

# ESCUELA DE $R$ - ESTADÍSTICA

## 3RA EDICIÓN - 2017

### Introducción al uso de $R$ como lenguaje de programación estadístico

Miguel Flores S.\*    Andrés Peña M. †

*R Users Group - Ecuador*

*Quito-Ecuador*

Octubre 2017

#### Resumen

El presente módulo introduce al participante en el manejo del software  $R$ , lo cual comprende la instalación y configuración del software necesario para el entorno de programación estadístico, la discusión de los conceptos del lenguaje de programación genéricos. Además, temas teórico-prácticos de la estadística, programación básica con el uso del script, manipulación de bases de datos, funciones y herramientas gráficas.

**Palabras Clave:** software  $R$ , programación, estadística, bases de datos.

---

\*Docente-Investigador del Departamento de Matemática de la Facultad de Ciencias de la Escuela Politécnica Nacional. Contacto: [miguel.flores@epn.edu.ec](mailto:miguel.flores@epn.edu.ec)

†Analista de Estadísticas Económicas del Instituto Nacional de Estadística y Censos (INEC). Contacto: [andres.pena@epn.edu.ec](mailto:andres.pena@epn.edu.ec)

# Índice

<b>1. Objetivos del curso</b>	<b>1</b>
<b>2. Visión general de <i>R</i>: instalación <i>RStudio</i> y otros</b>	<b>1</b>
2.1. Introducción . . . . .	1
2.2. Ventajas . . . . .	1
2.3. Desventajas . . . . .	2
2.4. Cómo funciona <i>R</i> . . . . .	2
<b>3. Instalación de <i>R</i>, <i>RStudio</i> y paquetes</b>	<b>2</b>
3.1. Instalación de <i>R</i> . . . . .	2
3.2. <i>R</i> como una calculadora . . . . .	3
3.3. Instalación de <i>RStudio</i> . . . . .	3
3.4. Instalación de paquetes . . . . .	4
<b>4. Data management, parte I</b>	<b>5</b>
4.1. Estructuras de Datos . . . . .	5
4.1.1. Vectores . . . . .	6
4.1.2. Listas . . . . .	13
4.1.3. Atributos . . . . .	15
4.1.4. Matrices . . . . .	15
4.1.5. Factores . . . . .	17
4.1.6. Data Frame . . . . .	19
4.2. R Data Frames . . . . .	24
4.3. Función structure . . . . .	26
4.4. Missing Values . . . . .	27
<b>5. Importación y Exportación de datos</b>	<b>27</b>
5.1. Lectura de datos . . . . .	27
5.2. Directorio de trabajo . . . . .	28
5.3. ¿Cómo leer archivos en R? . . . . .	29
5.3.1. Lectura de archivos .txt . . . . .	29
5.3.2. Lectura de archivos .csv . . . . .	30
5.3.3. Lectura de archivos .xls, .xlsx . . . . .	30
5.3.4. Lectura de archivos .sav . . . . .	31

<b>6. Estructuras de control y funciones:</b>	<b>31</b>
6.1. Introducción . . . . .	31
6.2. Porqué construir funciones? . . . . .	32
6.3. Estructura de una función . . . . .	32
6.4. Argumentos . . . . .	33
6.5. Argumentos por default . . . . .	33
6.6. Lexical Scoping . . . . .	34
6.7. Estructuras de Control . . . . .	35
6.7.1. Sentencia if . . . . .	35
6.7.2. Sentencia else . . . . .	36
6.7.3. Sentencia if else . . . . .	36
6.7.4. Sentencia for . . . . .	37
6.7.5. Ejercicios sobre Estructuras de Control . . . . .	38
<b>7. Introducción a gráficas “ggplot”</b>	<b>38</b>
7.1. Sistemas gráficos de R . . . . .	38
7.2. Sistemas Gráficos Base . . . . .	39
7.3. Facetas . . . . .	40
7.4. Sistema gráfico Lattice . . . . .	40
7.5. Sistema gráfico ggplot2 . . . . .	41
7.6. Generación el primer gráfico ggplot2 . . . . .	41
7.7. Histogramas . . . . .	42
7.8. Curvas de densidad . . . . .	42
7.9. Diagrama de Caja bigotes . . . . .	43
7.10. Diagrama de Barras . . . . .	43
7.11. Diagrama de pie . . . . .	44
7.12. Gráficos múltiples de distribución . . . . .	44
7.13. Gráfico de linea básico . . . . .	45
7.14. Gráfico de lineas múltiples . . . . .	45
7.15. Generación de gráficos ggplot . . . . .	46
<b>8. Bibliografía</b>	<b>48</b>

# Índice de figuras

1.	Autores de R . . . . .	1
2.	Funcionamiento de R . . . . .	2
3.	Instalación de R . . . . .	3
4.	Instalación de RStudio . . . . .	4
5.	Estructuras de datos en R . . . . .	5
6.	Función combinar . . . . .	7
7.	Operaciones con vectores . . . . .	10
8.	Coerción de vectores . . . . .	12
9.	Lista en R . . . . .	13
10.	Elementos de una lista . . . . .	14
11.	Ejemplo de factor en R . . . . .	18
12.	Ejemplo de factor en R . . . . .	20
13.	Extracción de elementos de un Data Frame . . . . .	22
14.	Varios formatos que lee el R . . . . .	28
15.	Cómo lee los archivos el R . . . . .	29
16.	Función . . . . .	31
17.	Estructura de una Función . . . . .	32
18.	Sentencia If . . . . .	35
19.	Declaración Sentencia If . . . . .	35
20.	Sentencia else . . . . .	36
21.	Sentencia if else . . . . .	36
22.	Sentencia for . . . . .	37
23.	Declaración Sentencia for . . . . .	37
24.	Sistema Gráfico de R . . . . .	38

# 1. Objetivos del curso

El objetivo del curso es actualizar-ampliar los conocimientos en la optimización de la gestión de información cuantitativa empoderando al participante de destrezas en el manejo del software R en data management, estadística descriptiva, gráficos, modelos lineales, clasificación, series de tiempo, con base a aplicaciones y casos prácticos.

- Introducir al participante en el uso del software R como lenguaje de programación.
- Mejorar las habilidades de gestión de información del participante.
- Incrementar el conocimiento de modelos y su aplicación.

## 2. Visión general de *R*: instalación *RStudio* y otros

### 2.1. Introducción

*R* es un sistema para análisis estadístico y gráficos creado por Robert Gentleman y Ross Ihaka, profesores del departamento de Estadística de University of Auckland - New Zealand.

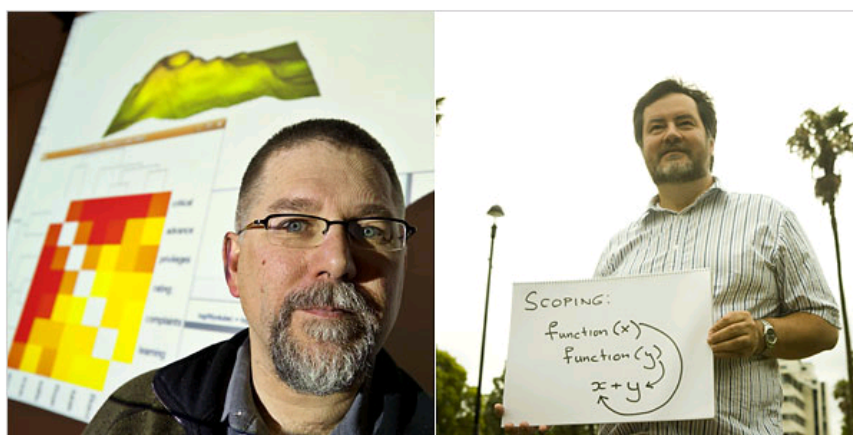


Figura 1: Autores de R

### 2.2. Ventajas

- Open Source (Costo nulo)
- Comunidad dinámica integrada por estadísticos de renombre (R User Group Ecuador, Stackoverflow, Comunidad R Hispano, R en español, ...)
- Multi-plataforma: Disponible para Windows, Mac, Linux y Android.
- Es de código abierto.
- Facilidad de enlace con LaTeX y generar reportes dinámicos.

- Amplia bibliografía en la web y libros publicados por editoriales prestigiosas (Springer, Wiley, Chapman & Hall, ...)
- Lenguaje intuitivo para la lectura de funciones y algoritmos.

## 2.3. Desventajas

- Por ser un programa libre, no posee garantía ni un departamento de atención al cliente.
- *R* no posee una interfaz amigable, las tareas se llevan a través de líneas de código, lo cual puede ser incómodo para el usuario.
- *R* particularmente no es un lenguaje de programación rápida.

## 2.4. Cómo funciona *R*

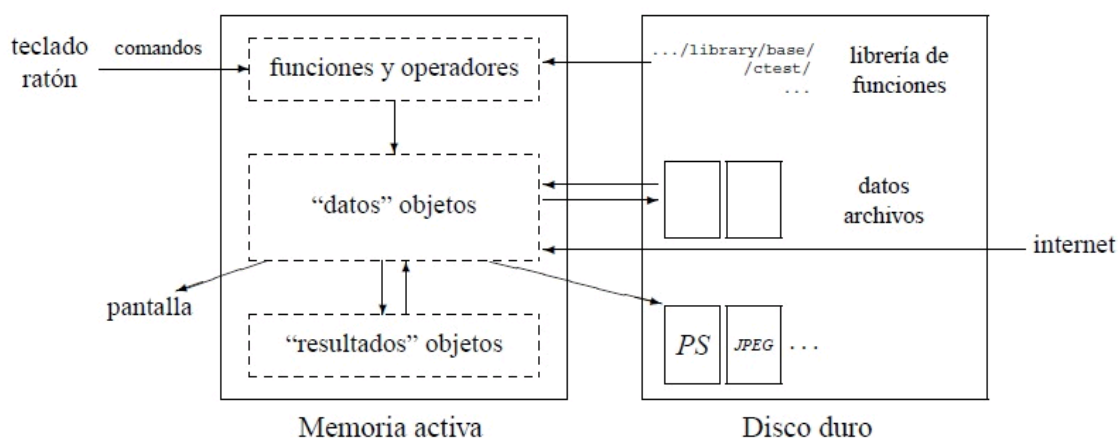


Figura 2: Funcionamiento de R

## 3. Instalación de *R*, *RStudio* y paquetes

### 3.1. Instalación de *R*

- Ingresar a <https://www.r-project.org/>
- Elegir el CRAN (Comprehensive R Archive Network)
- Escoger versión del sistema operativo.
- Instalar R por primera vez.

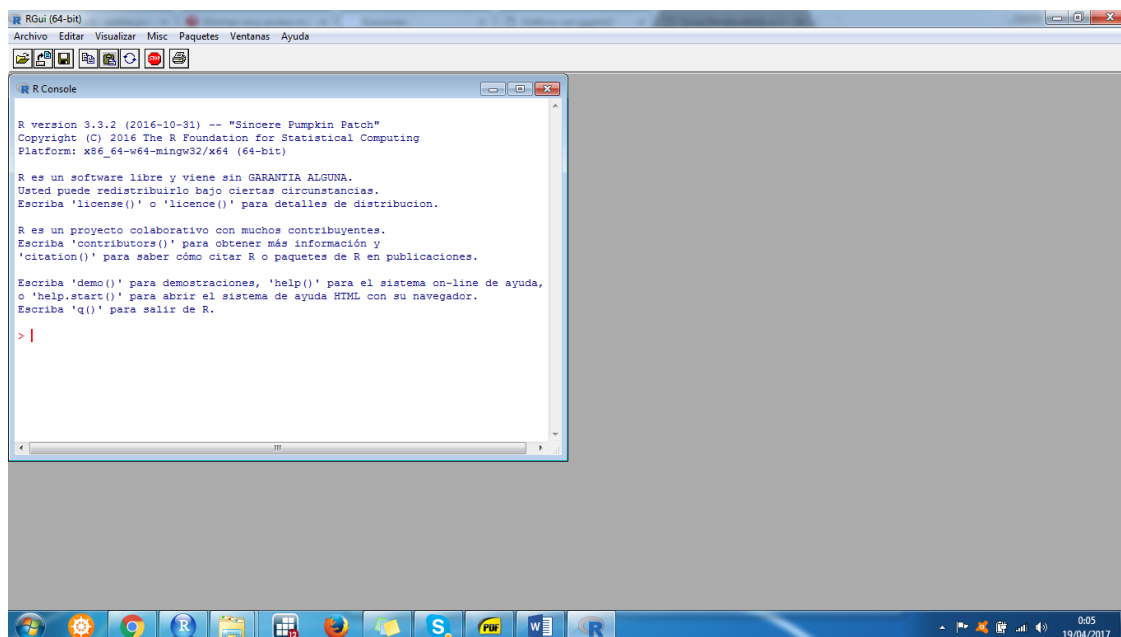


Figura 3: Instalación de R

### 3.2. R como una calculadora

```
3*5
```

```
## [1] 15
```

```
12.3/2.6
```

```
## [1] 4.7
```

```
sqrt(16)
```

```
## [1] 4
```

Además, cada operación anterior puede ser almacenada en un “objeto”

```
a<-3*5
```

```
b<-12.3/2.6
```

```
D<-sqrt(16)
```

### 3.3. Instalación de RStudio

- Ingresar a <https://www.rstudio.com/>
- Productos;

- Escoger versión según el Sistema Operativo.

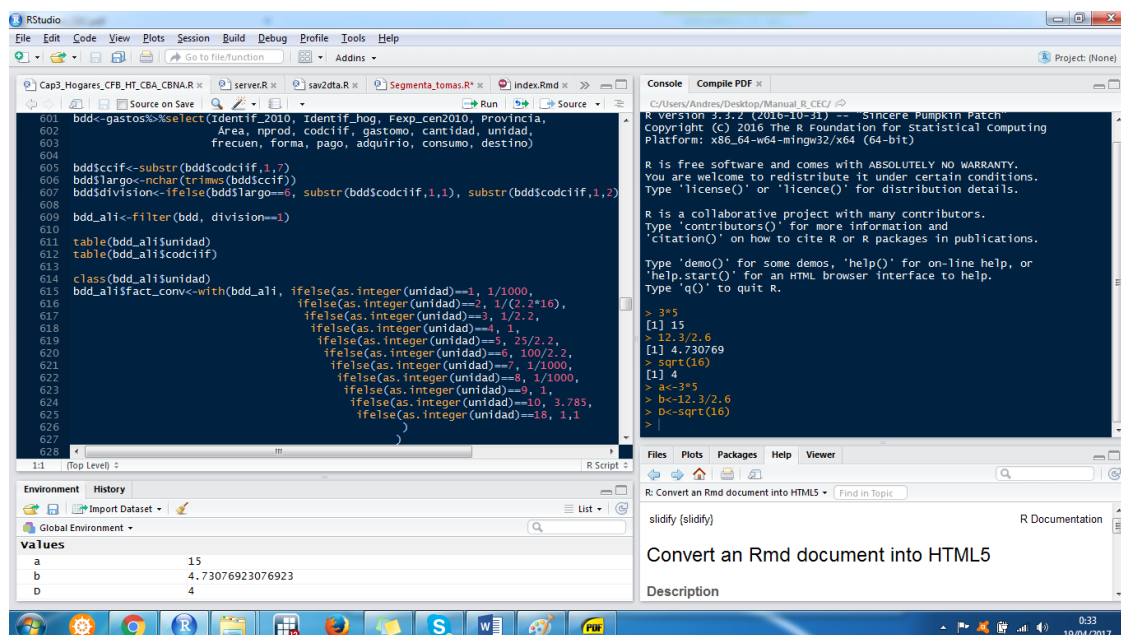


Figura 4: Instalación de RStudio

### 3.4. Instalación de paquetes

- Además de las funciones básicas, *R* tiene un gran número de paquetes especializados.
- Los paquetes se instalan una sola vez y deben ser cargados en cada inicio de sesión.

Se utiliza la siguiente función:

```
install.packages("pkgname", dependencies = TRUE)
```

Una vez instalado, debemos cargarlo con el comando:

```
library(pkgname)
require(pkgname)
```

Para instalar un paquete desde otro repositorio:

```
install.packages("pkgname", repos="http://r-forge.r-project.org", type="source")
```

Para poder ver los paquetes instalados y disponibles podemos utilizar:

```
library()
installed.packages()
.packages(all.available=TRUE)
available.packages()
```



Para poder saber cuál fue el último paquete instalado utilizamos:

```
(.packages())
```

```
## [1] "knitr"      "stats"      "graphics"   "grDevices"  "utils"      "datasets"
```

```
## [7] "methods"   "base"
```

Es importante saber que funciones tiene cada paquete, para esto utilizamos:

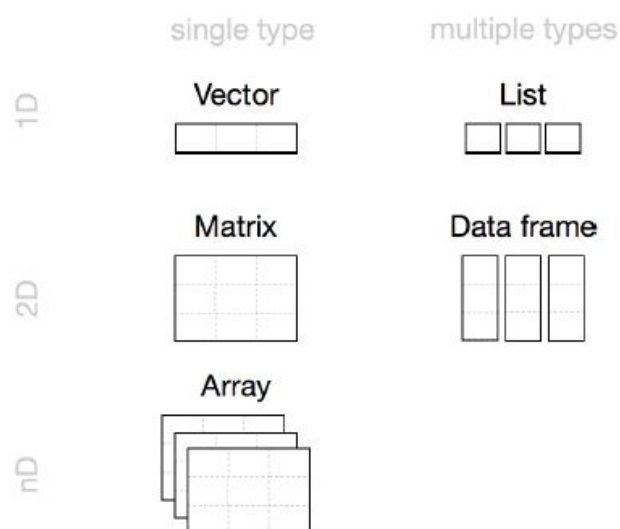
```
ls("package:pckgname")
```

## 4. Data management, parte I

### 4.1. Estructuras de Datos

Las estructuras de datos en R se organizan por:

- Dimensionalidad; y
- Homogeneidad o heterogeneidad.



**Figura 5:** Estructuras de datos en R

R no tiene estructuras 0d (tipo escalar). Los números y strings individuales son considerados como vectores de longitud 1.

```
# no mostrar advertencias
options(warn = -1)
x <- 5
x
```

```
## [1] 5

y <- "SSL"
y

## [1] "SSL"
```

El texto luego de `#` es un comentario (no es interpretado por *R*). Para crear un objeto se utiliza el operador de asignación `<-`.

#### 4.1.1. Vectores

La estructura de datos básica de R son los vectores, estos se dividen en:

- Vectores atómicos; y
- Listas.

##### Propiedades:

Tipo: ¿Qué es?

```
typeof(x)
```

Longitud: Número de elementos.

```
length(x)
```

Atributos: Características.

```
attributes(x)
```

**Vectores Atómicos:** Los elementos de un vector atómico son del mismo tipo a diferencia de los elementos de una lista que pueden ser de diferente tipo. Los tipos comunes son:

- double (numeric);
- integer;
- character;
- logical.



Figura 6: Función combinar

Un vector es creado mediante la función `c()` (combinar).

```
vec <- c(1, 2)
vec

## [1] 1 2
```

### Tipos de Vectores Atómicos:

- Vector double;

```
dbl_vec <- c(3.5, 2, -1)
```

- Vector entero: Use el sufijo L para crear un vector entero;

```
int_vec <- c(3L, 7L, 1L)
```

- Vector character: Use “ ” para crear un vector character;

```
chr_vec <- c("Science", "&", "Apps")
```

- Vector lógico: Use TRUE y FALSE o T y F para crear un vector lógico.

```
log_vec <- c(FALSE, TRUE, F, T)
```

Para determinar si un vector `vec` es atómico utilizamos `is.atomic(vec)`.

```
vec <- c(3.5, 2, -1)
is.atomic(vec)

## [1] TRUE
```

Para determinar el tipo de un vector `vec` utilizamos `typeof(vec)`.

```
vec <- c("Science", "&", "Apps")
typeof(vec)

## [1] "character"
```

Para verificar si un vector `vec` es de un tipo en específico, se utilizan las funciones “is”:

```
is.character(vec)
is.double(vec)
is.integer(vec)
is.logical(vec)
```

**Elementos de un vector atómico:** La componente  $i$  de `vec` se obtiene mediante `vec[i]`.

6	1	3	6	10	5
---	---	---	---	----	---

`vec[5]`

Componente 5 de `vec`:

```
vec <- c(6, 1, 3, 6, 10, 5)
vec[5]

## [1] 10
```

Para seleccionar varios elementos utilizamos `vec[c(elementos)]`.

```
# elementos 2 y 4
vec[c(2, 4)]

## [1] 1 6
```

Para omitir el elemento  $i$  de `vec` se utiliza `vec[-i]`.

```
vec <- c(6, 1, 3, 6, 10, 5)
vec[-5]

## [1] 6 1 3 6 5
```

Para omitir varios elementos utilizamos `vec[-c(elementos)]`.

```
# omitir los elementos 2 y 4
vec[-c(2, 4)]

## [1] 6 3 10 5

# omitir los elementos del 1 al 3
vec[-c(1, 2, 3)]

## [1] 6 10 5
```

**Generación de secuencias:** El operador `a:b` genera el vector  $a, a+1, a+2, \dots, b$ .

```
1:10
```

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

```
5:-5
```

```
## [1] 5 4 3 2 1 0 -1 -2 -3 -4 -5
```

La función `seq()` genera secuencias controlando: inicio, fin y salto.

```
seq(from = 1, to = 10, by = 0.5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5  
## [15] 8.0 8.5 9.0 9.5 10.0
```

```
seq(from = -4, to = 4, length = 5)
```

```
## [1] -4 -2 0 2 4
```

`vec[c(TRUE, FALSE)]` permite obtener determinados elementos:

6	1	3	6	10	5
---	---	---	---	----	---

```
vec[c(F, T, F, T, F, T)]
```

```
# elementos 2, 4, 6  
vec <- c(6, 1, 3, 6, 10, 5)  
vec[c(FALSE, TRUE, FALSE, TRUE, FALSE, TRUE)]  
  
## [1] 1 6 5
```

Equivalente a:

```
vec[c(2, 4, 6)]  
  
## [1] 1 6 5
```

Utilice `TRUE`, `FALSE` en lugar de `T`, `F`.

**Operaciones con vectores:** A continuación, se resumen las operaciones básicas con vectores atómicos:

Operación	Descripción
$a + b$	Suma
$a - b$	Resta
$a * b$	Multiplicación
$x / y$	División
$x \wedge y$	Potencia
$\text{sqrt}(x)$	Raíz cuadrada
$\text{abs}(x)$	Valor absoluto
$\text{exp}(x)$	Exponencial
$\text{log}(x, \text{base}=n)$	Logaritmo en base n
$\text{factorial}(x)$	Factorial
$x \% \% y$	Módulo
$x \% / \% y$	División entera
$x == y$	Test de igualdad
$x != y$	Test de desigualdad
$x <= y$	Test menor o igual que
$x >= y$	Test mayor o igual que
$x \&\& y$	Conjunción para escalares
$x    y$	Disyunción para escalares
$x \& y$	Conjunción para vectores
$x   y$	Disyunción para vectores
$!x$	Negación

**Figura 7:** Operaciones con vectores

Función `all(x)` retorna TRUE si todas las componentes de x son TRUE.

```
x <- c(TRUE, TRUE, TRUE)
all(x)

## [1] TRUE

y <- c(TRUE, FALSE, TRUE)
all(y)

## [1] FALSE
```

Función `any(x)` retorna TRUE si al menos una componente de x es TRUE.

```
x <- c(FALSE, FALSE, FALSE)
any(x)

## [1] FALSE

y <- c(FALSE, TRUE, FALSE)
any(y)

## [1] TRUE
```

Negación `!x` cambia las componentes de x de: TRUE a FALSE y de FALSE a TRUE:

```
x <- c(TRUE, FALSE, TRUE)
!x

## [1] FALSE TRUE FALSE
```

Conjunción  $x \& y$  retorna TRUE solo si las componentes de  $x \& y$  son TRUE:

```
x <- c(TRUE, TRUE, FALSE)
y <- c(FALSE, TRUE, TRUE)
x & y

## [1] FALSE TRUE FALSE

x && y # conjuncion para escalares

## [1] FALSE
```

Disyunción  $x | y$  retorna TRUE si al menos una componente de  $x$  o  $y$  es TRUE

```
x <- c(TRUE, TRUE, FALSE)
y <- c(FALSE, TRUE, TRUE)
x | y

## [1] TRUE TRUE TRUE

x || y # disyuncion para escalares

## [1] TRUE
```

Dado un vector lógico  $x$ :

```
x <- c(TRUE, TRUE, FALSE, TRUE, FALSE)
```

$\text{sum}(x)$ . Cuenta el número de TRUE en  $x$ .

```
sum(x)

## [1] 3
```

$\text{mean}(x)$ . Muestra el porcentaje de TRUE en  $x$ .

```
mean(x)

## [1] 0.6

sum(x)/length(x)

## [1] 0.6
```

Test de igualdad y desigualdad

```
x <- c(3, 5, 2, 3, 1)
x==3

## [1] TRUE FALSE FALSE TRUE FALSE

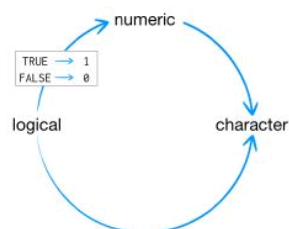
y <- x==3 # componentes iguales a 3
z <- x!=3 # componentes distintas a 3
data.frame(x, y, z)

##   x     y     z
## 1 3  TRUE FALSE
## 2 5 FALSE  TRUE
## 3 2 FALSE  TRUE
## 4 3  TRUE FALSE
## 5 1 FALSE  TRUE

x[x!=3] # componentes de x distintas a 3

## [1] 5 2 1
```

**Coerción:** Si combinamos tipos diferentes, serán coercionados al tipo más flexible dado por la jerarquía:



$\text{character} \leq \text{double} \leq \text{integer} \leq \text{logical}$

**Figura 8:** Coerción de vectores

Para coercionar un vector  $x$  a un determinado tipo, se utilizan las funciones “as”

```
as.character(x)

as.double(x)

as.integer(x)

as.logical(x)

as.numeric(x)
```



### 4.1.2. Listas

Una lista es un vector que puede contener elementos de cualquier tipo y de distinta longitud.



Figura 9: Lista en R

Para crear una lista se utiliza la función `list()` en lugar de `c()`.

```
lst <- list(c(1, 2), c(TRUE), c("a", "b", "c"))
lst

## [[1]]
## [1] 1 2
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] "a" "b" "c"
```

Tests para una lista:

```
lst <- list(1:3, c("Source", "Stat", "Lab"), c(TRUE, FALSE), c(1.3, 4.5))
typeof(lst)

## [1] "list"

is.atomic(lst)

## [1] FALSE
```

Para probar si `lst` es una lista se utiliza:

```
is.list(lst)

## [1] TRUE
```

Coerción de vector atómico `lst` a una lista

```
lst <- c("ssl", 4, 0.5)
as.list(lst)

## [[1]]
```

```
## [1] "ssl"
##
## [[2]]
## [1] "4"
##
## [[3]]
## [1] "0.5"
```

Coerción de lista lst a vector atómico (reglas de coerción)

```
lst <- list(1:3, c("Science", "&", "Apps"), c(TRUE, FALSE), c(1.3, 4.5))
unlist(lst)

## [1] "1" "2" "3" "Science" "&" "Apps" "TRUE"
## [8] "FALSE" "1.3" "4.5"
```

**Elementos de una lista:** Para acceder al elemento i de la lista se utiliza x[i]. Para acceder al objeto que contiene el elemento i se utiliza x[[i]].

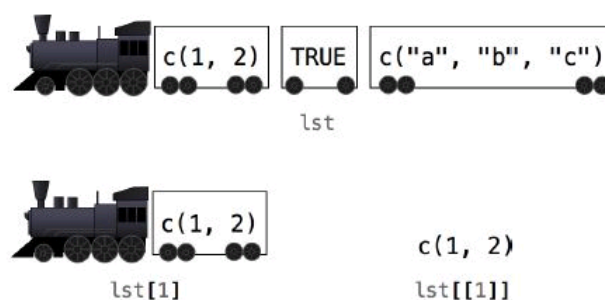


Figura 10: Elementos de una lista

```
lst <- list(c(1, 2), c(TRUE), c("a", "b", "c"))
lst[1] ; lst[[1]]

## [[1]]
## [1] 1 2
## [1] 1 2
```

En una lista con nombre podemos utilizar el simbolo \$ para extraer determinados elementos:

```
lst <- list(v=1:3, w=c("Source", "Stat", "Lab"), x=c(TRUE, FALSE), y=c(1.3, 4.5))
lst$w

## [1] "Source" "Stat" "Lab"
```

Utilizar lst\$w resulta equivalente a utilizar lst[[2]]

```
lst[[2]]

## [1] "Source" "Stat"   "Lab"
```

### 4.1.3. Atributos

Los objetos en R poseen varios atributos tales como: names, class, dim, etc. La función attributes(x) muestra los atributos de un objeto x.

#### Vector atómico:

```
# Vector sin el atributo names
vec <- c(3, 6, -1, 0.5)
attributes(vec)

## NULL

# Vector con el atributo names
vec <- c(a=3, b=6, c=-1, d=0.5)
attributes(vec)

## $names
## [1] "a" "b" "c" "d"
```

#### Lista:

```
# Lista sin el atributo names
lst <- list(1:3, c("Science", "&", "Apps"), c(TRUE, FALSE))
attributes(lst)

## NULL

# Lista con el atributo names
lst <- list(nomb1=1:3, nomb2=c("Source", "Stat", "Lab"), nomb3=c(TRUE,FALSE))
attributes(lst)

## $names
## [1] "nomb1" "nomb2" "nomb3"
```

### 4.1.4. Matrices

Una matriz es un vector con el atributo dimensión dim. El atributo dim es un vector de longitud 2: c(nrow, ncol).

```
mtx <- matrix (1:12,nrow=3, ncol=4, byrow=FALSE)
# se construye por columnas por default (byrow=FALSE)
mtx

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

attributes(mtx)

## $dim
## [1] 3 4
```

dim() se usa para añadir el atributo dimensión a un vector, o para hallar la dimensión de una matriz.

```
mtx <- 1:12
mtx

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12

dim(mtx) <- c(3,4)
mtx

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

mtx <- matrix (1:12, nrow=3, ncol=4, byrow=FALSE)
dim(mtx)

## [1] 3 4
```

### Elementos de una matriz:

```
(mtx <- matrix (1:12, nrow=3, ncol=4, byrow=FALSE))

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

mtx[1,2] # componente 1, 2

## [1] 4
```

```
mtx[,3] # columna 3

## [1] 7 8 9

mtx[1,] # fila 1

## [1] 1 4 7 10

mtx[,c(2,4)] # columnas 2 y 4

##      [,1] [,2]
## [1,]    4   10
## [2,]    5   11
## [3,]    6   12

mtx[c(1,3),] # filas 1 y 3

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    3    6    9   12

mtx[,c(FALSE, TRUE, TRUE, FALSE)] # columnas 2 y 4

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

mtx[c(TRUE, FALSE, TRUE),] # filas 1 y 3

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    3    6    9   12
```

#### 4.1.5. Factores

Es la estructura de datos utilizada para almacenar variables categóricas. ¿Por qué utilizar factores?

- Un vector `c(Femenino, Masculino)` (factor) presenta mayor información que un vector `c(1, 2)`.
- Un vector `character` `c("Femenino", "Masculino")` no pueden ser incluido en modelos de regresión, un factor `c(Femenino, Masculino)` si.



**Figura 11:** Ejemplo de factor en R

Un factor se crea mediante la función:

```
factor()
```

Si se dispone de un vector integer:

```
vec <- c(1, 2, 2, 1, 2, 1, 2)
vec

## [1] 1 2 2 1 2 1 2
```

La función factor asigna labels a los levels (o categorías) de la variable. Los levels del vector vec son los valores 1, 2.

```
# Creación de un factor
fac <- factor(vec, levels=c(1,2), labels = c("Femenino", "Masculino"))
fac

## [1] Femenino Masculino Masculino Femenino Masculino Femenino Masculino
## Levels: Femenino Masculino
```

Si se dispone de un vector character:

```
# Creación de un factor
vec <- c("Femenino", "Masculino", "Masculino", "Femenino", "Masculino",
        "Femenino", "Masculino")
vec

## [1] "Femenino" "Masculino" "Masculino" "Femenino" "Masculino" "Femenino"
## [7] "Masculino"
```

La función factor asigna labels a los levels (o categorías) de la variable. En este caso los levels del vector vec son los valores "Femenino", "Masculino".

```
# Creación de un factor
# por defecto labels = levels
fac <- factor(vec, levels= c("Femenino", "Masculino"), labels=c("FEM", "MASC"))
fac

## [1] FEM MASC MASC FEM MASC FEM MASC
## Levels: FEM MASC
```

Para obtener los atributos de un factor utilizamos `attributes()`.

```
vec <- c(1, 2, 2, 1, 2, 1, 2)
fac <- factor(vec, levels=c(1,2), labels = c("Femenino", "Masculino"))
# Atributos de un factor
attributes(fac)

## $levels
## [1] "Femenino" "Masculino"
##
## $class
## [1] "factor"
```

Para considerar el vector entero utilizamos `unclass()`.

```
# eliminación del atributo class
unclass(fac)

## [1] 1 2 2 1 2 1 2
## attr("levels")
## [1] "Femenino" "Masculino"
```

Para realizar conteos por categoría, se utiliza la función `table()`.

```
fac <- factor(vec, levels=c(1,2), labels = c("Femenino", "Masculino"))
# frecuencias
table(fac)

## fac
## Femenino Masculino
##      3      4

# porcentaje
prop.table(table(fac))

## fac
## Femenino Masculino
##  0.43    0.57
```

#### 4.1.6. Data Frame

Es una lista en la cual todos los elementos tienen la misma longitud. A diferencia de las matrices, pueden almacenar vectores atómicos de cualquier tipo. Presenta varios atributos adicionales `class`, `rownames`, `names`. Es la estructura de datos más utilizada para almacenar data tabulada.

data frame

1	"R"	TRUE
2	"S"	FALSE
3	"T"	TRUE

numeric    character    logical

**Figura 12:** Ejemplo de factor en R

Para crear un data frame se utiliza la función `data.frame()`. Con los siguientes vectores atómicos:

```
dbl_vec <- c(1, 2, 3)
chr_vec <- c("R", "S", "T")
log_vec <- c(TRUE, FALSE, TRUE)
```

Creamos el data frame df:

```
df <- data.frame(dbl_vec, chr_vec, log_vec)
df

##   dbl_vec chr_vec log_vec
## 1      1      R    TRUE
## 2      2      S   FALSE
## 3      3      T    TRUE
```

Un data frame es una lista:

```
typeof(df) # Tipo de un data frame

## [1] "list"
```

Su clase es `data.frame`:

```
class(df)

## [1] "data.frame"
```

Atributos de un data frame:

```
attributes(df)

## $names
## [1] "dbl_vec" "chr_vec" "log_vec"
```



```
##  
## $row.names  
## [1] 1 2 3  
##  
## $class  
## [1] "data.frame"
```

Nombres de las columnas de df:

```
names(df)  
  
## [1] "dbl_vec" "chr_vec" "log_vec"  
  
colnames(df)  
  
## [1] "dbl_vec" "chr_vec" "log_vec"
```

Nombres de las filas de df:

```
rownames(df)  
  
## [1] "1" "2" "3"
```

Dimensión: Filas y columnas

```
dim(df)  
  
## [1] 3 3
```

Número de filas y columnas

```
nrow(df)  
  
## [1] 3  
  
ncol(df)  
  
## [1] 3
```

**Elementos de un data frame:**

Mediante `df[i, j]` se obtiene la componente *i, j* del data frame.

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

`df[2, c(2,3)]`

**Figura 13:** Extracción de elementos de un Data Frame

```
nomb <- c("John", "Paul", "George", "Ringo")
nac <- c(1940, 1941, 1943, 1940)
instr <- c("guitar", "bass", "guitar", "drums")
```

```
df <- data.frame(nomb, nac, instr)
df[2, c(2,3)]
```

```
##      nac instr
## 2 1941  bass
```

```
print(df)
```

```
##      nomb  nac  instr
## 1   John 1940  guitar
## 2   Paul 1941   bass
## 3 George 1943  guitar
## 4  Ringo 1940  drums
```

```
df[2, 2] # componente 2, 2
```

```
## [1] 1941
```

```
df[3, 1] # componente 3, 1
```

```
## [1] George
## Levels: George John Paul Ringo
```

```
df[3, ] # fila 3
```

```
##      nomb  nac  instr
## 3 George 1943  guitar
```

```
df[c(1, 4), ] # filas 1, 4
```

```
##      nomb  nac  instr
## 1   John 1940  guitar
## 4  Ringo 1940  drums
```

Se pueden seleccionar ciertas filas mediante TRUE y FALSE.

```
df[c(TRUE, TRUE, FALSE, TRUE), ] # equivalente a df[c(1, 2, 4), ]

##      nomb  nac instr
## 1   John 1940 guitar
## 2   Paul 1941  bass
## 4  Ringo 1940  drums
```

Importante: Reciclado:

```
df[c(TRUE, FALSE), ] # equivalente a df[c(1, 3), ]

##      nomb  nac instr
## 1   John 1940 guitar
## 3 George 1943 guitar
```

Importante: Filtrado o subsetting:

```
df[, 3]=="guitar" # columna 3 de df igual a "guitar"

## [1]  TRUE FALSE  TRUE FALSE

f_guitar <- df[, 3]=="guitar"
```

Filas donde la columna 3 es igual a "guitar"

```
df[f_guitar, ]

##      nomb  nac instr
## 1   John 1940 guitar
## 3 George 1943 guitar

df[, 2] # columna 2

## [1] 1940 1941 1943 1940

df[, c(1, 3)] # columnas 1, 3

##      nomb  instr
## 1   John guitar
## 2   Paul  bass
## 3 George guitar
## 4  Ringo  drums
```

Columna de nombre "nac"

```
df[, "nac"] # equivalente a df[, 2]
```

```
## [1] 1940 1941 1943 1940
```

Columnas de nombres "nomb y nac"

```
df[, c("nomb", "nac")] # equivalente a df[, c(1, 2)]
```

```
##      nomb  nac
## 1   John 1940
## 2   Paul 1941
## 3 George 1943
## 4  Ringo 1940
```

Se pueden seleccionar ciertas columnas mediante TRUE y FALSE:

```
df[, c(TRUE, FALSE, TRUE)] # equivalente a df[, c(1, 3)]
```

```
##      nomb instr
## 1   John guitar
## 2   Paul  bass
## 3 George guitar
## 4  Ringo drums
```

Importante: Reciclado

```
df[, c(TRUE, FALSE)] # equivalente a df[, c(1, 3)]
```

```
##      nomb instr
## 1   John guitar
## 2   Paul  bass
## 3 George guitar
## 4  Ringo drums
```

## 4.2. R Data Frames

R posee varios data frames en sus bases de datos internas, por ejemplo: mtcars

```
data(mtcars)
```

Visualización:

```
View(mtcars)
```

```
# n primeras filas
head(mtcars,n = 2)

##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4      21   6  160 110  3.9 2.6   16  0  1    4    4
## Mazda RX4 Wag  21   6  160 110  3.9 2.9   17  0  1    4    4

# n últimas filas
tail(mtcars,n = 2)

##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Maserati Bora  15   8  301 335  3.5 3.6   15  0  1    5    8
## Volvo 142E     21   4  121 109  4.1 2.8   19  1  1    4    2
```

Atributos del data frame:

```
attributes(mtcars)

## $names
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
##
## $row.names
## [1] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710"
## [4] "Hornet 4 Drive" "Hornet Sportabout" "Valiant"
## [7] "Duster 360" "Merc 240D" "Merc 230"
## [10] "Merc 280" "Merc 280C" "Merc 450SE"
## [13] "Merc 450SL" "Merc 450SLC" "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial" "Fiat 128"
## [19] "Honda Civic" "Toyota Corolla" "Toyota Corona"
## [22] "Dodge Challenger" "AMC Javelin" "Camaro Z28"
## [25] "Pontiac Firebird" "Fiat X1-9" "Porsche 914-2"
## [28] "Lotus Europa" "Ford Pantera L" "Ferrari Dino"
## [31] "Maserati Bora" "Volvo 142E"
##
## $class
## [1] "data.frame"
```

Nombres de las columnas:

```
names(mtcars)

## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"

colnames(mtcars)

## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
```

Nombres de las filas

```
rownames(mtcars)
```

```
## [1] "Mazda RX4"          "Mazda RX4 Wag"      "Datsun 710"
## [4] "Hornet 4 Drive"     "Hornet Sportabout"  "Valiant"
## [7] "Duster 360"        "Merc 240D"          "Merc 230"
## [10] "Merc 280"          "Merc 280C"          "Merc 450SE"
## [13] "Merc 450SL"        "Merc 450SLC"        "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"  "Fiat 128"
## [19] "Honda Civic"        "Toyota Corolla"     "Toyota Corona"
## [22] "Dodge Challenger"   "AMC Javelin"        "Camaro Z28"
## [25] "Pontiac Firebird"   "Fiat X1-9"          "Porsche 914-2"
## [28] "Lotus Europa"       "Ford Pantera L"     "Ferrari Dino"
## [31] "Maserati Bora"      "Volvo 142E"
```

Dimensiones en un data frame:

```
# Dimensión
```

```
dim(mtcars)
```

```
## [1] 32 11
```

```
# Número columnas
```

```
ncol(mtcars)
```

```
## [1] 11
```

```
# Número filas
```

```
nrow(mtcars)
```

```
## [1] 32
```

### 4.3. Función structure

La función `str()` (structure) presenta una descripción compacta de la estructura de datos. Usar `View()` es costoso computacionalmente.

```
str(mtcars)
```

```
## 'data.frame': 32 obs. of 11 variables:
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num 160 160 108 258 360 ...
## $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
```

```
## $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num 16.5 17 18.6 19.4 17 ...
## $ vs : num 0 0 1 1 0 1 0 1 1 ...
## $ am : num 1 1 1 0 0 0 0 0 0 ...
## $ gear: num 4 4 4 3 3 3 3 4 4 ...
## $ carb: num 4 4 1 1 2 1 4 2 2 ...
```

## 4.4. Missing Values

Los valores perdidos en R se denotan por NA, NaN. NA Not Available. Dato perdido.

```
x <- c(NA, 3.2, NA, 5, NA)
x

## [1] NA 3.2 NA 5.0 NA
```

NaN Not a Number. Resultado de una indeterminación:

```
y <- c(-1, 0/0, 0.8, 5, Inf*0)
y

## [1] -1.0 NaN 0.8 5.0 NaN
```

Inf representa infinito.

is.na(x) retorna TRUE para elementos NA de un vector atómico o lista.

```
x <- c(0, 3.2, NA, 5, NA)
is.na(x)

## [1] FALSE FALSE TRUE FALSE TRUE
```

is.nan(x) retorna TRUE para elementos NaN de un vector atómico.

```
y <- c(-1, 0/0, 5, Inf*0)
is.nan(y)

## [1] FALSE TRUE FALSE TRUE
```

## 5. Importación y Exportación de datos

### 5.1. Lectura de datos

R puede acceder a información almacenada en distintos formatos:

- Archivos de excel .xls, .xlsx, .csv;
- Archivos de texto plano .txt;
- Archivos de spss .sav;
- Archivos de la web;
- Archivos de bases de datos, etc.



Figura 14: Varios formatos que lee el R

## 5.2. Directorio de trabajo

Working directory (wd). Es la dirección donde se almacenan, leen y escriben los archivos utilizados y generados mediante R. `getwd()` permite obtener el wd actual:

```
getwd()

## [1] "C:/Users/Andres/Desktop/Manual_R_RUGE"
```

`setwd()` permite setear un nuevo wd, por ejemplo:

```
setwd("C:/Documentos/Curso_CEC_EPN")
```

`list.files()` enlista los nombres de los archivos en el wd actual. El parámetro `pattern` se usa para enlistar los nombres de los archivos que contienen un determinado patrón.

```
list.files(pattern = ".R")

## [1] "autoresR.png"           "coercionR.png"
## [3] "combinarR.png"          "dataFrameR.png"
## [5] "elementosListaR.png"    "elemVectorR.png"
## [7] "elemVectorTF_R.png"     "factorR.png"
## [9] "GURE.png"               "importR.png"
## [11] "installR.png"           "installRStudio.png"
```



```
## [13] "leeArchivosR.png"          "listaR.png"
## [15] "Manual Curso R_CEC_EPN_M1.pdf" "Manual_CEC.Rnw"
## [17] "Manual_R_CEC_EPN_M1.pdf"    "Manual_R_CEC_tapas-1-2.pdf"
## [19] "objetosR.png"              "operacionesVectorR.png"
```

`file.exists()` verifica si existe una carpeta con un determinado nombre en el wd.

```
# Verifica si existe o no una carpeta con nombre "ssl"
file.exists("capacitacion")

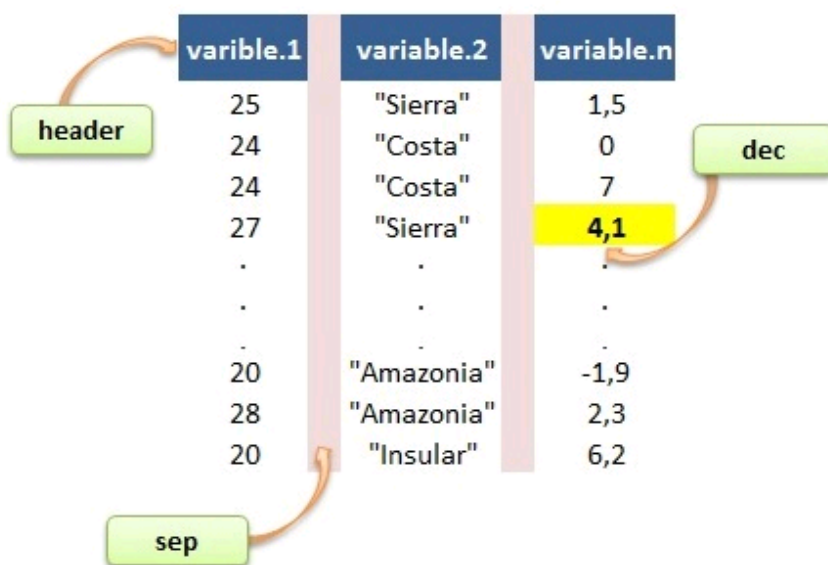
## [1] FALSE
```

`dir.create()` permite crear nuevas carpetas en el wd.

```
# Crea una nueva carpeta con nombre "ssl"
dir.create("ssl")
```

R sobrescribe una carpeta sobre otra, de ahí la importancia de utilizar previamente `file.exists()`.

### 5.3. ¿Cómo leer archivos en R?



	variable.1	variable.2	variable.n
	25	"Sierra"	1,5
	24	"Costa"	0
	24	"Costa"	7
	27	"Sierra"	4,1
	.	.	.
	.	.	.
	.	.	.
	20	"Amazonia"	-1,9
	28	"Amazonia"	2,3
	20	"Insular"	6,2

Figura 15: Cómo lee los archivos el R

#### 5.3.1. Lectura de archivos .txt

Se requiere leer un archivo en formato txt: `archivo.txt`. Se utiliza la función `read.table()`

```
data_txt <- read.table(file = "archivo.txt", sep = "\t", dec = ",", header = TRUE)
str(data_txt)
```

Parámetros:

- file: nombre del archivo (incluida extensión);
- sep: caracter utilizado para separar columnas (variables);
- dec: caracter utilizado para decimales;
- header: TRUE, si la primera fila contiene los nombres de las columnas.

str() describe la estructura de datos.

Por defecto R coerciona las variables categóricas a factor. El parametro stringsAsFactors=FALSE evita esta coerción.

```
data_txt <- read.table(file = "archivo.txt", sep = "\t", dec = ",",
                      header = TRUE, stringsAsFactors = FALSE)
```

### 5.3.2. Lectura de archivos .csv

Se requiere leer un archivo en formato csv: archivo.csv, se utiliza la función read.csv que presenta por defecto los argumentos: sep = ",", dec = ".", , header = TRUE

```
data_csv <- read.csv(file = "archivo.csv")
```

read.csv2(): Se debe especificar los argumentos: sep, dec, header

```
data_csv2 <- read.csv2(file = "archivo.csv", sep = ",", dec = ".", header = TRUE)
```

### 5.3.3. Lectura de archivos .xls, .xlsx

Se requiere leer un archivo en formato xls o xlsx: archivo.xlsx. Se utiliza la función read\_excel() del paquete readxl

```
install.packages("readxl", dependencies = TRUE)
library(readxl)
ls("package:readxl")
```

Lectura del archivo archivo.xlsx:

```
data_xlsx <- read_excel("archivo.xlsx", sheet = "datos", col_names = TRUE, na="")
read.table("clipboard", sep = "\t", header = FALSE)
```

Parámetros:

- sheet: Nombre de la hoja que contiene la data (recibe también el número de hoja);
- col\_names: TRUE si la primera fila contiene los nombres de las columnas;
- na: los caracter que se coercion a NA.

#### 5.3.4. Lectura de archivos .sav

Se requiere leer archivos desde spss, es decir en formato .sav: archivo.sav. Se utiliza la función read.spss() del paquete foreign.

```
install.packages("foreign", dependencies = TRUE)
library(foreign)
ls("package:foreign")

data_sav <- read.spss(file="archivo.sav", use.value.labels = TRUE,
                      to.data.frame = TRUE)
```

Parámetros:

- file: nombre del archivo (incluida extensión);
- use.value.labels: TRUE, si se consideran las etiquetas de las variables;
- to.data.frame: TRUE, para coercionar el archivo leído a data frame.

## 6. Estructuras de control y funciones:

### 6.1. Introducción

Una función es un conjunto de instrucciones, que a partir de uno o más datos de entrada, permiten generar y retornar un determinado resultado.

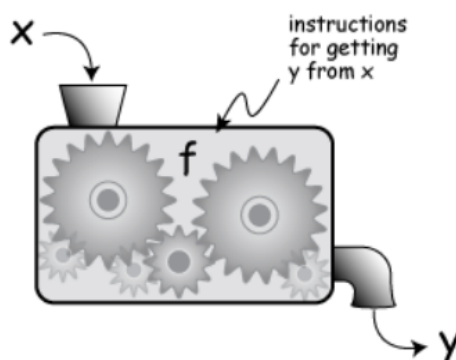


Figura 16: Función

## 6.2. Porqué construir funciones?

**Practicidad:** Permiten ejecutar un conjunto de instrucciones en una sola línea de código.

Ejemplo: La función `sort`, las instrucciones necesarias para ordenar un vector.

```
x <- c(3, 1, 6, 2)
x

## [1] 3 1 6 2
```

```
sort(x)

## [1] 1 2 3 6
```

**Generalización:** Permiten diseñar códigos para casos determinados y ajustarlos a casos generales.

Ejemplo: Hallar la media de 3 y 4

```
(3 + 4) / 2

## [1] 3.5
```

Para hallar el promedio de varios pares de números, creamos la función `promedio`.

```
promedio <- function(a, b) {
  prom <- (a + b) / 2
  return(prom)
}
```

## 6.3. Estructura de una función

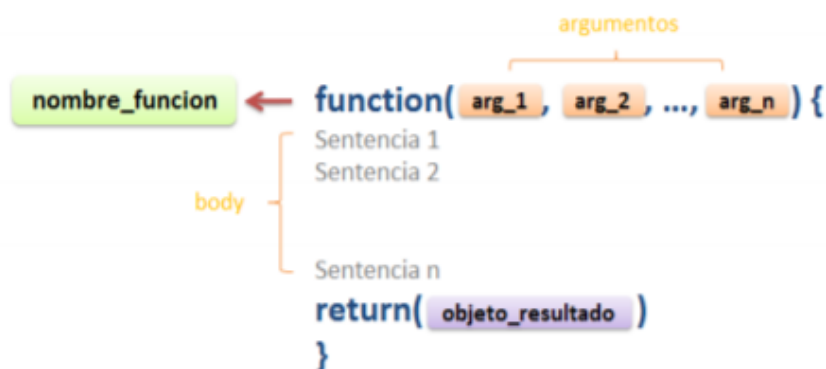


Figura 17: Estructura de una Función

El comando `function()` tiene como tarea crear funciones a partir de ciertos argumentos que son proporcionados por el usuario.

## 6.4. Argumentos

No existe un límite en la cantidad de argumentos que una función puede recibir.

```
promedio_pond <- function(a, b, peso1, peso2) {  
  prom <- peso1*a + peso2*b  
  return (prom)  
}
```

Ejemplos:

```
promedio_pond(a = 1, b = 5, peso1 = 0.8, peso2 = 0.2)  
  
## [1] 1.8
```

```
promedio_pond(a = 5, b = 1, peso1 = 0.8, peso2 = 0.2)  
  
## [1] 4.2
```

## 6.5. Argumentos por default

Existen ocasiones que no se requieren ingresar ciertos argumentos para que la función pueda ejecutarse.

```
promedio_pond <- function(a, b, peso1, peso2) {  
  prom <- peso1*a + peso2*b  
  return (prom)  
}
```

```
promedio_pond(a = 1, b = 5)
```

```
## Error in promedio_pond(a = 1, b = 5): el argumento "peso1" está ausente, sin valor  
por omisión
```

Estos valores por default deben colocarse en la función:

```
promedio_pond <- function(a, b, peso1 = 0.5, peso2 = 0.5) {  
  prom <- peso1*a + peso2*b  
  return (prom)  
}
```

```
promedio_pond(a = 1, b = 5)
```

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

## 6.6. Lexical Scoping

**Variable libre:** Una variable en una función se dice libre si no ha sido definida o no es un argumento.

El lexical Scoping permite entender cómo *R* asocia un valor a una variable libre (crear un objeto).

```
f <- function(x, y) {  
  return(x^2 + y / z)  
}
```

**Argumentos:** x, y. **Variables libres:** z.

```
f(x = 3, y = 6)
```

```
## [1] Inf  15  15 Inf  15
```

Considere la siguiente función:

```
z <- 10  
f <- function(x) {  
  z <- 2  
  z^2 + g(x)  
}  
g <- function(x) {  
  x*z  
}
```

Cuál es el valor de  $f(x = 3)$  ?

```
f(x = 3)
```

```
## [1] 34
```

### Ejemplos: Funciones

1. Escriba una función que reciba un vector numérico y un valor de reemplazo (por default 0) para recodificar los datos perdidos de dicho vector.
2. Escriba una función que reciba un vector y calcule el porcentaje de datos perdidos.
3. Escriba una función que permita conocer si una variable numérica es constante.
4. Escriba una función que permita conocer si el número de categorías o niveles en una variable cualitativa exceden a un valor `n.levels` dado.

## 6.7. Estructuras de Control

Son herramientas que permiten manejar de una forma mucho más estructurada el flujo de ejecución de un código. Las estructuras de control se clasifican en:

- Condicionales: if, else (e ifelse).
- Loops o bucles: for, while.

### 6.7.1. Sentencia if

La sentencia if realiza una tarea para un determinado caso.

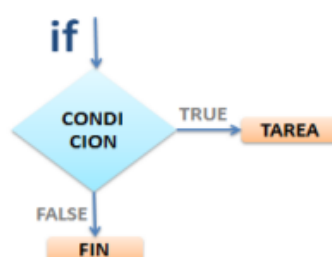


Figura 18: Sentencia If

CONDICION: Vector lógico 1d **TRUE/FALSE**. Si condicion es **TRUE** se ejecuta la tarea.

Forma de declarar la sentencia if:

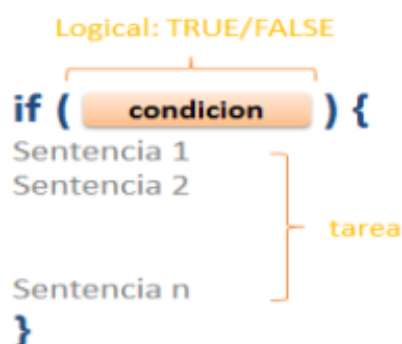


Figura 19: Declaración Sentencia If

```

x <- 2
if(x > 0) {
  x <- x + 1
}
x

## [1] 3
  
```

### 6.7.2. Sentencia else

La sentencia else trabaja conjuntamente con la sentencia if

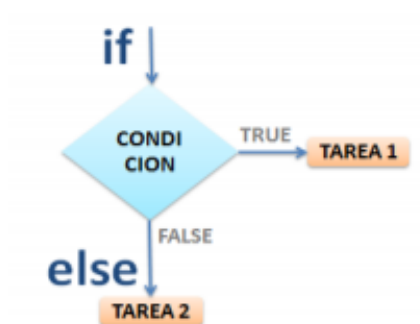


Figura 20: Sentencia else

Si la condición es **TRUE** se ejecuta la Tarea 1, caso contrario se ejecuta la Tarea 2.

### 6.7.3. Sentencia if else

Forma de declarar la sentencia **if else**:

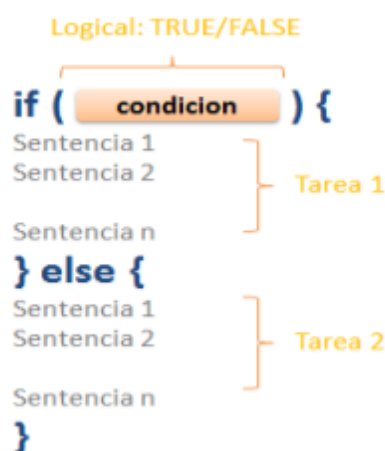


Figura 21: Sentencia if else

Ejemplo:

```
x <- -1
if (x >= 0) {
  print("El número es positivo")
} else {
  print("El número es negativo")
}

## [1] "El número es negativo"
```



#### 6.7.4. Sentencia for

La sentencia for permite repetir una acción un número determinado de veces.

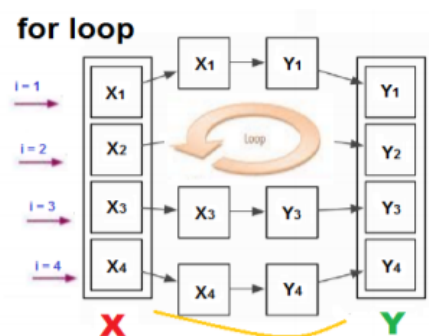


Figura 22: Sentencia for

Forma de declarar la sentencia for:

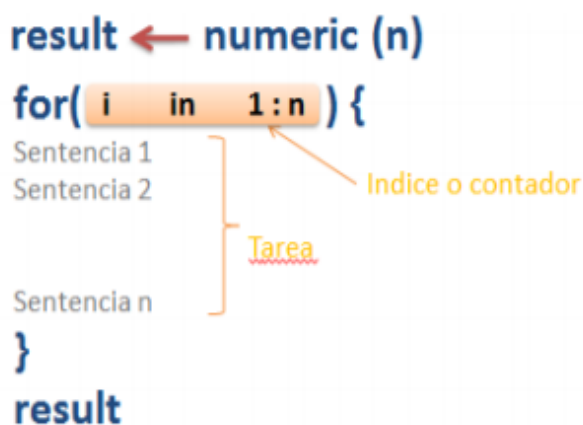


Figura 23: Declaración Sentencia for

Ejemplo: Suma de filas/columnas: m: matriz de 4x3 de Poisson

```
m <- matrix(data = rpois(n = 12, lambda = 8), nrow = 4, ncol = 3)
```

```
m
```

```
##      [,1] [,2] [,3]
## [1,]    7    4    5
## [2,]    9    9    9
## [3,]   13    6    7
## [4,]    9    8    7
```

```
col_sumas <- numeric(ncol(m))
```

```
for (i in 1:4){
```

```
col_sumas[i] <- sum(m[i,])
```

```
}
```

```
col_sumas
```

## [1] 16 27 26 24

### 6.7.5. Ejercicios sobre Estructuras de Control

1. Escriba un algoritmo que permita evaluar la función.

$$f(x) = \begin{cases} x^2 + 2x + 3 & \text{if } x < 0 \\ x + 3 & \text{if } 0 \leq x < 2 \\ x^2 + 4x - 7 & \text{if } 2 \leq x. \end{cases}$$

2. Escriba un algoritmo que almacene la siguiente suma:

$$\sum_{i=10}^{100} (i^3 + 4i^2)$$

3. Escriba un algoritmo que calcule el promedio móvil de orden 2 de  $x = (x_1, x_2, \dots, x_n)$ .  
El promedio móvil de orden 2 es el vector de longitud  $n - 1$  cuya  $i$ -ésima componente es  $\frac{x_i + x_{i+1}}{2}$

## 7. Introducción a gráficas “ggplot”

### 7.1. Sistemas gráficos de R

R actualmente constituye una de las principales herramientas empleada en la generación de gráficos estadísticos de alta calidad y complejidad. Presenta características muy superiores a los clásicos softwares estadísticos generalmente utilizados.

Actualmente R dispone de tres tipos de sistemas (R paquetes) gráficos, entre ellos tenemos:

Base, Lattice, Ggplot2

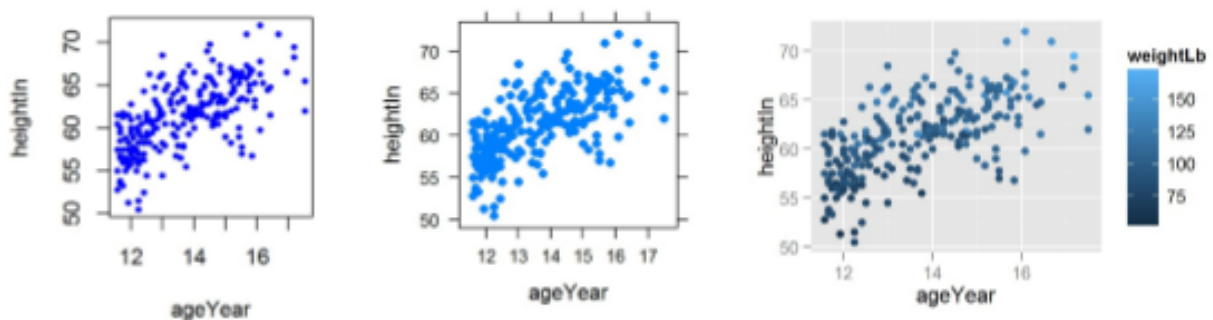


Figura 24: Sistema Gráfico de R

## 7.2. Sistemas Gráficos Base

Se basa en la utilización de los paquetes:

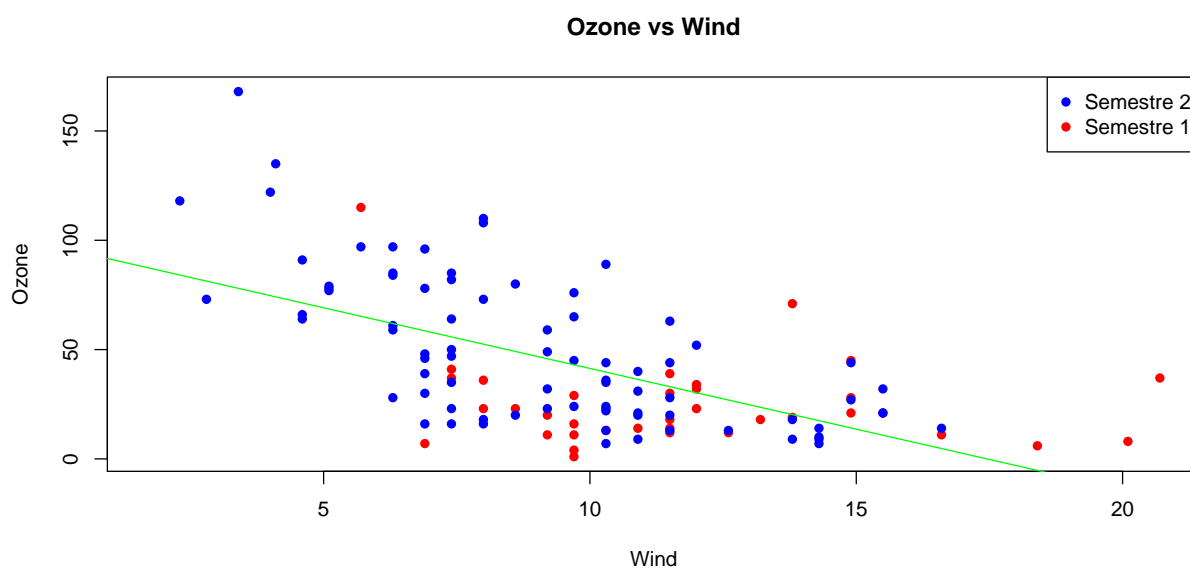
- graphics.Funciones gráficas hist() plot(), etc.
- grDevices.Funciones de dispositivos gráficos pdf() png() ,etc

Características:

- Modelo paleta de artista
- Crear lienzo del gráfico
- Añadir elementos utilizando las funciones lines(), points(), etc
- Muy intuitivo
- No posee un lenguaje gráfico estándar
- Llamando a varias funciones (códigos extensos)
- Una vez empezado el gráfico no es posible volver al inicio
- Permite fijar inicialmente layouts (margenes,espacios,etc) a través de la función par ().

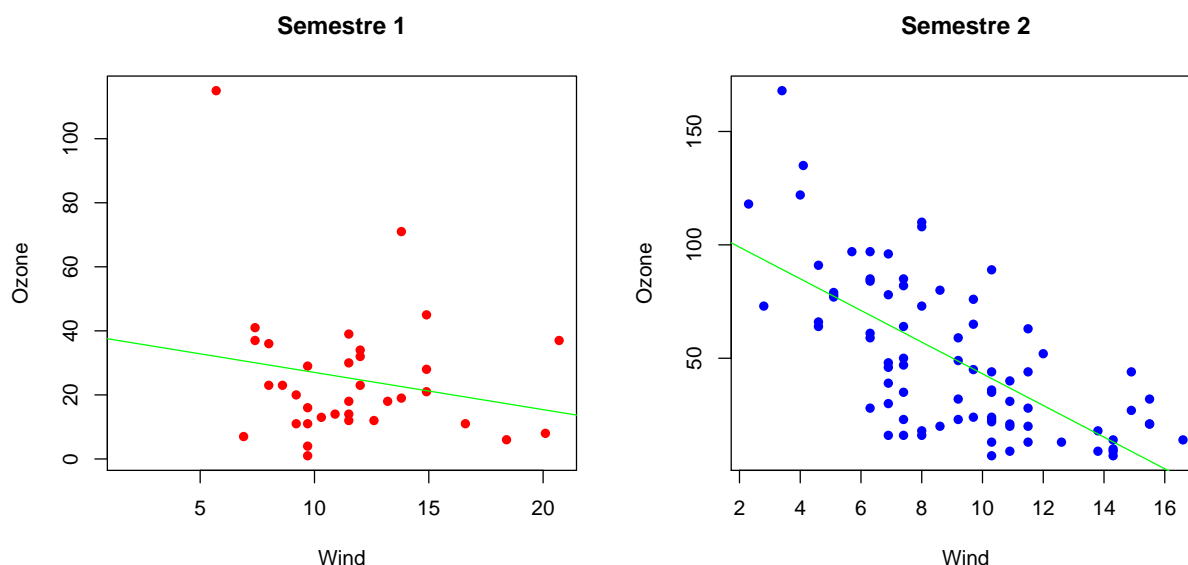
### Ejemplo

```
with(airquality, plot(Wind, Ozone, main = "Ozone vs Wind", type = "n"))  
with(subset(airquality, Month <= 6), points(Wind, Ozone, col = "red", pch=16))  
with(subset(airquality, Month > 6), points(Wind, Ozone, col = "blue", pch=16))  
model <- lm(Ozone ~ Wind, airquality)  
abline(model, lwd = 1, col= "green")  
legend("topright", pch = 16, col = c("blue", "red"),  
      legend = c("Semestre 2", "Semestre 1"))
```



## 7.3. Facetas

```
par(mfrow=c(1,2))
with(subset(airquality, Month <= 6),
     plot(Wind, Ozone, col = "red", pch=16, main="Semestre 1"))
model1 <- lm(Ozone ~ Wind, subset(airquality, Month <= 6))
abline(model1, lwd = 1, col= "green")
with(subset(airquality, Month > 6),
     plot(Wind, Ozone, col = "blue", pch=16, main="Semestre 2"))
model2 <- lm(Ozone ~ Wind, subset(airquality, Month > 6))
abline(model2, lwd = 1, col= "green")
```



## 7.4. Sistema gráfico Lattice

Se basa en la utilización de los paquetes:

- Lattice. Funciones gráficas `xyplot()` `bwplot()`, etc.
- Grid. Para gráficos condicionados (Facetas).

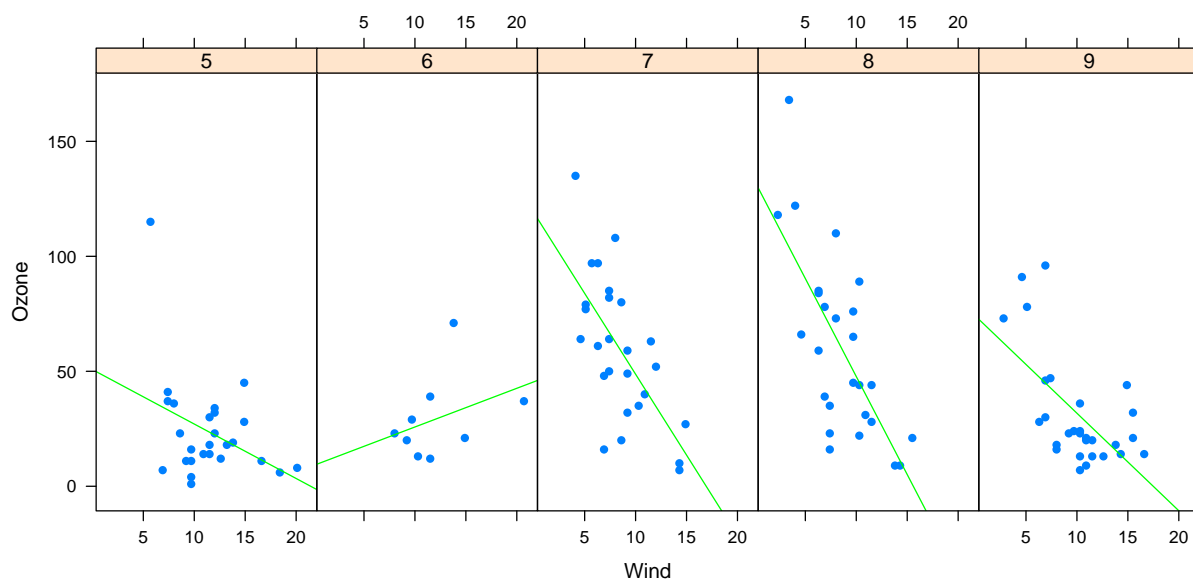
Características:

- No es muy intuitivo
- Llamado de una sola función.
- Layouts (margenes,espacios,etc) se fijan automáticamente

Ejemplo:

```
library(lattice)
airquality <- transform(airquality, Month = as.factor (Month))

graf <- xyplot(Ozone~Wind | Month, data = airquality, layout=c(5,1),
              panel = function(x, y){
                panel.xyplot(x, y, pch=16)
                panel.lmline(x, y, col = "green")})
print(graf)
```



## 7.5. Sistema gráfico ggplot2

Ggplot2 es el sistema de gráficos resultante de la combinación de las características de los sistemas base y lattice. Fue desarrollado por Hardley Wickham como una implementación de la gramática de gráficos (Grammar of Graphics) propuesta por Leland Wilkinson

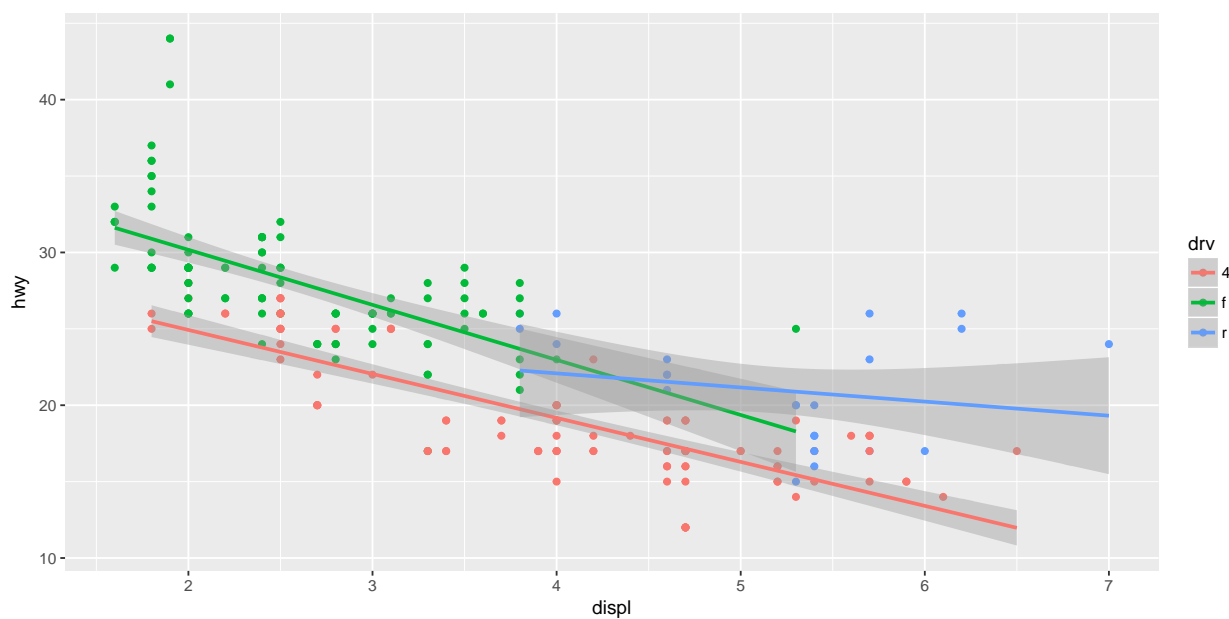
Base + Lattice = ggplot2

ggplot2 se encuentra disponible en el repositorio CRAN, se puede instalar y cargar en el área de trabajo directamente ejecutando las líneas de código siguientes:

```
#install.packages("ggplot2", dependencies=TRUE)
library(ggplot2)
```

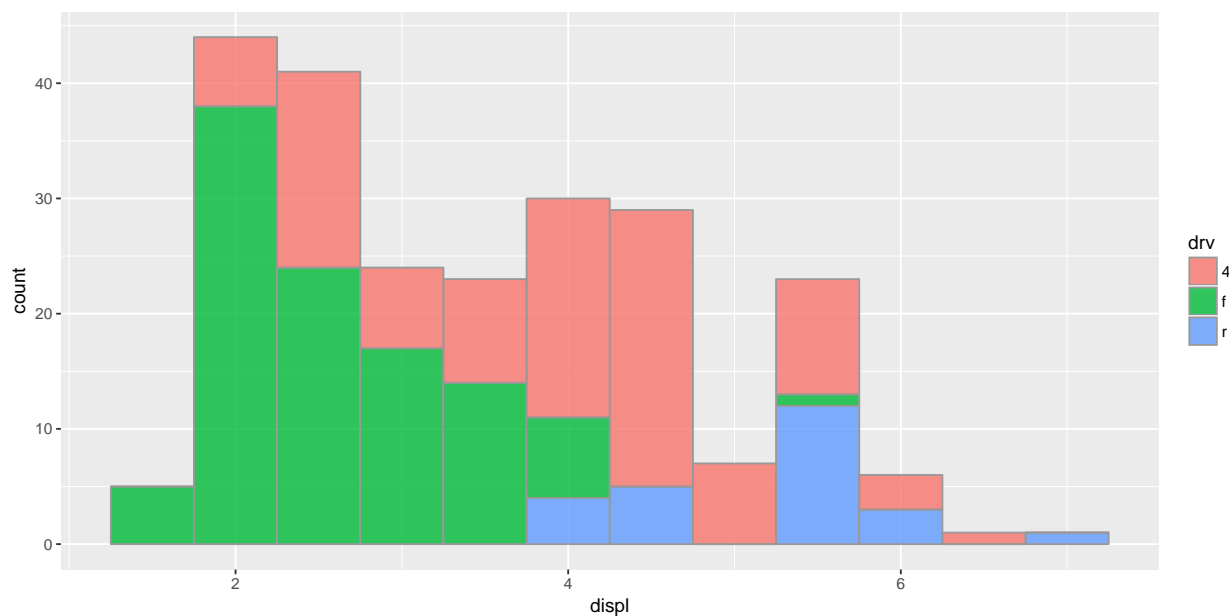
## 7.6. Generación el primer gráfico ggplot2

```
library(ggplot2)
g <- ggplot(mpg, aes(x=displ, y=hwy, color=drv))
g + geom_point() + geom_smooth(method="lm")
```



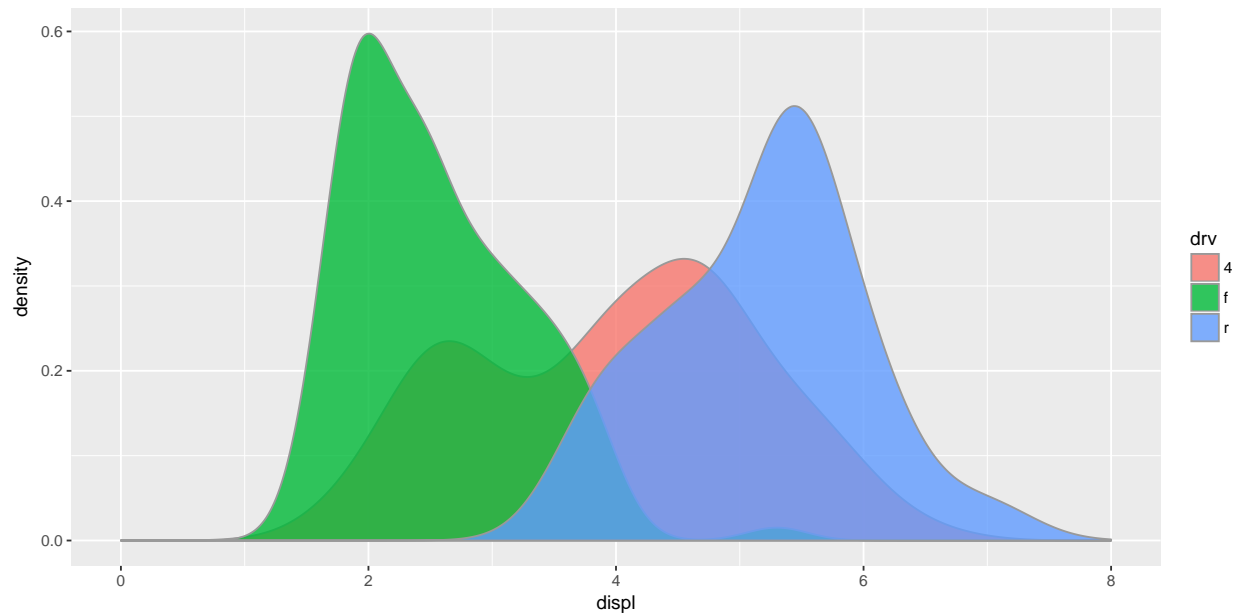
## 7.7. Histogramas

```
library(ggplot2)
g <- ggplot(mpg, aes(x=displ, fill=drv))
g + geom_histogram(binwidth=0.5, alpha = 0.8, colour="gray60")
```



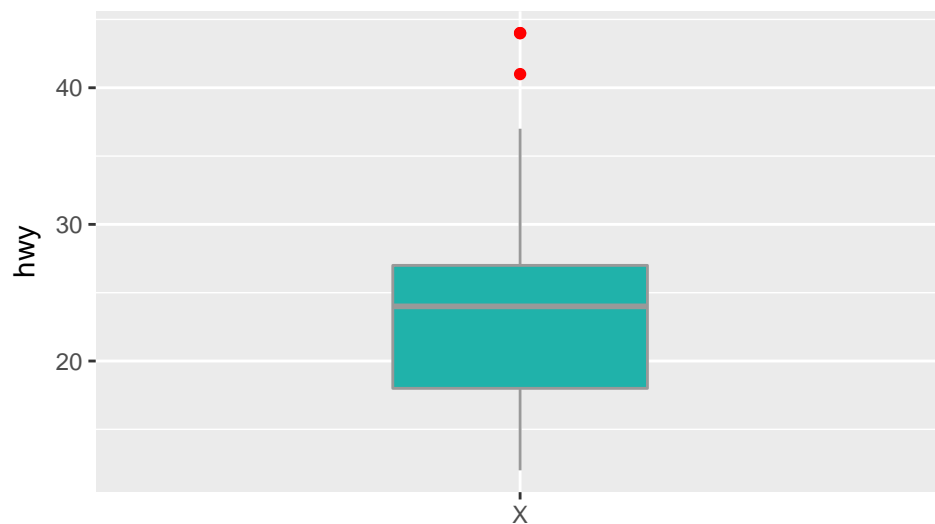
## 7.8. Curvas de densidad

```
library(ggplot2)
g <- ggplot(mpg, aes(x=displ, fill=drv))
g + geom_density(alpha=0.8, colour="gray60") + xlim(c(0, 8))
```



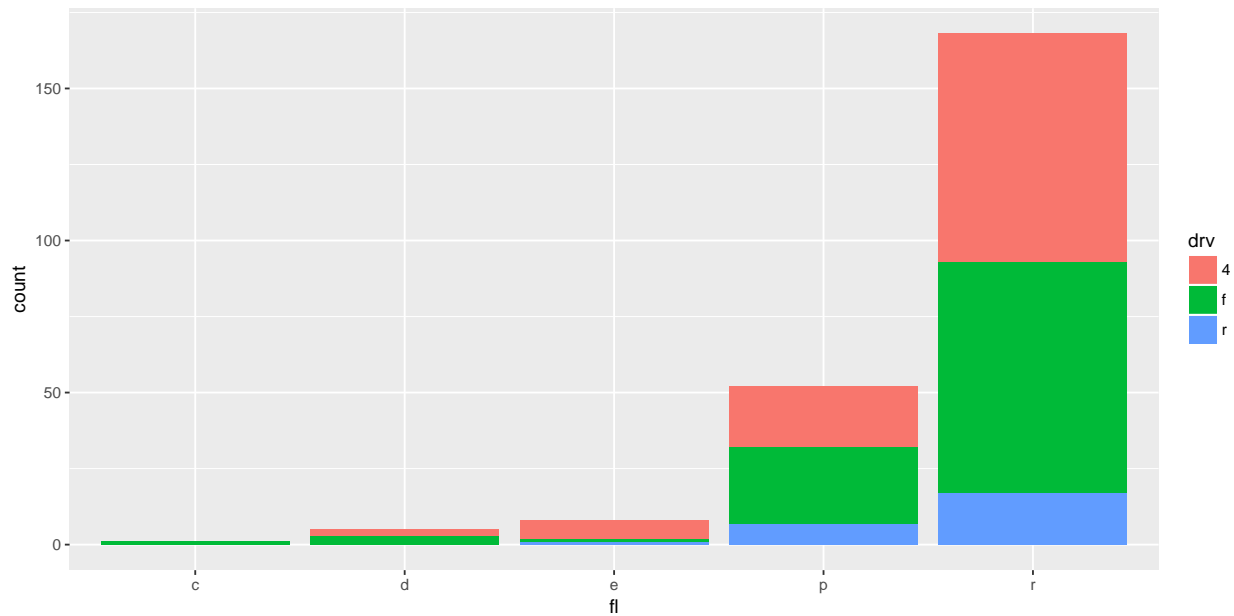
## 7.9. Diagrama de Caja bigotes

```
library(ggplot2)
g <- ggplot(mpg, aes(x="X", y=hwy))
g + geom_boxplot(width=0.3, fill="lightseagreen", color="gray60",
  outlier.colour = "red") + labs(x="")
```



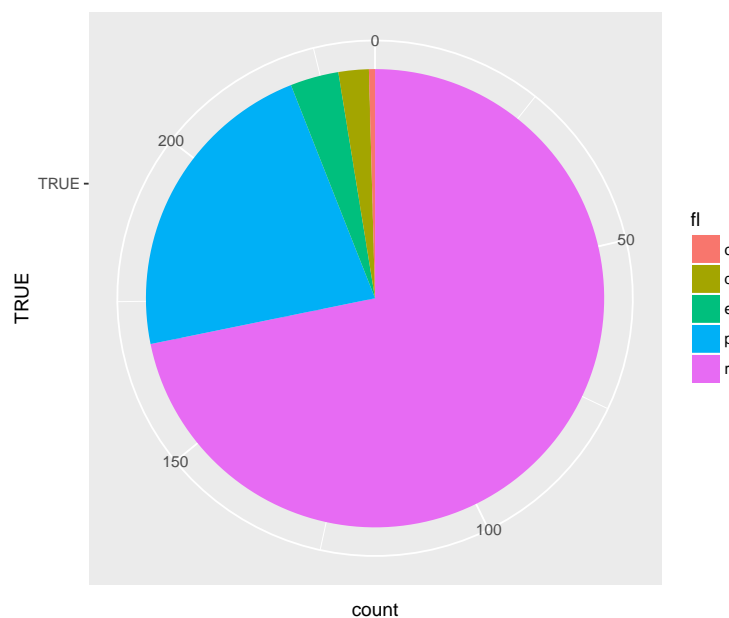
## 7.10. Diagrama de Barras

```
library(ggplot2)
g <- ggplot(mpg, aes(x=fl, fill=drv))
g + geom_bar()
```



## 7.11. Diagrama de pie

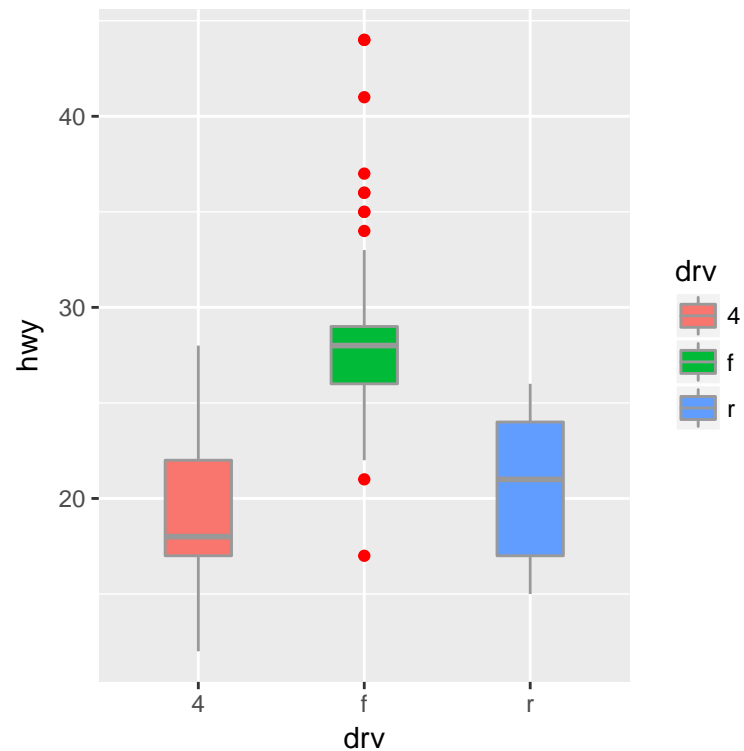
```
library(ggplot2)
g <- ggplot(mpg, aes(x=TRUE, fill=fl)) + geom_bar(width=1)
g + coord_polar(theta = "y")
```



## 7.12. Gráficos múltiples de distribución

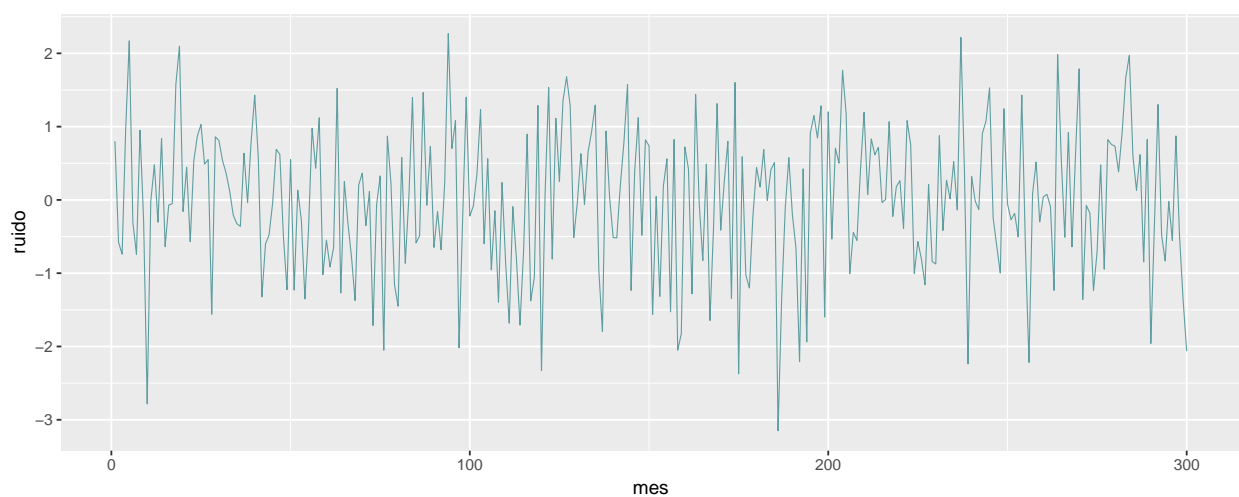
```
library(ggplot2)
g <- ggplot(mpg, aes(x=drv, y=hwy, fill=drv))
g + geom_boxplot(width=0.4, colour="gray60", outlier.colour = "red")
```





### 7.13. Gráfico de linea básico

```
library(ggplot2)
mes <- 1:300; ruido <- rnorm(300,0,1)
d <- data.frame(mes,ruido)
g <- ggplot(d, aes(x=mes, y=ruido))
g + geom_line(colour="cadetblue", size=0.3)
```



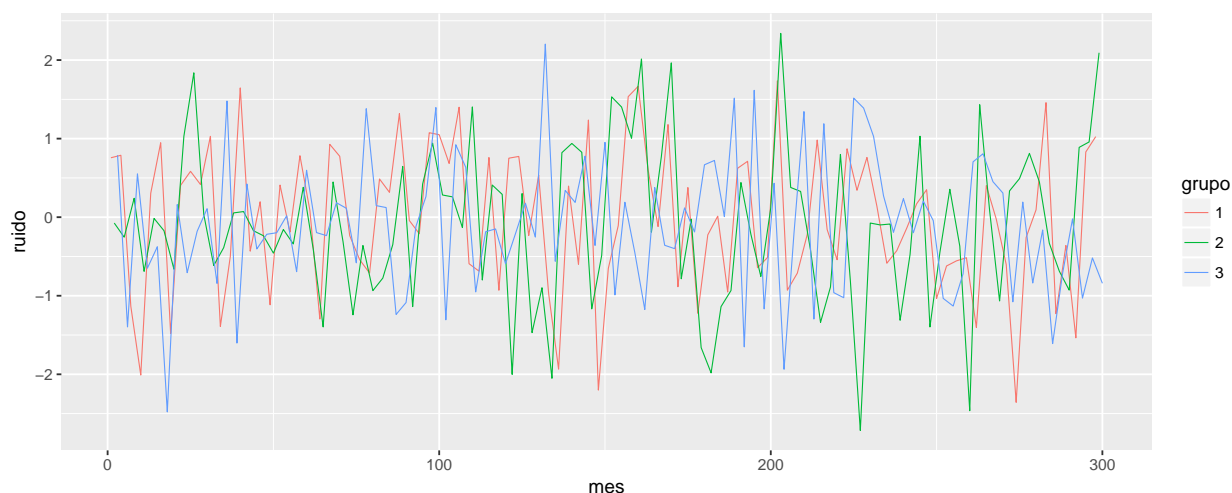
### 7.14. Gráfico de líneas múltiples

```
library(ggplot2)
mes <- 1:300; ruido <- rnorm(300,0,1); grupo <- factor(rep(1:3,100))

d <- data.frame(mes,ruido,grupo)

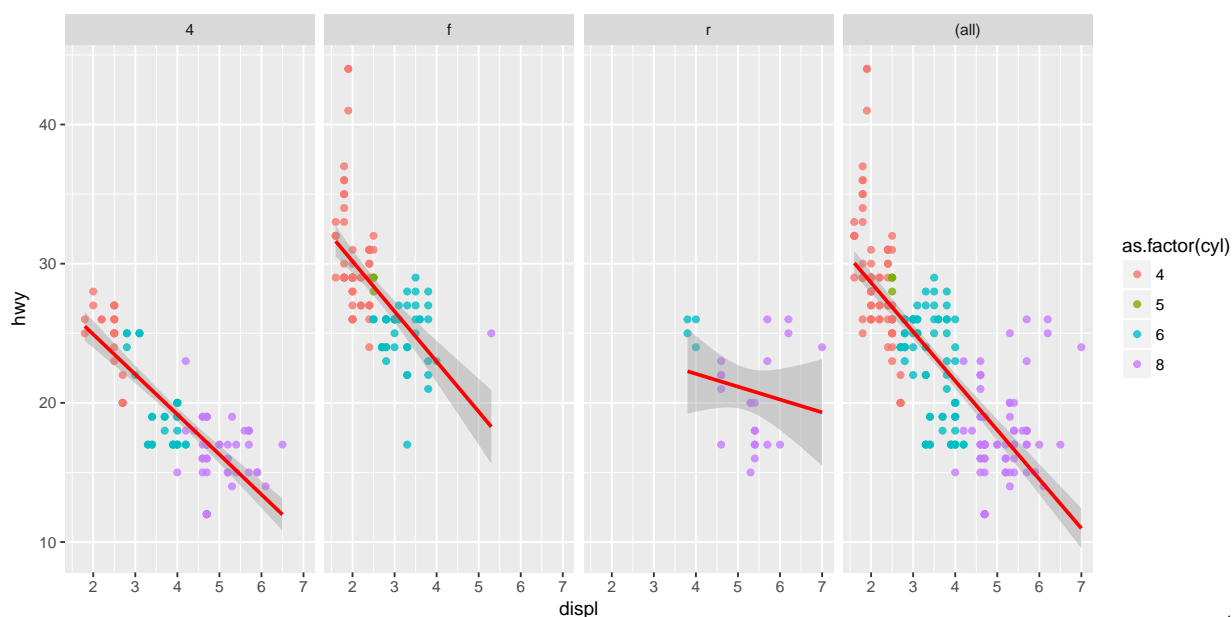
g <- ggplot(d, aes(x=mes, y=ruido, colour=grupo))

g + geom_line(size=0.3)
```

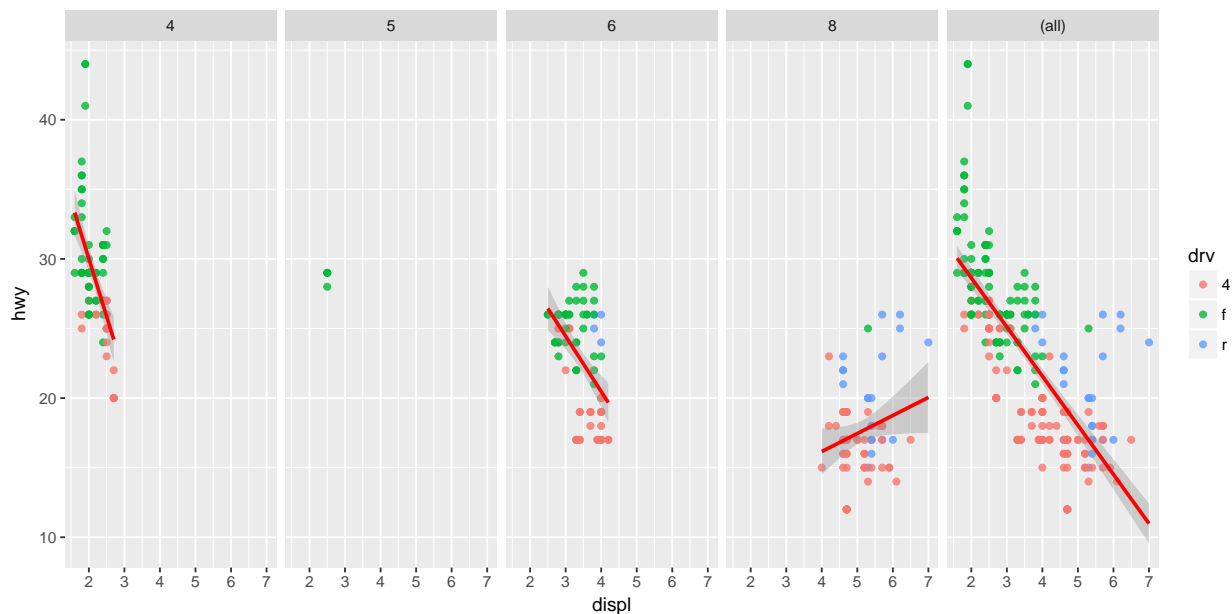


## 7.15. Generación de gráficos ggplot

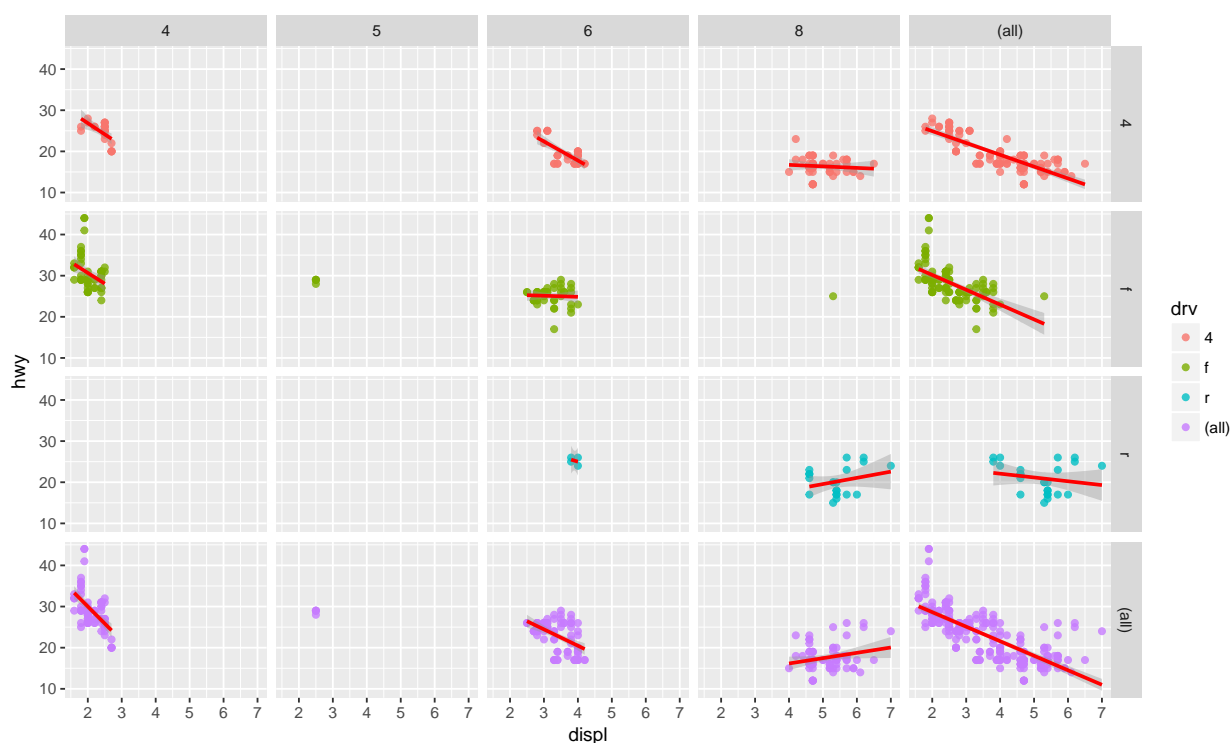
```
library(ggplot2)
g <- ggplot(mpg, aes(x=displ, y=hwy, color=as.factor(cyl)))
g + geom_point(alpha=0.8) + geom_smooth(method="lm",colour="red") +
  facet_grid(.~drv,margins = TRUE)
```



```
library(ggplot2)
g <- ggplot(mpg, aes(x=displ, y=hwy, color=drv))
g + geom_point(alpha=0.8) + geom_smooth(method="lm", colour="red") +
  facet_grid(.~cyl,margins = TRUE)
```



```
library(ggplot2)
g <- ggplot(mpg, aes(x=displ, y=hwy, color=drv))
g + geom_point(alpha=0.8) + geom_smooth(method="lm", colour="red") +
  facet_grid(drv ~ cyl,margins = TRUE)
```



## 8. Bibliografía

Faraway, J.J. (2004). Linear models with R. Chapman and Hall.

Faraway, J.J. (2006). Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models. Chapman and Hall.

Ritz, C. y Streibig, J.C. (2008). Nonlinear regression with R. Springer.

Sheather, S.J. (2009). A modern approach to regression with R. Springer.