplot() +
  geom_jitter(size=0.7, color="grey50") +
  theme_bw()

gf2<-ggplot(data = msleep, aes(awake)) +
  geom_histogram(fill="green",color="red" ) +
  theme_classic()

ggsave("results/miBoxplot.pdf",plot=gf1)
## Saving 8 x 5 in image
```

**10.24.1.1 Cambiando el tamaño de la imagen y los dpi's** También, podemos agregar el tamaño de la imagen y el número de **dpi's**, por ejemplo:

```
g1<-ggplot(msleep, aes(x = bodywt, y = sleep_total)) +
  geom_point() +
  theme_bw()
ggsave("results/migrafica3.jpg", plot=g1, width=5,height=3.5,dpi=300)
```

### 10.24.2 Guardando la imagen con instrucciones de R base y dev.off()

Por otra parte podemos **guardar** nuestras **gráficas** utilizando las **instrucciones básicas de R**, como las que se muestran a continuación, **acompañadas de dev.off()**.

- pdf()
- png()
- tiff()
- jpeg()

Por ejemplo:

```
pdf("results/migrafica4.pdf")
ggplot(msleep, aes(x = bodywt, y = sleep_total)) +
  geom_point() +
  theme_bw()

dev.off()
## pdf
## 2
```

Nota: Es importante, colocar siempre el \*\*\*`dev.off()`\*\*\*, para hacer el cierre del archivo que acabamos de crear.

### 10.25 Ejercicios 16

1. Lea el archivo BacterialGrow.csv (sus campos estan separados por comas, no por tabuladores)
  - a. Grafica el crecimiento de las diferentes bacterias vs el tiempo, en una gráfica de líneas. Ponga color a las líneas en términos de la clase de bacteria.
  - b. Poner títulos a los ejes y a la gráfica. Además, de cambiar el estilo del texto de las leyendas, en términos del tipo de letra y color de la misma.
  - c. Cambia el “tema”, por alguno de los definidos, que no sea theme\_gray() o genera tu propio “tema”.
  - d. Guarda la imagen en un archivo.
2. Guarda las imágenes generadas en los ejercicios anteriores, trata de utilizar los parámetros width, height, res y dpi.

La respuesta se encuentran disponibles en: [Solución Ejercicios 8](#)

### 10.26 Ligas de información de ggplot2

Existen mas de 30 geometrías y sus extensiones. Puedes revisarlas con calma en:

<https://www.ggplot2-exts.org>

Para terminar de entender ggplot2, puedes revisar con calma la hoja de ayuda: ggplot2-cheatsheet.pdf (Existen otras hojas en el sitio: <http://rstudio.com/cheatsheets>)

## 11 Respuestas de Ejercicios

### 11.1 Solución Ejercicios 1

1. Busca qué hace la función `rm`

```
?rm
```

```
## remove                  package:base          R Documentation
##
## Remove Objects from a Specified Environment
##
## Description:
##
##   'remove' and 'rm' can be used to remove objects. These can be
##   specified successively as character strings, or in the character
##   vector 'list', or through a combination of both. All objects thus
##   specified will be removed.
```

## Introducción a R y RStudio

La función `rm` sirve para borrar un objeto.

2. ¿Cuáles son los argumentos de la función `rep`?

```
args(rep)
## function (x, ...)
## NULL
```

3. Proporciona tres paquetes que contengan el término `prediction`

```
stats::predict
modelr::add_predictions
locfit::predict.locfit
```

Regresar a [Ejercicios 1](#)

## 11.2 Solución Ejercicios 2

1. Busca los archivos `.RData` y `.Rhistory`.

Abrir una terminal, moverse al directorio de trabajo, y con el comando `ls -a` listamos los archivos ocultos en Unix.

2. ¿Cómo veo el contenido del archivo `.Rhistory`?

Una manera es utilizar el comando `more` de Unix.

3. ¿En qué directorio fueron creados los archivos `.RData` y `.Rhistory`?

Los archivos ocultos `.RData` y `.Rhistory` se crean en el directorio de trabajo (el último definido, por el usuario)

4. ¿Por qué se crearon ahí?

Los archivos ocultos `.RData` y `.Rhistory` se crean siempre en el directorio de trabajo (actual).

Regresar a [Ejercicios 2](#)

## 11.3 Solución Ejercicios 3

1. ¿Cuáles de las siguientes definiciones son correctas?

```
variable.Uno <- 10      # Correcto
25 <- "curso R"       # Incorrecto
## Error in 25 <- "curso R": invalid (do_set) left-hand side to assignment
Sqrt25 <- 5            # Correcto
nombre-25 <- 50         # Incorrecto
## Error in nombre - 25 <- 50: could not find function "-<-
resultado <- 10/0        # Correcto
cer0<-3/-Inf            # Correcto
Inf<-48                 # Incorrecto
## Error in Inf <- 48: invalid (do_set) left-hand side to assignment
```

Regresar a [Ejercicios 3](#)

### 11.4 Solución Ejercicios 4

- Predice el resultado, luego corrobóralo:

```
8 + 2
## [1] 10
13 / 4
## [1] 3.25
6 + 5 * 10 - 3
## [1] 53
6 + 5 * (10 - 3)
## [1] 41
9 (8 - 4)
## Error in eval(expr, envir, enclos): attempt to apply non-function
A + b
## Error in eval(expr, envir, enclos): object 'b' not found
A <- 5; a <- 11; A + a
## [1] 16
c <- "prueba"
a + c
## Error in a + c: non-numeric argument to binary operator
```

- Calcula algunos datos sobre una circunferencia a partir de su radio.

- Guarda en la variable radio el valor del radio de la circunferencia con la que vamos a trabajar.

```
radio <- 20
```

- Calcula el perímetro de la circunferencia.  $2\pi \times \text{radio}$

```
perimetro <- 2 * pi * radio
```

- Calcula el área del círculo delimitado por dicha  $\pi \times r^2$  circunferencia.

```
area <- pi * radio ^ 2
```

- Escribe las siguientes expresiones aritméticas en R:

$$\frac{a}{b} + 1$$

```
a/b+1
```

$$\frac{a+b}{c+d}$$

```
(a+b)/(c+d)
```

$$(a+b)\frac{c}{d}$$

```
(a+b)*(c/d)
```

## Introducción a R y RStudio

$$\frac{c + \frac{b}{a}}{d + \frac{a}{c}}$$

```
(c+b/a)/(d+a/c)
```

Regresar a [Ejercicios 4](#)

## 11.5 Solución Ejercicios 5

1. Escribir la expresión necesaria para evaluar si el valor almacenado en la variable x, coincide con alguno de una serie de valores, por ejemplo 2, 7 y 9.

```
(x == 2) | (x == 7) | (x == 9)  
## [1] NA
```

2. Escribir la expresión necesaria para evaluar si el valor almacenado en la variable no coincide con alguno de una serie de valores, por ejemplo 2, 7 y 9

```
!((x == 2) | (x == 7) | (x == 9))  
## [1] NA
```

3. Escribe una expresión lógica que compruebe si una variable contiene un valor par. Nota: el operador %% devuelve el resto de la división entera.

```
(y %% 2) == 0  
## [1] NA
```

4. Intenta predecir el resultado de evaluar las siguientes expresiones lógicas. Piensa en el orden en que se evalúan las subexpresiones dentro de cada expresión. Los valores de las variables son:

```
a <- TRUE; b <- TRUE; c <- FALSE; d <- FALSE  
c || !a && b  
## [1] FALSE  
!(a || c) || b && !c  
## [1] TRUE  
!( !( ! (a && c || d)))  
## [1] TRUE  
!(5<3) && a || !(d || c)  
## [1] TRUE
```

Regresar a [Ejercicios 5](#)

## 11.6 Solución Ejercicios 6

1. Para cada una de las siguientes asignaciones, indica, de qué tipo es el vector resultante:

```
N<-c(5,"cinco")          ## Caracteres  
Solucion<- c(TRUE,FALSE,TRUE) ## Lógico  
Res<-(uno,1)              ## Error
```

## Introducción a R y RStudio

2. Genere un vector con las mediciones del tamaño (superficie), de por lo menos 5 planetas diferentes.

```
planetas<-c(74.8, 510.1, 42.7, 144.8, 7618)
```

- a. Agregue los nombres a cada planeta usando la función names.

```
names(planetas)<-c("Mercurio", "Tierra", "Saturno", "Marte", "Neptuno")
```

- b. Calcule del tamaño el: promedio, mínimo y máximo.

```
mean(planetas)
## [1] 1678.08
min(planetas)
## [1] 42.7
max(planetas)
## [1] 7618
```

- c. Agregue dos planetas más, junto con sus datos de superficie.

```
planetas<-c(planetas, 61.42, 460.2)
names(planetas)[6]<-"Jupiter"
names(planetas)[7]<-"Venus"
```

3. Escribe una expresión que simule el lanzamiento de una moneda 10 veces. Obtén como resultado de la expresión un vector de valores “cara” y “cruz”. Cuenta las ocurrencias de cara y cruz, y la proporción de cada valor. Tip, revise la función sample.

```
experimento <- sample(c("cruz", "cara"), size = 10, replace = TRUE)
experimento
## [1] "cruz" "cara" "cara" "cruz" "cara" "cruz" "cruz" "cruz" "cruz" "cruz"
cara<-experimento[experimento == "cara"]
num_cara<-length(cara)
num_cara
## [1] 3
num_cruz<-10 - num_cara
num_cruz
## [1] 7
```

4. Crea un vector aleatorio de 10 elementos y selecciona aquellos elementos del vector que ocupan índices impares. Usa seq para expresar el vector de índices impares y la función rnorm para generar los números aleatorios.

```
vectNum<-rnorm(10)
vectNum
## [1] 1.1657473 -0.1063841 1.7446083 1.0146729 0.6099442 -0.5856691
## [7] 1.5574157 -1.1697411 0.2199812 0.2064737
vectNum[seq(1,10, by=2)]
## [1] 1.1657473 1.7446083 0.6099442 1.5574157 0.2199812
```

Regresar a [Ejercicios 6](#)

## 11.7 Solución Ejercicios 7

- Con los siguientes vectores, genere un **dataframe**, llamado planetas.

```
name = c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")
type = c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
        "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant")
diameter = c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
rotation = c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
rings = c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE)
```

```
planetas <- data.frame(
  name=c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"),
  type=c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
         "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant"),
  diameter=c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883),
  rotation=c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67),
  rings=c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE))

planetas
##      name           type   diameter rotation rings
## 1 Mercury Terrestrial planet     0.382    58.64 FALSE
## 2 Venus  Terrestrial planet     0.949   -243.02 FALSE
## 3 Earth   Terrestrial planet     1.000     1.00 FALSE
## 4 Mars   Terrestrial planet     0.532     1.03 FALSE
## 5 Jupiter Gas giant            11.209     0.41  TRUE
## 6 Saturn  Gas giant            9.449     0.43  TRUE
## 7 Uranus  Gas giant            4.007    -0.72  TRUE
## 8 Neptune Gas giant            3.883     0.67  TRUE
```

- Obenga el promedio de los valores de las columnas **diameter** y **rotation**

```
mean(planetas$diameter)
## [1] 3.926375
mean(planetas[,4])
## [1] -22.695
```

- Aplique la función **summary** al objeto planetas y vea que sucede.

```
summary(planetas)
##      name           type   diameter   rotation
##  Length:8       Length:8      Min.   : 0.3820  Min.   :-243.0200
##  Class :character Class :character  1st Qu.: 0.8448  1st Qu.:  0.1275
##  Mode  :character Mode  :character  Median : 2.4415  Median :  0.5500
##                                         Mean   : 3.9264  Mean   : -22.6950
##                                         3rd Qu.: 5.3675  3rd Qu.:  1.0075
##                                         Max.   :11.2090  Max.   :  58.6400
##      rings
##  Mode :logical
##  FALSE:4
##  TRUE :4
##
```

## Introducción a R y RStudio

```
##  
##
```

**summary** sólo obtiene los valores estadísticos de las columnas numéricas.

obtenga: a. Todos los planetas con un diámetro mayor a 1.

```
# Obteniendo primero los renglones que cumplen la condición  
keepPlanetas <- planetas$diameter > 1  
# Consultando la información de los renglones que cumplieron la condición  
planetas[keepPlanetas, ]  
##      name      type diameter rotation rings  
## 5 Jupiter Gas giant    11.209     0.41  TRUE  
## 6 Saturn  Gas giant     9.449     0.43  TRUE  
## 7 Uranus Gas giant     4.007    -0.72  TRUE  
## 8 Neptune Gas giant    3.883     0.67  TRUE
```

b. Los planetas que tengan anillos (rings) y que el valor de rotación (rotation) sea menor o igual 0.5

```
# Obteniendo primero los renglones que cumplen la condición  
keepPlanetas <- (planetas$rings == TRUE) & (planetas$rotation <= 0.5)  
# Consultando la información de los renglones que cumplieron la condición  
planetas[keepPlanetas, ]  
##      name      type diameter rotation rings  
## 5 Jupiter Gas giant    11.209     0.41  TRUE  
## 6 Saturn  Gas giant     9.449     0.43  TRUE  
## 7 Uranus Gas giant     4.007    -0.72  TRUE
```

Regresar a [Ejercicios 7](#)

## 11.8 Solución Ejercicios 8

1. Utilizando la función **write.table**, almacene en un archivo el objeto **planetas**, creado en [Ejercicios 7](#).

```
write.table(planetas,"planetasInfo.txt",sep = "\t",quote = F)
```

Regresar a [Ejercicios 8](#)

## 11.9 Solución Ejercicios 9

1. De los datos DNase, que vienen en las bases de datos de R:
  - Explore los datos del dataframe, ¿cuántas columnas y renglones tiene?, ¿qué describe cada dato?
  - Grafique la densidad (en el eje de las y) vs.la concentración.

```
dim(DNase)  
help(DNase)
```

## Introducción a R y RStudio

```
## Run
##   an ordered factor with levels 10 < ... < 3 indicating the assay run.
##
## conc
##   a numeric vector giving the known concentration of the protein.
##
## density
##   a numeric vector giving the measured optical density (dimensionless) in the assay.
##   Duplicate optical density measurements were obtained.
```

```
DNase %>% ggplot(aes(x=conc,y=density)) + geom_point()
```

Regresar a [Ejercicios 9](#)

## 11.10 Solución Ejercicios 10

1. De la gráfica que realizamos en [Ejercicios 9](#), cambiar el color de los puntos con base en la columna **Run**. Además, colocar un símbolo distinto al punto y cambiar la transparencia a 0.4.

```
DNase %>% ggplot(aes(x=conc,y=density,color=Run)) + geom_point(shape=18,alpha=0.4)
```

Regresar a [Ejercicios 10](#)

## 11.11 Solución Ejercicios 11

1. De los datos de **msleep**, graficar la relación que existe entre las características **bodywt** y **awake**, colorear los puntos con base en la característica **vore** y agregar la línea de tendencia (**geom\_smooth**) general, es decir para todos los puntos.

```
msleep %>% filter(!is.na(vore)) %>%
  ggplot(aes(x = bodywt, y = awake)) +
  geom_point(aes(color=vore)) +
  geom_smooth()
```

Regresar a [Ejercicios 11](#)

## 11.12 Solución Ejercicios 12

1. De los datos **DNase**, que vienen en las bases de datos de R:
  - Hacer una grafica en múltiples paneles para visualizar la relación que existe entre la densidad (en el eje de las y) vs.la concentración (eje de las x), para las corridas 1 a 4.

```
DNase %>% filter(Run %in% 1:4) %>%
  ggplot(aes(x=conc,y=density)) + geom_point() +
  facet_grid(Run~.)
```

[Regresar a Ejercicios 12](#)

### 11.13 Solución Ejercicios 13

1. Utiliza la gráfica realizada en [Ejercicios 11](#) y agrega un título, subtítulo, títulos a los ejes y dales formato, en términos, del estilo, tamaño, familia y color de la letra, así como de la posición.

```
g1<-msleep %>% filter(!is.na(vore)) %>%
  ggplot(aes(x = bodywt, y = sleep_total)) +
  geom_point() +
  facet_grid(vore~conservation) +
  labs(x="Peso de los animales",
       y="Tiempo de sueño",
       title= "Relación peso vs tiempo de sueño",
       subtitle = "(en horas)")
g1
```

2. Dar formato a la etiquetas de las leyendas (color, tipo de letra, etc.)

```
g1 + theme(plot.title=element_text(size=15,
                                    family="Helvetica", face="bold.italic", colour="green"))
```

[Regresar a Ejercicios 13](#)

### 11.14 Solución Ejercicios 14

1. De la gráfica generada en [Ejercicios 13](#), cambiar el tema utilizando algún tema por defecto, para cambiar la apariencia de la gráfica.

```
g1<-msleep %>% filter(!is.na(vore)) %>%
  ggplot(aes(x = bodywt, y = sleep_total)) +
  geom_point() +
  facet_grid(vore~conservation) +
  labs(x="Peso de los animales",
       y="Tiempo de sueño",
       title= "Relación peso vs tiempo de sueño",
       subtitle = "(en horas)")
g1 + theme_bw() +
  theme(plot.title=element_text(size=15,
                                family="Helvetica", face="bold.italic", colour="green"))
```

2. Generar un tema personalizado, que puedas utilizar en tus gráficas. Puedes modificar, por ejemplo, el color de fondo el color y ancho del borde, entre otras. Aplicar este tema a alguna gráfica

```
elTema<-
  theme(panel.background=
        element_rect(fill="turquoise")) +
  theme(panel.border=
```

## Introducción a R y RStudio

```
element_rect(color="snow4",
              fill=NA,size=1.5))

g1<-msleep %>% filter(!is.na(vore)) %>%
  ggplot(aes(x = bodywt, y = sleep_total)) +
  geom_point() +
  elTema
g1
```

[Regresar a Ejercicios 14](#)

## 11.15 Solución Ejercicios 15

1. Con los datos de DNase, realiza un histograma de frecuencias, para visualizar la distribución de los valores de la característica density. Dale formato a las barras y a la gráfica en general, en términos del color de las barras, el tema de la gráfica y las leyendas de la misma.

```
DNase %>%
  ggplot(aes(x=density)) +
  geom_histogram(fill="blue",color="white") +
  theme_dark() +
  labs(title = "Datos DNase",x="densidad",y="frecuencia")
```

2. Con los datos de msleep realiza una grafica de barras o un *boxplot*, para visualizar las horas que los animales permanecen despiertos (awake, en eje de las y) vs el tipo de alimentación (vore, eje de las x). Dale formato a la gráfica y en el caso del boxplot agrega la distribución de los puntos con la capa *geom\_jitter*.

```
msleep %>% filter(!is.na(vore)) %>%
  ggplot(aes(x = vore, y = awake,color=vore)) +
  geom_boxplot() +
  geom_jitter(color="grey",size=1) +
  theme_bw()
```

[Regresar a Ejercicios 15](#)

## 11.16 Solución Ejercicios 16

1. Lea el archivo BacterialGrow.csv (sus campos estan separados por comas, no por tabuladores)

```
bactgrow<- read.csv("/data/BacterialGrow.csv")
```

2. Grafica el crecimiento de las diferentes bacterias vs el tiempo, en una gráfica de líneas. Ponga color a las líneas en términos de la clase de bacteria.
3. Poner títulos a los ejes y a la gráfica. Además, de cambiar el estilo del texto de las leyendas, en términos del tipo de letra y color de la misma.

```
head(bactgrow)

bact <- ggplot(data = bactgrow, aes(x = Time, y = Grow,color = Bacteria)) +
  geom_line() + theme_classic() +
  geom_point(shape=1) +
  labs(x="Time (minutes)", y="Total grow", title= "Bacterial growing") +
  theme(plot.title=element_text(size=15, family="Helvetica",face="bold", hjust=0.5),
        legend.text=element_text(face="italic",family="Times", colour="blue",size=10),
        legend.title = element_text(face="bold",family="Times", colour="blue",size=12))

bact
```

4. Guarde la imagen en un archivo

```
ggsave(filename="results/BacterialPlot.png",plot =bact, device= "png" )
```

[Regresar a Ejercicios 16](#)

## 12 Apéndices

### 12.1 Apéndice 1

#### 12.1.1 Iniciando R desde línea de comandos

Desde línea de comandos, en una terminal, escribir R y dar **enter**.

Automáticamente aparecerá en la pantalla información relacionada con la versión de R, los derechos reservados, cómo citar R, entre otras cosas más. Y al final aparecerá el símbolo de mayor que ">", que es el *prompt* de R.

#### 12.1.2 Manejo de paquetes desde línea de comandos

```
# Para listar los paquetes instalados
library()

# Para cargar un paquete en particular, en este caso ggplot2
library(ggplot2)

# Para visualizar los paquetes que ya están cargados
search()
## [1] ".GlobalEnv"          "package:ggplot2"      "package:flextable"
## [4] "package:dplyr"         "package:plyr"        "package:kableExtra"
## [7] "package:knitr"         "package:BiocStyle"   "package:stats"
## [10] "package:graphics"
## [ reached getOption("max.print") -- omitted 6 entries ]

# Para visualizar las funciones del paquete colocado en la posición 6
```

## Introducción a R y RStudio

```
ls(6)
## [1] "%>%"           "add_footnote"    "add_header_above" "add_indent"
## [5] "as_image"        "auto_index"      "cell_spec"       "collapse_rows"
## [9] "column_spec"     "footnote"
## [ reached getOption("max.print") -- omitted 43 entries ]
```

Para instalar un paquete utilizamos el comando:

```
install.package("knitr")
```

En este caso se instalaría el paquete llamado `knitr`

### 12.1.3 Consultando la ayuda desde línea de comandos

Existen diversas maneras de obtener ayuda sobre los comandos de R, a continuación veremos las más frecuentes. Una manera clásica de obtener ayuda respecto a un comando es utilizando la palabra `help(comando)` o `?comando`, por ejemplo, para consultar la ayuda del comando `solve`:

```
help(solve)
## Help on topic 'solve' was found in the following packages:
##
##   Package          Library
##   Matrix           /Library/Frameworks/R.framework/Versions/4.0/Resources/library
##   base             /Library/Frameworks/R.framework/Resources/library
##
## 
## Using the first match ...

?solve
## Help on topic 'solve' was found in the following packages:
##
##   Package          Library
##   Matrix           /Library/Frameworks/R.framework/Versions/4.0/Resources/library
##   base             /Library/Frameworks/R.framework/Resources/library
##
## 
## Using the first match ...
```

```
## solve          package:base          R Documentation
##
## Solve a System of Equations
##
## Description:
##
##   This generic function solves the equation 'a %*% x = b' for 'x',
##   where 'b' can be either a vector or a matrix.
##
## Usage:
##
```

## Introducción a R y RStudio

```
##      solve(a, b, ...)
##      ## Default S3 method:
##      solve(a, b, tol, LINPACK = FALSE, ...)
##
## Arguments:
##
##      a: a square numeric or complex matrix containing the
##          coefficients of the linear system. Logical matrices are
##          coerced to numeric.
##
##      b: a numeric or complex vector or matrix giving the right-hand
```

Para solicitar ayuda de funciones con caracteres especiales o algunas palabras reservadas, es necesario poner entre comillas la función o palabra reservada, ya que si no se hace así obtendremos un mensaje de error, por ejemplo, `for` es una palabra reservada de R, consultemos la ayuda sin poner comillas.

```
help(for)
## Error: <text>:1:9: unexpected ')'
## 1: help(for)
##           ^
```

Pero si colocamos la palabra entre comillas, obtendremos la ayuda, sin problemas.

```
help("for")
```

```
## Control                  package:base          R Documentation
##
## Control Flow
##
## Description:
##
##      These are the basic control-flow constructs of the R language.
##      They function in much the same way as control statements in any
##      Algol-like language. They are all reserved words.
```

**12.1.3.1 Ayuda en general** Para abrir la ayuda general en un navegador (sólo si tenemos la ayuda en HTML instalada y tenemos conexión a red), utilizamos:

```
help.start()
```

**12.1.3.2 Ayuda relacionada a un término** Podemos hacer búsquedas de información relacionada con un término en particular por medio de la función, `help.search`, la cual nos indicará los paquetes en donde se encuentra dicho término y una descripción muy pequeña del mismo. Por ejemplo:

```
help.search("clustering")
```

## Introducción a R y RStudio

```
## Vignettes with name or keyword or title matching 'clustering' using
## fuzzy matching:
##
## graph::clusterGraph      clusterGraph and distGraph
## Concepts: clustering
##
##
## Type 'vignette("FOO", package="PKG")' to inspect entries 'PKG::FOO'.
##
##
## Help files with alias or concept or title matching 'clustering' using
## fuzzy matching:
##
##
## Biostrings::stringDist
##                         String Distance/Alignment Score Matrix
## Concepts: Clustering
## Heatplus::oldCutplot.dendrogram
##                         Plot Subtrees of a Dendrogram in Different
##                         Colors
## Concepts: Clustering
```

**12.1.3.3 Ayuda sobre ejemplos de uso de una función** La función `example`, nos muestra ejemplos de ejecución de alguna función en particular, siempre y cuando la función que buscamos tenga en su descripción ejemplos.

```
example("data.frame")
##
## dt.frm> L3 <- LETTERS[1:3]
##
## dt.frm> fac <- sample(L3, 10, replace = TRUE)
##
## dt.frm> (d <- data.frame(x = 1, y = 1:10, fac = fac))
##       x y fac
## [ reached 'max' / getOption("max.print") -- omitted 10 rows ]
##
## dt.frm> ## The "same" with automatic column names:
## dt.frm> data.frame(1, 1:10, sample(L3, 10, replace = TRUE))
##       X1 X1.10 sample.L3..10..replace...TRUE.
## [ reached 'max' / getOption("max.print") -- omitted 10 rows ]
##
## dt.frm> is.data.frame(d)
## [1] TRUE
##
## dt.frm> ## do not convert to factor, using I():
## dt.frm> (dd <- cbind(d, char = I(letters[1:10])))
##       x y fac char
## [ reached 'max' / getOption("max.print") -- omitted 10 rows ]
```

## Introducción a R y RStudio

```
##  
## dtfrm> rbind(class = sapply(dd, class), mode = sapply(dd, mode))  
##      x          y         fac        char  
## [ reached getOption("max.print") -- omitted 2 rows ]  
##  
## dtfrm> stopifnot(1:10 == row.names(d)) # {coercion}  
##  
## dtfrm> (d0 <- d[, FALSE]) # data frame with 0 columns and 10 rows  
## data frame with 0 columns and 10 rows  
##  
## dtfrm> (d.0 <- d[FALSE, ]) # <0 rows> data frame (3 named cols)  
## [1] x  
## [ reached getOption("max.print") -- omitted 2 entries ]  
## <0 rows> (or 0-length row.names)  
##  
## dtfrm> (d00 <- d0[FALSE, ]) # data frame with 0 columns and 0 rows  
## data frame with 0 columns and 0 rows
```

**12.1.3.4 Ayuda sobre argumentos de una función** También se puede solicitar ayuda relacionada a los argumentos de una determinada función.

```
args(data.frame)  
## function (... , row.names = NULL, check.rows = FALSE, check.names = TRUE,  
##           fix.empty.names = TRUE, stringsAsFactors = default.stringsAsFactors())  
## NULL
```

**12.1.3.5 Búsquedas aproximadas** Finalmente, una alternativa para realizar búsquedas por aproximación, es decir, poniendo sólo una parte del término que busacamos, es con el comando **apropos**.

```
apropos("solve")  
## [1] "backsolve"      "forwardsolve"    "qr.solve"       "solve"  
## [5] "solve.default"   "solve.qr"
```

## 12.2 Guardando y Cargando el Espacio de Trabajo desde Línea de comandos

Para almacenar y cargar los objetos que se crearon en una sesión de trabajo, utilizamos las funciones **save.image** y **load**.

```
save.image(file = "archivo.Rdata")
```

Recordemos que con la función **dir()** podemos consultar los archivos del directorio de trabajo, para verificar que se creó el archivo **archivo.Rdata**.

Para almacenar el archivo en un directorio diferente al de trabajo, se requiere especificar la ruta.

Si queremos llamar a los objetos almacenados en el archivo **archivo.Rdata**, utilizamos:

```
load(file = "archivo.Rdata")
```

Si el archivo no se encuentra en el directorio de trabajo, es necesario especificar la ruta en donde se encuentra.

### 12.2.1 Almacenando, en un archivo, objetos específicos creados en una sesión

Cuando queremos guardar uno o varios objetos creados en una sesión, usamos el comando `save`.

```
save(objeto, file = "archivo.Rdata")
save(objeto1,objeto2, objetoX, file = "archivo.Rdata")
```

Para cargar estos archivos utilizamos nuevamente la función `load`.

### 12.2.2 Guardando y Cargando el Historial de Comandos desde Línea de comandos

Si queremos guardar la historia de los comandos utilizados en un momento dado, lo hacemos con el comando `savehistory`. Por ejemplo:

```
savehistory(file = "archivo.Rhistory")
```

Utilicemos `dir()` para verificar que el archivo se creó

Para cargar el archivo `archivo.Rhistory` utilizamos la instrucción `loadhistory`.

```
loadhistory(file = "archivo.Rhistory")
```

## 12.3 Apeéndice 2

### 12.3.1 Manipulación de Data frames con `dplyr`

#### 12.3.2 ¿Qué es `dplyr`?

El paquete `dplyr` es un poderoso paquete de R que proporciona un conjunto de funciones (o verbos) útiles para manejar y resumir datos tabulares, como dataframes, de una manera fácil. Las funciones más comunes

En esta sección revisaremos 5 de sus funciones más comunes que se utilizan en el análisis de datos, así como el uso de `pipes` para combinarlas. Estas funciones son:

1. `select`
2. `filter`
3. `group_by`
4. `summarize`
5. `mutate`

Recordemos que para utilizar este paquete debemos cargarlo:

```
library("dplyr")
```

### 12.3.3 Gramática de `dplyr`

`dplyr` puede ser vista como una gramática para el manejo de datos, donde cada función es un verbo que trabaja de la siguiente manera:

1. El primer argumento es el nombre del data.frame
2. Los siguientes argumentos describen qué se hace con el data.frame usando los nombres de variables.
3. El resultado es un nuevo data.frame

### 12.3.4 La función `select()`

Es común tener conjuntos de datos con muchas variables. En este caso lo primero que queremos hacer es enfocarnos solo en algunas de ellas.

`select()`, nos permite seleccionar un subconjunto de variables. Por ejemplo, si quisieramos trabajar solo con las columnas `name`, `brainwt` y `bodywt` de nuestro 'data.frame' `msleep`, lo haríamos de la siguiente manera:

```
name_brainwt_bodywt<-select(msleep,name,brainwt,bodywt)

head(name_brainwt_bodywt)
## # A tibble: 6 x 3
##   name                  brainwt   bodywt
##   <chr>                 <dbl>     <dbl>
## 1 Cheetah                NA       50
## 2 Owl monkey             0.0155    0.48
## 3 Mountain beaver        NA       1.35
## 4 Greater short-tailed shrew 0.00029  0.019
## 5 Cow                     0.423    600
## 6 Three-toed sloth       NA       3.85
```

En este caso hemos usado la gramática "normal", pero la fortaleza de `dplyr` consiste en combinar funciones usando pipes (`%>%`). Es decir,

```
msleep %>% select(name,brainwt,bodywt) %>% head
## # A tibble: 6 x 3
##   name                  brainwt   bodywt
##   <chr>                 <dbl>     <dbl>
## 1 Cheetah                NA       50
## 2 Owl monkey             0.0155    0.48
## 3 Mountain beaver        NA       1.35
## 4 Greater short-tailed shrew 0.00029  0.019
## 5 Cow                     0.423    600
## 6 Three-toed sloth       NA       3.85
```

## Introducción a R y RStudio

El concepto de `pipe` (tuberías) es el mismo que en `Unix`. En este caso `%>%` representa el `pipe`, el cual indica que queremos usar el comando de la izquierda como entrada al comando de la derecha del pipe.

### 12.3.5 La función `filter()`

`filter()` permite filtrar observaciones a partir de sus valores. El primer argumento es el nombre de dataframe. El segundo y los siguientes argumentos son las expresiones de filtro. Por ejemplo, si queremos obtener la lista de todas las especies domesticadas de `msleep`, filtraremos los datos en la columna `conservation`.

```
msleep %>% filter(conservation=="domesticated") %>% head
## # A tibble: 6 x 11
##   name   genus  vore  order conservation sleep_total sleep_rem sleep_cycle awake
##   <chr>  <chr> <chr> <chr>      <dbl>      <dbl>      <dbl> <dbl>
## 1 Cow    Bos    herbi Arti~ domesticated     4       0.7      0.667  20
## 2 Dog    Canis  carni Carn~ domesticated   10.1      2.9      0.333  13.9
## 3 Guine~ Cavis  herbi Rode~ domesticated   9.4       0.8      0.217  14.6
## 4 Chinc~ Chinc~ herbi Rode~ domesticated  12.5      1.5      0.117  11.5
## 5 Horse  Equus  herbi Peri~ domesticated   2.9       0.6      1        21.1
## 6 Donkey Equus  herbi Peri~ domesticated   3.1       0.4      NA      20.9
## # ... with 2 more variables: brainwt <dbl>, bodywt <dbl>
```

Recordemos que podemos combinar funciones utilizando pipes. Por ejemplo, si queremos obtener sólo los nombres de las especies domesticadas utilizaremos `filter` y `select`

```
msleep %>% filter(conservation=="domesticated") %>% select(name) %>% head
## # A tibble: 6 x 1
##   name
##   <chr>
## 1 Cow
## 2 Dog
## 3 Guinea pig
## 4 Chinchilla
## 5 Horse
## 6 Donkey
```

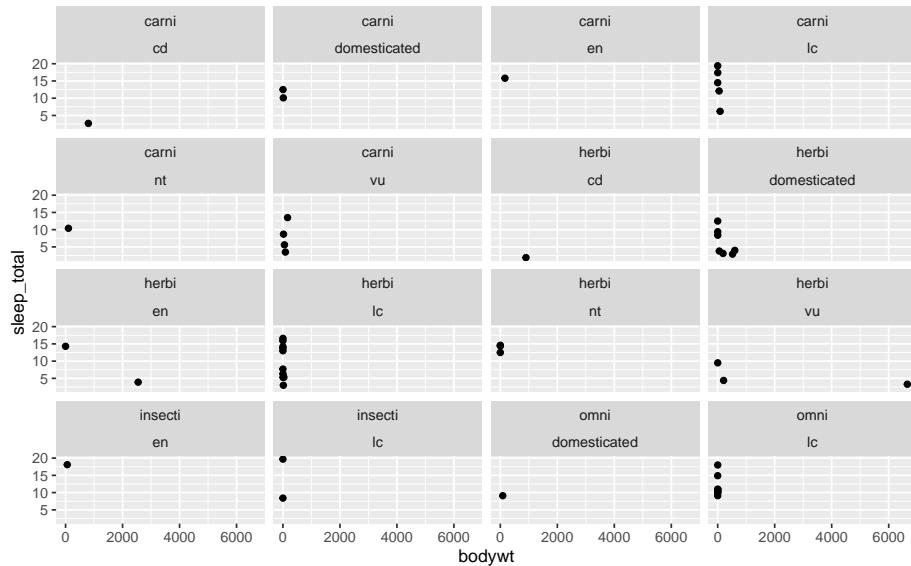
## 12.4 Apéndice 3

### 12.5 `facet_wrap()`

También podemos dividir en paneles las gráficas utilizando `facet_wrap()`. En este caso trata de ordenar los paneles en el mismo número de renglones y columnas. Por ejemplo:

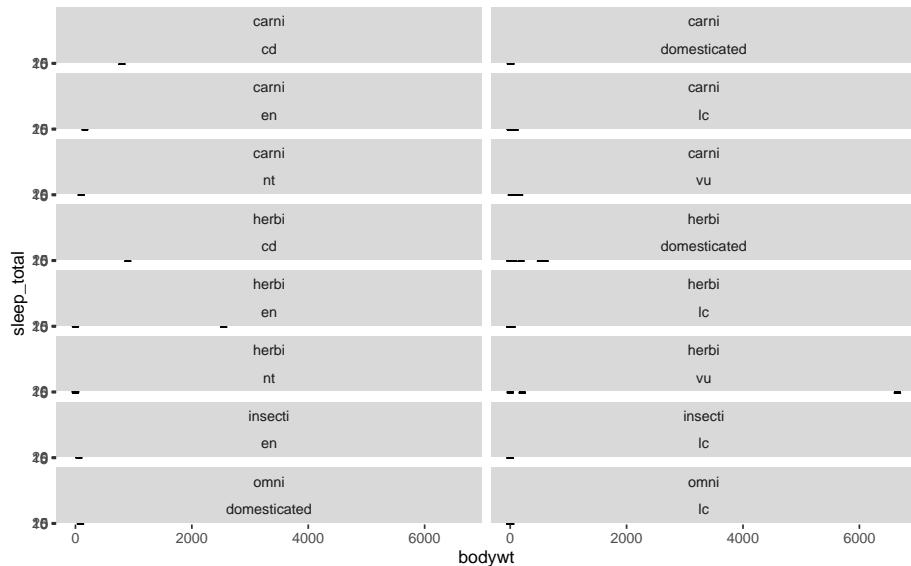
```
keep_rows <- !(is.na(msleep$vore)) & !(is.na(msleep$conservation))
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total)) +
       geom_point() +
       facet_wrap(vore ~ conservation)
```

## Introducción a R y RStudio



Podemos cambiar el número de columnas que queremos que se despliegue, por ejemplo, lo podemos poner en 2, por medio del parámetro `ncol`.

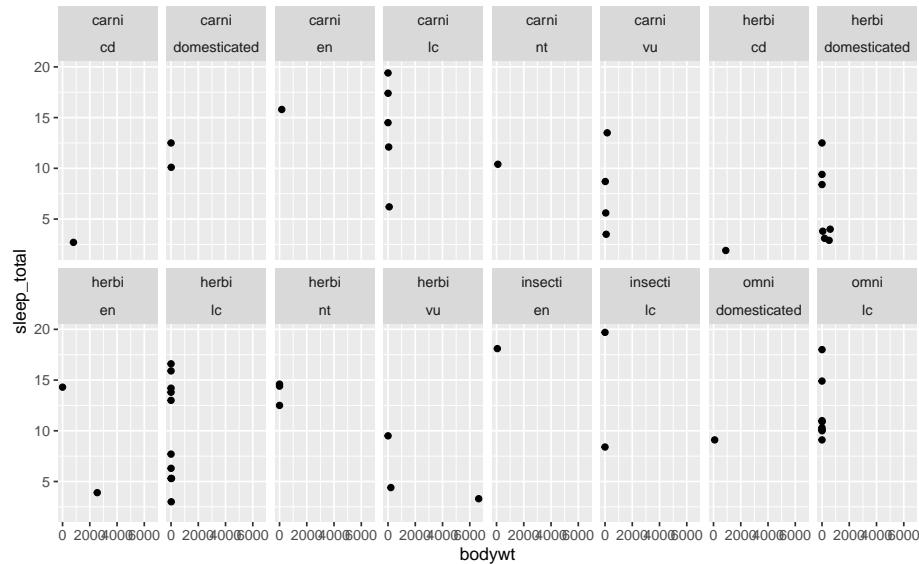
```
keep_rows <- !is.na(msleep$vore) & !is.na(msleep$conservation)
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total)) +
       geom_point() +
       facet_wrap(vore ~ conservation, ncol = 2)
```



Por otro lado, cuando graficamos, solo renglones o solo columnas, por defecto lo organiza por poniendo 3 elementos por renglón. Sin embargo, podemos cambiar el orden por medio de los parámetros `ncol` y `nrow`.

```
keep_rows <- !is.na(msleep$vore) & !is.na(msleep$conservation)
ggplot(data = msleep[keep_rows, ],
       aes(x = bodywt, y = sleep_total)) +
```

```
geom_point( ) +
facet_wrap(vore ~ conservation, nrow = 2)
```



## 12.6 Apéndice 4

### 12.6.1 Cambiando el fondo de mi gráfica de manera personalizada

Para cambiar las características de gráfica, de forma personalizada, en términos del fondo y las líneas divisorias (*grid*), entre otras, hacemos uso de la capa `theme`. Por ejemplo, vamos a modificar el `panel` principal de la gráfica. Los elementos que modificaremos serán:

1. Fondo (background)
2. Borde (border)
3. Líneas divisorias (grid.major, grid.minor)

```
keep_rows <- !(is.na(msleep$vore))
g1<-ggplot(data = msleep[keep_rows, ],
            aes(x = bodywt, y = sleep_total,
                color = vore)) +
            geom_point( ) +
            labs(title="Relación entre el peso y las horas de sueño",
                 subtitle="El peso está definido en Kg",
                 x="Peso del animal", y="Horas de sueño")

## Modificando el fondo
g1 <-g1 +
  theme(panel.background=
        element_rect(fill="lightyellow2")) +
  theme(panel.border=
        element_rect(color="black",
                    fill=NA,size=1.5))
```

## Introducción a R y RStudio

```
## Modificando el grid, mayor y menor
g1 <- g1 +
  theme(panel.grid.major=
    element_line(colour="grey70")) +
  theme(panel.grid.minor=
    element_line(colour="grey 60",
                linetype="dashed"))

## Modificando el tema de las leyendas

g1 <- g1 +
  theme(legend.background=
    element_rect(color="black",
                 fill="khaki")) +
  theme(legend.key=element_rect(fill=NA))

print(g1)
```



### 12.6.2 Asignando el diseño a un objeto

Si el diseño que creamos es el que vamos a utilizar en diversas gráficas podemos asignar estas capas a un objeto y utilizarlo en otras gráficas. Por ejemplo,

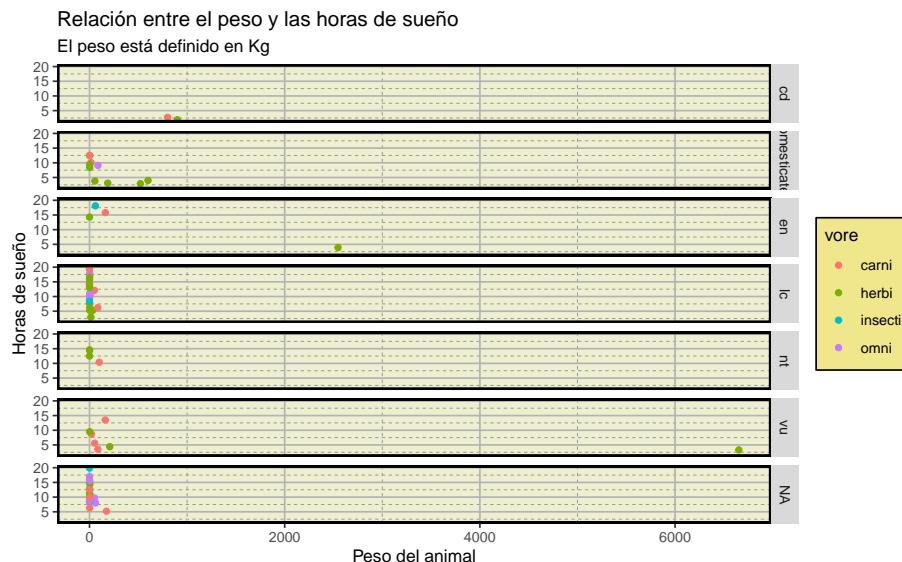
```
Mitema<-
  theme(panel.background=
    element_rect(fill="lightyellow2")) +
  theme(panel.border=
    element_rect(color="black",
                fill=NA,size=1.5)) +
  theme(panel.grid.major=
    element_line(colour="grey70")) +
  theme(panel.grid.minor=
```

## Introducción a R y RStudio

```
element_line(colour="grey 60",
             linetype="dashed")) +
theme(legend.background=
  element_rect(color="black",
               fill="khaki")) +
theme(legend.key=element_rect(fill=NA))

keep_rows <- !(is.na(msleep$vore))
g1<-ggplot(data = msleep[keep_rows, ],
            aes(x = bodywt, y = sleep_total,
                color = vore)) +
  geom_point() +
  labs(title="Relación entre el peso y las horas de sueño",
       subtitle="El peso está definido en Kg",
       x="Peso del animal", y="Horas de sueño")

g1 + facet_grid(conservation ~.) + Mitema
```



## 12.7 Apéndice 5

### 12.7.1 Gráfica de Pie

```
msleep %>% filter(!is.na(vore)) %>% count(vore) %>%
  mutate(porcent=(n/sum(n)*100)) %>%
  ggplot(aes(x="", y=porcent, fill=vore)) +
  geom_bar(width = 1, stat = "identity") +
  coord_polar("y", start=0) + theme_void() +
  geom_text(aes(label = round(porcent,1)), position = position_stack(vjust = 0.5))
```

