Programación y Estadística con R

Resumen

Este curso es una introducción rápida a un «entorno para la computación estadística y los gráficos», que proporciona una amplia variedad de técnicas estadísticas y gráficas: modelización lineal y no lineal, pruebas estadísticas, análisis de series temporales, clasificación, agrupación, etc. Prácticamente todos los análisis estadísticos que se realizan en Bioinformática se pueden llevar a cabo con R. Además, la «minería de datos» está bien cubierta en R: el clustering (a menudo llamado «análisis no supervisado») en muchas de sus variantes (jerárquico, k-means y familia, modelos de mezcla, fuzzy, etc), bi-clustering, clasificación y discriminación (desde el análisis discriminante a los árboles de clasificación, bagging, máquinas de vectores soporte, etc), todos tienen muchos paquetes en R. Así, tareas como la búsqueda de subgrupos homogéneos en conjuntos de genes/sujetos, la identificación de genes que muestran una expresión diferencial (con ajuste para pruebas múltiples), la construcción de algoritmos de predicción de clases para separar a los pacientes de buen y mal pronóstico en función del perfil genético, o la identificación de regiones del genoma con pérdidas/ganancias de ADN (alteraciones del número de copias) pueden llevarse a cabo en R de forma inmediata.

Sandra Mingo Ramírez

UAM - 2024/25

25 de noviembre de 2024 18:05

Universidad Autónoma de Madrid
Bioinformática y Biología Computacional

Código en Github

Índice general

Parte I

Programación en R

Capítulo I

RStudio y primeras nociones

En RStudio, se puede crear un nuevo fichero en File > New File > R script. Se abre un nuevo fichero en el que se puede programar. En R, la asignación de variables se realiza con <-. En la parte superior derecha, se pueden ver todas las variables que se han asignado en la sesión, los datos y las funciones.

```
x <- 9
y <- matrix(1:20, ncol = 4)
```

En la parte inferior derecha hay una pestaña para poder visualizar los gráficos. Desde ese menú, se puede guardar, pero esto no es recomendable, ya que el gráfico se ajusta al tamaño de la pantalla y luego eso no es reproducible. En otra pestaña aparece un listado de todos los paquetes instalados en el disco duro, aunque luego haya que cargarlos en cada script en el que se desee usar. Al pulsar en el nombre de un paquete, se va a la página de ayuda del mismo. También es posible acceder con:

```
help(rnorm)
```

La mayor parte del trabajo «real» con R requerirá la instalación de paquetes. Los paquetes proporcionan funcionalidad adicional. Los paquetes están disponibles en muchas fuentes diferentes, pero posiblemente las principales ahora son CRAN y BioConductor. Si un paquete está disponible en CRAN, puedes hacer lo siguiente:

```
install.packages("nombre-paquete") # 1 paquete
install.packages(c("paquete1", "paquete2")) # varios paquetes
```

En Bioinformática, BioConductor es una fuente bien conocida de muchos paquetes diferentes. Los paquetes de BioConductor pueden instalarse de varias maneras, y existe una herramienta semiautomatizada que permite instalar conjuntos de paquetes BioC. Implican hacer algo como

```
BiocManager::install("nombre-paquete")
```

A veces los paquetes dependen de otros paquetes. Si este es el caso, por defecto, los mecanismos anteriores también instalarán las dependencias. Con algunas interfaces

gráficas de usuario (en algunos sistemas operativos) también puede instalar paquetes desde una entrada de menú. Por ejemplo, en Windows, hay una entrada en la barra de menú llamada Paquetes, que permite instalar desde Internet, cambiar los repositorios, instalar desde archivos zip locales, etc. Del mismo modo, desde RStudio hay una entrada para instalar paquetes (en «Herramientas»). Los paquetes también están disponibles desde otros lugares (RForge, github, etc); a menudo encontrarás instrucciones allí.

Siempre puedes simplemente matar RStudio; pero eso no es agradable. En todos los sistemas escribir q() en el símbolo del sistema debería detener R/RStudio. También habrá entradas de menú (por ejemplo, «Salir de RStudio» en «Archivo», etc). A continuación sale la pregunta de si se debe guardar el workspace, y en general querremos decir que no.

Capítulo II

Ejemplo

II.1. Introducción al test de la t

En un test de la t, la hipótesis nula (H_0) suele representar lo contrario de lo que se desea demostrar. Por ejemplo, si nuestro objetivo es comprobar si hay diferencias entre dos muestras, la hipótesis nula establece que ambas son iguales. A continuación, se utiliza la fórmula de la t para obtener un valor estadístico, cuya distribución se examina bajo la suposición de que H_0 es cierta. Luego, se calcula la probabilidad de observar un resultado tan extremo o más extremo que el obtenido bajo H_0. Esta probabilidad se denomina p-valor, y su interpretación indica cuánta evidencia hay en contra de H_0: un p-valor bajo sugiere que lo observado es improbable bajo H_0.

$$t = \frac{x_A - x_B}{SD_{x_A, x_B}}$$

Es importante aclarar que el p-valor no representa la probabilidad de que H_0 sea cierta, ni la probabilidad de que H_0 o la hipótesis alternativa (H_1) se cumplan dado los datos. Lo que el p-valor señala es que, o bien H_0 es falsa, o ha ocurrido un evento tan improbable como el valor observado. No se "rechaza" H_0 de manera concluyente, sino que simplemente no se acepta si el p-valor es suficientemente bajo. En este análisis, se compara el resultado observado con todos aquellos más extremos, algo que es distinto de seleccionar el valor que hace los datos lo más probables posible (como se hace en la máxima verosimilitud).

Por ejemplo, una moneda perfectamente equilibrada tiene una probabilidad de 0.5^6 de que al lanzarla seis veces, salga exactamente tres veces cara y tres veces cruz. Aunque este número es pequeño, no implica que la hipótesis alternativa sea necesariamente más probable, ya que otros resultados también podrían ser igualmente o más improbables. En la mayoría de los casos de comparación de medias, los datos no están restringidos a un único valor.

Cuando H_0 es cierta:

$$Pr(p - valor \leq 0,05) = 0,05$$

$$Pr(p - valor \leq 0,01) = 0,01$$

En muchos casos se comprueba más de una H_0. En un screening, se analizan 20.000 genes y se decide elegir todos aquellos que tengan un p-valor inferior a 0,05. Esa lista, sobre el total de los genes, la probabilidad de rechazar H_0 cuando es cierta, es muy superior al 5 %, aunque se cumpla para cada gen individual. Así, se debe trasladar la lógica al test múltiple, puesto que si no se va a rechazar H_0 en muchas ocasiones cuando no se debería.

II.2. Problema de las pruebas múltiples

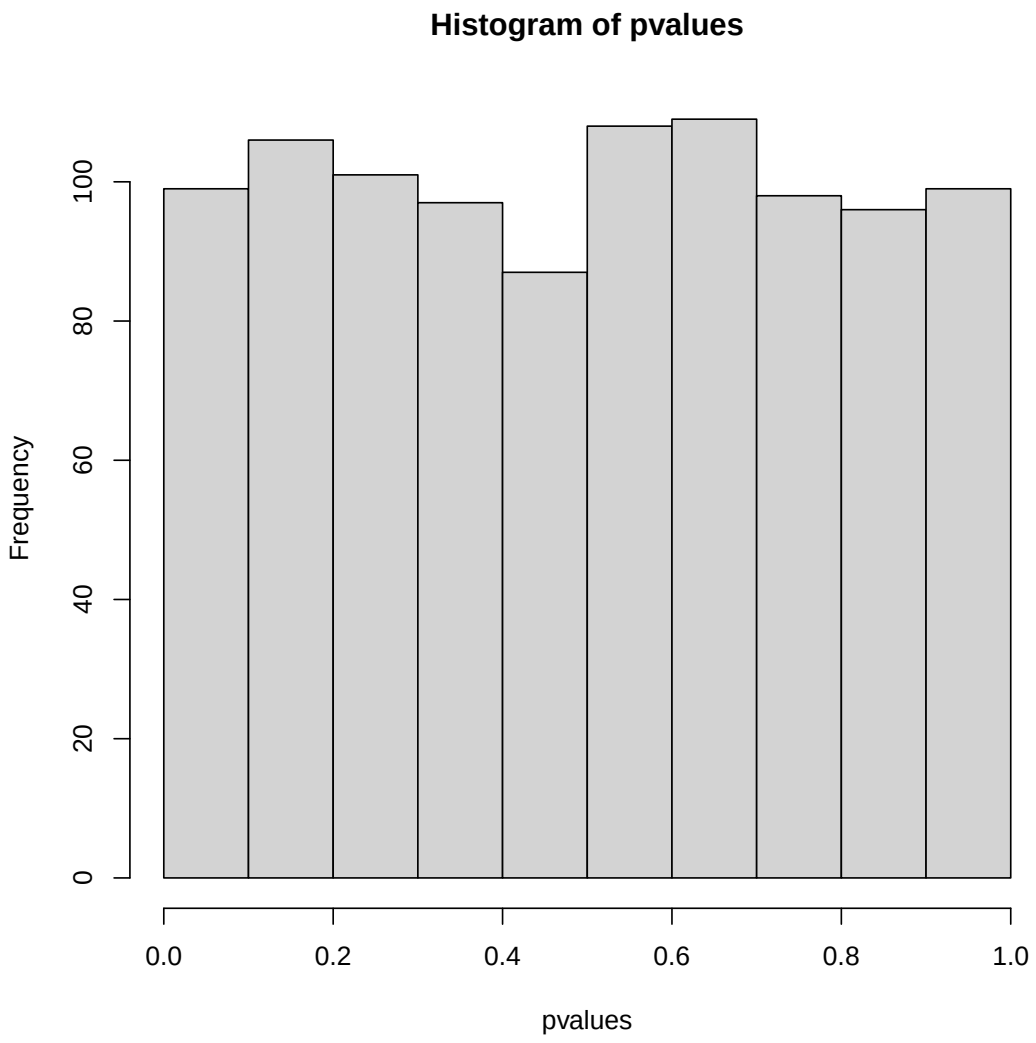
Es posible que hayamos oído hablar del problema de las pruebas múltiples con los microarrays: si observamos los p-valores de un gran número de pruebas, podemos ser inducidos a pensar erróneamente que está ocurriendo algo (es decir, que hay genes expresados de forma diferencial) cuando, en realidad, no hay absolutamente ninguna señal en los datos. A nosotros esto nos convence. Pero tienes un colega testarudo que no lo está. Ha decidido utilizar un ejemplo numérico sencillo para mostrarle el problema. Este es el escenario ficticio: 50 sujetos, de los cuales 30 tienen cáncer y 20 no. Medimos 1000 genes, pero ninguno de los genes tiene diferencias reales entre los dos grupos; para simplificar, todos los genes tienen la misma distribución (una distribución normal). Haremos una prueba t por gen, mostrará un histograma de los valores p e informaremos del número de genes «significativos» (genes con p < 0,05). Este es el código R:

```
randomdata <- matrix(rnorm(50 * 1000), ncol = 50)
class <- factor(c(rep("NC", 20), rep("cancer", 30)))
pvalues <- apply(randomdata, 1,
                 function(x) t.test(x ~ class)$p.value)
```

Para leer el código, se empieza por la función más interna, que en este caso es `rnorm`. Así, primero se generan 50.000 entradas de distribución normal (1000 genes por 50 personas) de los que se quiere realizar 1000 contrastes de hipótesis (uno por gen) y representar el aspecto de la distribución (que será uniforme). Todas las entradas se organizan en una matriz con 50 columnas. Después, se crean los dos grupos que se están analizando mediante repeticiones (función `rep`). El comando de `factor` crea las etiquetas. En R, se puede llamar al test de la t de varias maneras, siendo una estándar con la interfaz de tipo fórmula (x ∼ class), dividiendo así x en los distintos niveles que se han creado previamente. La sintaxis siempre es una variable que va cambiando (en este caso, las filas) antes de la virgulilla y una variable constante después de la virgulilla (los distintos niveles). La función `apply` permite aplicar una función a un objeto o conjunto de datos, evitando así tener que realizar un bucle for. El primer argumento es el objeto, el segundo la dimensión del objeto a lo que se quiere aplicar (si se recorren filas, columnas, etc.), y el tercero la función que se va a aplicar. La función `t.test` devuelve objetos a los que se puede acceder, como el valor t, df, p-valor, la media de cada grupo, etc. Se puede acceder al nombre de todos los valores mediante `names(t.test(x ~ class))`. En nuestro caso, x es el valor que irá adquiriendo el número de filas a recorrer. En este caso, se define la función en el momento de llamarla, pero también se puede definir antes y utilizarla en el apply. En este caso se define dentro porque es una función corta que solo se utilizará en ese momento, por

lo que no es necesario crearla fuera. Si por el contrario fuese una función a la que quisiéramos acceder posteriormente o que fuese compleja con varias líneas, se suele crear fuera. Por último, se accede a los p-valores y se guardan en la variable `pvalues`. Esos p-valores se pueden representar a continuación en un histograma y calcular todos aquellos que sean menores o iguales que 0,05.

```
hist(pvalues)
```




```
sum(pvalues <= 0.05)

## [1] 49
```

Al realizar la suma de una lógica booleana, se coercia para que los valores falsos se conviertan en 0 y los verdaderos en 1. Así, al sumarlos, el resultado es numérico.

En resumen, en este ejemplo hemos visto los siguientes objetos:

- Vectores: colección de uno o más datos del mismo tipo.

- Matrices: conjunto de datos indexados por filas y columnas del mismo tipo.
- Arrays: generalización de una matriz que no tiene límite de dimensiones (pero debe tener una estructura rectangular).
- Data frames: estructura rectangular de dos dimensiones (filas y columnas) en la que cada columna puede ser de un tipo diferente.
- Listas: cajón desastre en el que se pueden meter muchas cosas de muchos tipos distintos. Muchas funciones devuelven listas u objetos que contienen listas.
- Factores: vectores de un tipo especial (variable categórica).
- Funciones: objetos que realizan una operación y devuelven algo.

En el siguiente código se muestran las distintas maneras de acceder a una matriz. La indexación funciona [filas, columnas], y si un campo está sin rellenar implica todos sus datos.

```
randomdata[, 1]
randomdata[2, ]
randomdata[, 2]
randomdata[2, 3]
```

Al ejecutar la variable `class` creada anteriormente, no solo devuelve la lista de los elementos con las distintas etiquetas, si no que también muestra al final los distintos niveles. Como `factor` por detrás les asignó un valor entero que corresponda a la etiqueta dada, cuando se pide convertir en numérico, se devuelve el entero. La asignación de los valores se realiza por orden alfanumérico.

```
class
as.numeric(class)
```

```
pvalues[1]

t.test(randomdata[1, ] ~ class)

t.test(randomdata[1, ] ~ class)$p.value

pvalues[1:10] < 0.05

sum(c(TRUE, TRUE, FALSE))

hist(c(1, 2, 7, 7, 7, 8, 8))
```

```
## For ease
rd2 <- randomdata[1:10, ]

## Where we will store results
pv2 <- vector(length = 10)

for(i in 1:10) {
    pv2[i] <- t.test(rd2[i, ] ~ class)$p.value
}

pv2

## Compare with
pvalues[1:10]
```

Ahora usamos apply. No lo hemos dicho explícitamente, pero cuando usamos apply estamos pasando una función (nuestra función anónima) a otra función. Esto es algo muy común y fácil en R: pasar funciones a otras funciones.

```
apply(rd2, 1, function(z) t.test(z ~ class)$p.value)
```

Esta es otra forma de hacerlo, pero es más verbosa (quizás incluso innecesariamente verbosa):

```
myfunction <- function(y, classfactor = class) {
    t.test(y ~ classfactor)$p.value
}

apply(rd2, 1, myfunction)
```

Capítulo III

La consola de R para cálculos interactivos

Independientemente de cómo interactuemos con R, una vez que iniciemos una sesión interactiva de R, siempre habrá una consola, que es donde podemos introducir comandos para que sean ejecutados por R. En RStudio, por ejemplo, la consola suele estar situada en la parte inferior izquierda.Todos los prompts en la consola empiezan con >.

```
1 + 2
```

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

Mira la salida. En este documento, los trozos de código, si muestran salida, mostrarán la salida precedida por ##. En R (como en Python), # es el carácter de comentario. En la consola, NO veremos el ## precediendo a la salida. Esto es sólo la forma en que está formateado en este documento (al igual que no se ve el > antes del comando). Fíjate también en que ves un [1], antes del 3. Esto se debe a que la salida de esa operación es, en realidad, un vector de longitud 1, y R está mostrando su índice. Aquí no ayuda mucho, pero lo haría si imprimiéramos 40 números:

```
1:40
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
## [21] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Se puede asignar 1 + 2 a una variable mediante <-. También se puede utilizar =, pero no se aconseja. Esto se debe a que se suele utilizar = cuando se pasan argumentos a una función, y utilizar la flecha permite diferenciar a simplevista las asignaciones. Para ver el valor de una variable, se puede escribir simplemente el nombre de la variable, utilizar print o hacer la asignación entre paréntesis (eso realiza la asignación y muestra el resultado por pantalla).

```
(v1 <- 1 + 2)
```

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

```
print(v1)
```

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

```
v1
```

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

Se pueden separar dos comandos con un punto y coma (;), pero utilizarlo no suele ser una buena idea, solo en casos muy concretos.

```
v1 <- 1 + 2; v1
```

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

Es posible dividir comandos en varias líneas si R puede entender que la expresión no se ha terminado:

```
v2 <-  4 - ( 3 * [Enter]
2)
```

Cuando se hace esto, se ve un + que indica que la línea se continúa y que R sigue esperando más input. No obstante, hay ocasiones en las que esto puede ser confuso, y se puede cancelar mediante Ctrl + c en Linux o pulsando Escape para abortar la operación.

Los paréntesis se ponen cuando el usuario opine que es apropiado y que facilite el entendimiento de una expresión. R utiliza las normas de precedencia usuales, pero en caso de duda, se pueden utilizar paréntesis.

```
v11 <- 3 * ( 5 + sqrt(13) - 3^(1/(4 + 1)))
```

III.1. Nombrar variables

Anteriormente hemos creado las variables v1 y v2. Los nombres de las variables deben comenzar con una letra. También pueden empezar por un punto, pero entonces estarán ocultas. A continuación se pueden mezclar letras, números, puntos y barras bajas. Los nombres de las variables son case-sensitive, es decir, se diferencia entre las mayúsculas y minúsculas (v1 es diferente a V1). Una vez que se ha creado una variable, se puede utilizar la variable en lugar del contenido:

```
v3 <- 5
(v4 <- v1 + v3)

## [1] 8

(v5 <- v1 * v3)

## [1] 15

(v6 <- v1 / v3)

## [1] 0.6
```

Las asignaciones posteriores sobreescriben las asignaciones previas.

```
(z2 <- 33)

## [1] 33

z2 <- 999
z2

## [1] 999

z2 <- "Now z2 is a sentence"
z2

## [1] "Now z2 is a sentence"
```

Se puede borrar una variable de la siguiente forma:

```
rm(z2)
```

III.2. Obtener ayuda

Se puede acceder a la página de ayuda mediante:

```
help(mean)
```

También se puede utilizar la siguiente sintaxis:

```
?mean
```

Hay otras formas de buscar ayuda sobre cómo hacer algo con R. Se puede buscar en Google, utilizar StackOverflow, etc. También hay un paquete `sos` que ayuda a buscar funciones y demás en paquetes que no están instalados, hacer un ranking de resultados de búsqueda, etc. A su vez, RStudio incluye un navegador de ayuda integrado. Todas las ayudas cuentan con una descripción de la función, los argumentos que admiten (y su orden en caso de pasarlos sin nombre; en general es mejor añadir el nombre de cada parámetro a la hora de pasarlo) y el valor, es decir, lo que devuelve. En algunos casos se especifican las fuentes y referencias. También hay una sección de ejemplos de uso de la función.

Lo visto anteriormente proporciona información de funciones concretas. No obstante, hay veces que no sabemos exactamente cómo se llama la función que buscamos. Para ello, se puede utilizar las siguientes formas:

```
apropos("normal")

## [1] "normal_print"  "normalizePath"

# help.search("normal")
```

El comando `apropos` busca todos los paquetes que contengan en el nombre lo que se esté buscando. Por el contrario, `help.search` busca todos aquellos paquetes que, en la página de ayuda, tengan lo que se esté buscando.

La función `args` devuelve los argumentos que se le puede pasar a una función.

```
args(rnorm)

## function (n, mean = 0, sd = 1)
## NULL
```

III.3. Mensajes de error

Los mensajes de error pueden ser un poco crípticos, pero en muchos casos leerlos ayuda a entender qué está pasando y cómo solucionar el problema. La mejor forma de parsear el mensaje de error es ir a la última línea que se ha ejecutado e ir ascendiendo para ver dónde puede estar el problema. A continuación se muestran algunos ejemplos de mensajes de errores:

```
apply(something, 1, mean)

## Error: objeto 'something' no encontrado
```

```r
apply(v3, 1, mean) # en la ayuda se especifica qué es X
```

```
## Error in apply(v3, 1, mean): dim(X) debe tener una longitud positiva
```

```r
apply(F, 1, mean)
```

```
## Error in apply(F, 1, mean): dim(X) debe tener una longitud positiva
```

```r
log("23")
```

```
## Error in log("23"): Argumento no numérico para una función matemática
```

```r
rnorm("a")
```

```
## Warning in rnorm("a"): NAs introducidos por coerción
## Error in rnorm("a"): invalid arguments
```

```r
lug(23) # debería ser log
```

```
## Error in lug(23): no se pudo encontrar la función "lug"
```

```r
rnorm(23, 1, 1, 1, 34)
```

```
## Error in rnorm(23, 1, 1, 1, 34): los argumentos no fueron usados
(1, 34)
```

```r
x <- 1:10
y <- 11:21
plot(x, y)
```

```
## Error in xy.coords(x, y, xlabel, ylabel, log): 'x' and 'y' lengths
differ
```

```r
lm(y ~ x)
```

```
## Error in model.frame.default(formula = y ~ x, drop.unused.levels
= TRUE): las longitudes variables difieren (encontradas para 'x')
```

```r
z <- 1:10
t.test(x ~ z)
```

```
## Error in t.test.formula(x ~ z): grouping factor must have exactly
2 levels
```

En la consola, poniendo el nombre de la función, se puede acceder al código que realiza la función por detrás. Esto puede ser útil cuando la página de ayuda no sea suficiente para intentar localizar lo que intenta hacer la función y por qué falla.

III.4. Estilo del código

Aunque el código se escriba para la máquina, también debe ser legible por humanos, tanto uno mismo del futuro como otras personas. Por tanto, se recomienda no extenderse más allá de la columna 80 y utilizar espacios. Hay muchas guías de estilo de código, pero esas dos normas son las más básicas: si una línea de código es excesivamente larga, cuesta leerla entera al no poder verla completa a simple vista y tener que scrollear.

Existe un paquete llamado `lintr` que permite corregir el estilo del código.

Los comentarios también forman parte del estilo de código. Se suele separar la documentación para el usuario de la función (documentación de cabecera) de la documentación dentro del código que explica por qué se hacen algunas cosas.

Capítulo IV

Leer datos en R y guardarlos desde R

Hay muchas formas de cargar datos en R. Un ejemplo es `read.table` que sirve para todo tipo de datos, pero también hay algunos comandos más concretos como `read_csv`.

```
X <- read.table("data/hit-table-500-text.txt")
head(X)
## We could save what we care about in variables with better names
align.length <- X[, 5]
score <- X[, 13]
summary(X)
```

El objeto no es una matriz, si no un data frame. Otro ejemplo sería el siguiente:

```
another.data.set <- read.table("data/AnotherDataSet.txt", header = TRUE)
summary(another.data.set)
```

```
##       ID                  Age           Sex
##  Length:5           Min.   :12.0   Length:5
##  Class :character   1st Qu.:13.0   Class :character
##  Mode  :character   Median :14.0   Mode  :character
##                     Mean   :14.8
##                     3rd Qu.:16.0
##                     Max.   :19.0
##        Y
##  Min.   :22.00
##  1st Qu.:23.40
##  Median :24.30
##  Mean   :24.14
##  3rd Qu.:25.00
##  Max.   :26.00
```

Si se pone que no hay cabecera, parece que se lee lo mismo, pero en realidad hay algunas diferencias. Cuando se especifica que hay una cabecera, la primera línea con la descripción de las columnas no está numerada, mientras que cuando no se especifica, sí se numera y se considera como la primera fila, y esto es un error. R, por defecto, pone que cabecera es falso. Cuando no se sabe si un documento tiene o no cabecera, primero se carga el documento y luego se comprueba si el contenido se ha cargado bien. Por defecto, las columnas están separadas por espacios o tabuladores.

IV.1. Localización de ficheros

Para que R pueda leer los ficheros, debe saber dónde buscarlos. Si los ficheros se encuentran en el directorio de trabajo, no hay ningún problema, ya que R los encuentra directamente. Para conocer el directorio de trabajo, se utiliza el comando `getwd()`. Si el fichero no se encuentra en el directorio de trabajo, hay varias opciones: proporcionar el path completo o mover el directorio de trabajo al lugar donde se encuentran los ficheros mediante `setwd()`. Para esto, es recomendable evitar en el nombre de directorios espacios, acentos y otros caracteres no ASCII.

IV.2. Missing values - NA

Los missing values son algo muy común en estadística. Lo más sencillo es llamarlos como NA de not available. Otra forma es NaN, not a number.

Puedes especificar el carácter que R debe interpretar como valor omitido, pero los dos procedimientos estándares son sustituir el valor como NA o sustituirlo por nada. Cuando haces cualquiera de los dos, en los datos que se leen deberías ver un NA. Lo mejor es, como de costumbre, ser explícito: utilizar un NA en sus datos originales, o utilizar alguna otra cadena de caracteres especiales para identificarlos. Lo más probable es que desees utilizar NA (o utilizar alguna otra combinación de caracteres y ser explícito), especialmente para las variables de carácter.

Por defecto, R considera cualquier secuencia de blancos y tabuladores como separadores. Por tanto, si un missing value se representa con un espacio, sería necesario especificar el separador (por ejemplo, `sep = "\n"`) para que no dé error (al considerar R el espacio como parte del separador).

Al utilizar `summary`, en las columnas que sean de tipo int aparece un contador con las filas que contienen un NA. Sin embargo, esto no es así en las columnas cuyo contenido sea texto. Por tanto, no nos podemos fiar si summary no nos dice que no hay, hay que comprobar que efectivamente no haya.

IV.3. Guardar tablas, datos y resultados

Es posible guardar los datos en uan matriz o de forma tabular con `write.table`:

```
write.table(X, file = "datos_guardados.txt")
```

El problema que tiene esto es que en el documento de salida tiene una columna adicional que indica el número de línea, y se emplean los espacios como separadores. Todo esto se puede especificar mediante argumentos concretos en la función:

```
write.table(X, file = "datos_guardados.txt", sep = "\t",
            quote = FALSE, row.names = FALSE)
```

En algunos casos, puede que los nombres de las filas sean importantes (por ejemplo, que sean el identificador). En ese caso, sería interesante guardar los nombres de las filas como columna en el dataframe:

```
X$columna_nueva <- rownames(X)
```

IV.4. Guardar una sesión en R: .RData

R permite guardar una imagen de la sesión actual en un fichero de extensión .RData. Esto se realiza mediante la función `save.image`:

```
save.image(file = "this.RData")
getwd() #donde se guarda
```

Esta función guarda el entorno global, es decir, lo que se haya añadido por el usuario: variables, ficheros (incluso los ocultos), funciones, pero no los paquetes. También se guarda el estado del generador de números aleatorios si se ha utilizado. También existe la posibilidad de guardar un objeto concreto. Esto se logra mediante `save(datos-a-guardar, file = "datos-guardados.RData")`.

En una nueva terminal de R, se pueden cargar las imágenes (ya sea la total o de unos objetos concretos) con `load("datos-guardados.RData")`.

Por último, es posible utilizar `saveRDS` para guardar objetos individuales serializados (en binario) y `readRDS` para leerlos posteriormente. Sirve para un único objeto, pero permite poder asignarlo a un nombre que se decide al cargarlo.

Capítulo V

Scripts

Mantener todo el código en uno o varios scripts y ejecutarlo directamente desde el script y no desde la consola tiene varias ventajas:

- Permite mantener un registro de todo lo que se ha hecho y tenerlo organizado, con comentarios, etc.
- Permite realizar cálculos no interactivos. Por ejemplo, ejecutar un análisis muy largo o volver a ejecutar todo el análisis y los gráficos sin querer.

V.1. Utilizar un script

Hay dos maneras básicas de utilizar un script:

- De forma interactiva; lo que se ha hecho hasta entonces. Por ejemplo, RStudio permite seleccionar una parte del código y lanzarlo al intérprete de R, ejecutándolo desde la consola.
- De forma no interactiva:
  - Utilizando `source("script.R")`. En la sesión de R en la que se haya puesto esto, se importan las variables, funciones (y todo) del script. La diferencia es que, como es no interactivo, si se llaman a funciones (como por ejemplo, `mean(x)`), no se muestra el resultado por pantalla; para ello sería necesario utilizar print.
  - Desde la shell. Esto tiene la ventaja de no tener que mantener una ventana abierta con R hasta que el código finalice, por lo que es muy cómodo para los trabajos muy largos. La forma preferida es:

    ```
    R --vanilla < script1.R > script1.Rout
    ```

    La opción de vanilla permite que la sesión sea lo más reproducible posible, es decir, sin cargar librerías adicionales, sesiones de R anteriores, etc. Otra manera muy similar es `R --vanilla -f script1.R > script1.Rout` Con esto lo que conseguimos es que el resultado del script1 se guarde directamente en otro fichero.

Capítulo VI

Estructuras de datos básicas en R

VI.1. Vectores

Los vectores son la estructura de datos más simple de R. Guardan una serie de objetos del mismo tipo, uno detrás de otro, en una sola dimensión.

```
v1 <- c(1, 2, 3) #vector de números enteros
#                (se guardan como floats si no se fuerza)
v2 <- c("a", "b", "cucu") #vector de strings
v3 <- c(1.9, 2.5, 0.6) #vector de números float
v4 <- c(4, "a") #convierte el 4 en "4"
```

La c viene de concatenar, ya que hace precisamente eso: concatena lo que se le ponga a continuación.

Muchas funciones operan directamente en vectores enteros sin necesidad de realizar un loop sobre cada uno de los objetos en él:

```
log(v1)
```

```
## [1] 0.0000000 0.6931472 1.0986123
```

```
exp(v3)
```

```
## [1]  6.685894 12.182494  1.822119
```

```
2 * v1
```

```
## [1] 2 4 6
```

```
v3/0.7
```

```
## [1] 2.7142857 3.5714286 0.8571429
```

VI.1.1. Funciones para crear vectores

Se pueden crear vectores concatenando elementos, pero hay otras dos funciones para crearlos que tienen algo de estructura: `seq` (de secuencia) y `rep` (de repetición). La función `seq` tiene cuatro formas de invocación:

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

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

seq(from = 1, to = 10, by = 2)

## [1] 1 3 5 7 9

seq(from = 1, to = 10, length.out = 3)

## [1]  1.0  5.5 10.0

1:5

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

`rep` también tiene varias invocaciones comunes:

```
rep(2, 5)

## [1] 2 2 2 2 2

rep(1:3, 2)

## [1] 1 2 3 1 2 3

rep(1:3, 2:4)

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

En este caso, es importante que el segundo argumento de `rep` sea un único valor (y repita todos los elementos del primer argumento las veces indicadas) o un conjunto de valores de las mismas dimensiones que el primer argumento (y se asigne a cada valor su respectivo número de repetición).

VI.2. Crear vectores a partir de otros vectores

Se pueden concatenar dos vectores:

```
v1 <- 1:4
v2 <- 7:12
(v3 <- c(v1, v2))

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

Si se emplean operaciones aritméticas en vectores que no son de la misma longitud, se utiliza la **regla de reciclaje**, es decir, se reutiliza el vector más pequeño cuando llega a su fin las veces necesarias hasta haber terminado las operaciones con el vector grande:

```
v1 <- 1:3
v2 <- 11:12
v1 + v2

## Warning in v1 + v2: longitud de objeto mayor no es múltiplo de la
longitud de uno menor

## [1] 12 14 14
```

En ocasiones se produce un warning que avisa sobre la reutilización de uno de los vectores. Sin embargo, esto no ocurre siempre, ua que el warning se suprime cuando el vector a reutilizar se repite una ronda concreta (y no se quede a medias durante el reciclaje).

VI.3. Logical operations

Se pueden comparar los elementos de un vector con algo para obtener un vector de elementos lógicos TRUE y FALSE. Esto es común en varios lenguajes de programación, pero hay que tener en cuenta la diferencia entre | y || y entre && y &. También se puede usar `xor` para obtener TRUE cuando solo uno de las condiciones es verdadera (no ambas).

```
v1 <- 1:5
v1 < 3

## [1]  TRUE  TRUE FALSE FALSE FALSE

(v2 <- (v1 < 3))

## [1]  TRUE  TRUE FALSE FALSE FALSE

v11 <- c(1, 1, 3, 5, 4)
v1 == v11
```

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

```
v1 != v11
```

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

```
!(v1 == v11)
```

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

```
identical(v1, v11)
```

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

```
v3 <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
!v3
```

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

```
v2 & v3
```

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

```
v2 | v3
```

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

```
(v1 > 3) & (v11 >= 2)
```

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

```
(v1 > 3) | (v11 >= 2)
```

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

```
xor(v2, v3)
```

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

VI.3.1. Valores lógicos 0 y 1

En R, al igual que en otros lenguajes de programación, se pueden utilizar valores lógicos como si fuesen numéricos: se puede tratar `TRUE` como 1 y `FALSE` como 0. Además, `TRUE` puede ser cualquier otro número diferente a 0.

El operador `which` opera en un vector lógico, no en el vector directamente, y devuelve las posiciones que son verdaderas. `length` cuenta la longitud de la salida:

```
vv <- c(1, 3, 10, 2, 9, 5, 4, 6:8)
length(which(vv < 5))

## [1] 4
```

Es importante remarcar no utilizar T para TRUE y F para FALSE, aunque se pueda hacer. Esto se debe a que se puede redefinir el valor de T y F a que no correspondan a TRUE y FALSE (lo cual es muy difícil de debuggear), mientras que TRUE y FALSE siempre significarán lo mismo al no poder redefinirse.

VI.3.2. Cortocircuito de operaciones lógicas

Los operadores && y || son cortocircuitos. Los dobles caracteres evalúan el segundo elemento sólo si la evaluación del primero no permite saber el resultado de la operación. Cuando se va a hacer un and y la primera condición es FALSE, no hace falta evaluar la segunda condición, ya que se conoce el resultado (de igual forma si en un or la primera condición es TRUE). Así, esto se puede utilizar para condicionar la ejecución de la segunda condición:

```
a <- "hola"
if (is.numeric(a) && log(a)) cat("\n we entered in the if")
```

En el ejemplo anterior, sólo se quiere evaluar el logaritmo de un número. Por ello, con &&, primero se evalúa si la variable es un número y, en caso afirmativo, se ejecuta el logaritmo. En caso de que la variable no sea numérica (como es el caso del ejemplo), utilizar un solo & resultaría en un error, y no sería lo que nos interesa.

```
a1 <- c(TRUE, FALSE)
b1 <- c(TRUE, TRUE)

a1 && b1

## Error in a1 && b1: 'length = 2' in coercion to 'logical(1)'

a1 || b1

## Error in a1 || b1: 'length = 2' in coercion to 'logical(1)'
```

Hay que tener en cuenta que no se deben utilizar vectores con más de un elemento con cortocircuitos, ya que sólo se evalúa el primer valor.

VI.4. Nombres de elementos

Los elementos de un vector pueden tener nombres (que deben ser únicos). Esto permite acceder a los vectores utilizando nombres en lugar de posiciones, lo que puede ser más intuitivo.

```
ages <- c(Juan = 23, Maria = 35, Irene = 12, Ana = 93)
names(ages)

## [1] "Juan"  "Maria" "Irene" "Ana"

ages

##  Juan Maria Irene   Ana
##    23    35    12    93

ages["Juan"]

## Juan
##   23
```

VI.5. Acceder y modificar elementos de un vector: indexación y subsetting

VI.5.1. Indexación de vectores

Hay cuatro formas para acceder a elementos específicos de un vector:

- Especificando las posiciones: mediante índices
- Dando los nombres de los elementos
- Utilizando un vector lógico
- Utilizando cualquier expresión que genere cualquiera de las anteriores.

Las posiciones y nombres se dan entre corchetes ([]).

Especificando las posiciones deseadas:

```r
(w <- 9:18)
```

```
##  [1]  9 10 11 12 13 14 15 16 17 18
```

```r
w[1]
```

```
## [1] 9
```

```r
w[2]
```

```
## [1] 10
```

```r
w[c(4, 3, 2)]
```

```
## [1] 12 11 10
```

```r
w[c(1, 3)] ## not the same as
```

```
## [1]  9 11
```

```r
w[c(3, 1)]
```

```
## [1] 11  9
```

```r
w[1:2]
```

```
## [1]  9 10
```

```r
w[3:6]
```

```
## [1] 11 12 13 14
```

```r
w[seq(1, 8, by = 3)]
```

```
## [1]  9 12 15
```

```r
vv <- seq(1, 8, by = 3)
w[vv]
```

```
## [1]  9 12 15
```

Especificando las posiciones que no se desean (el vector original no se modifica):

```r
w[-1]

## [1] 10 11 12 13 14 15 16 17 18

w[-c(1, 3)] ## of course, the same as following

## [1] 10 12 13 14 15 16 17 18

w[-c(3, 1)]

## [1] 10 12 13 14 15 16 17 18
```

Utilizando nombres

```r
ages <- c(Juan = 23, Maria = 35, Irene = 12, Ana = 93)
ages["Irene"]

## Irene
##    12

ages[c("Irene", "Juan", "Irene")]

## Irene  Juan Irene
##    12    23    12
```

Utilizando un vector lógico ...

```r
ages[c(FALSE, TRUE, TRUE, FALSE)]

## Maria Irene
##    35    12

## what are thes doing? Avoid these things
ages[c(FALSE, TRUE)]

## Maria   Ana
##    35    93

ages[c(TRUE, TRUE, FALSE)]

##  Juan Maria   Ana
##    23    35    93
```

... o algo que es un vector lógico implícito

```r
## All less than 12
w[w < 12]

## [1]  9 10 11

## same, but more confusing (here, not always)
w[!(w >= 12)]

## [1]  9 10 11

## All less than the median
w[w < median(w)]

## [1]  9 10 11 12 13
```

Si se puede acceder, también se puede modificar:

```r
ages["Irene"] <- 19
ages

##  Juan Maria Irene   Ana
##    23    35    19    93

w[1] <- 9999
w

##  [1] 9999   10   11   12   13   14   15   16   17   18

w[vv] <- 103
w

##  [1] 103  10  11 103  13  14 103  16  17  18
```

Pero compara esto:

```r
w[] <- 77
w[] <- 17:55

## Warning in w[] <- 17:55: número de elementos para sustituir no es
un múltiplo de la longitud del reemplazo

w <- 17:55
```

VI.6. Interludio: comparación de floats

Comparar valores numéricos muy similares puede ser complicado y muy delicado debido al redondeo y algunos números que no se pueden representar exactamente en notación binaria. De forma predeterminada, R muestra 7 dígitos significativos.

```
x  <-  1.999999
x

## [1] 1.999999

x - 2

## [1] -1e-06

x <- 1.9999999999999
x

## [1] 2

x-2

## [1] -9.992007e-14
```

Todos los dígitos están presentes, pero en el segundo caso, no se muestran. Además, `x-2` no es exactamente -1×10^{-13}. En R se suelen redondear los números con una precisión de 53 dígitos binarios, por lo que dos números decimales no serán iguales de forma diable a menos que hayan sido calculados por el mismo algoritmo, y ni siquiera entonces:

```
a <- sqrt(2)
a * a == 2
[1] FALSE
a * a - 2
[1] 4.440892e-16
```

Otro ejemplo:

```
0.1 + 0.2 == 0.3

## [1] FALSE

(0.1 + 0.2) - 0.3

## [1] 5.551115e-17
```

En resumen: desconfía extremadamente siempre que veas una comparación de igualdad de dos números en coma flotante; es poco probable que haga lo que quieres. Si sabes lo que estás haciendo, echa un vistazo a `all.equal` para comparaciones de igualdad de objetos casi iguales.

VI.7. Factores

Los factores son unos tipos especiales de vectores. Los necesitamos para diferenciar entre un vector de caracteres y un vector que representa variables categóricas. El vector `char.vec <- c("abc", "de", "fghi")` contiene varias cadenas de caracteres. Supongamos ahora que tenemos un estudio en el que registramos el sexo de los participantes. Cuando analizamos los datos queremos que R sepa que se trata de una variable categórica, donde cada etiqueta representa un posible valor de la categoría:

```
Sex.version1 <- factor(c("Female", "Female", "Female",
                         "Male", "Male"))
Sex.version2 <- factor(c("XX", "XX", "XX", "XY", "XY"))
Sex.version3 <- factor(c("Feminine", "Feminine", "Feminine",
                         "Masculine", "Masculine"))
Sex.version4 <- factor(c("fe", "fe", "fe", "ma", "ma"))
```

Queremos que todas esas codificaciones del sexo de cinco sujetos arrojen los mismos resultados de análisis, independientemente de lo que digan exactamente las etiquetas. Cada conjunto de etiquetas puede tener sus pros y sus contras (por ejemplo, la tercera probablemente está codificando el género, no el sexo; la última es demasiado críptica; la segunda sólo funciona para algunas especies; etc.). Independientemente de las etiquetas, lo que hay que tener en cuenta es que los tres primeros sujetos son del mismo tipo y los dos últimos son de un tipo diferente.

Reconocer los factores es esencial cuando se trata de variables que parecen números legítimos:

```
postal.code <- c(28001, 28001, 28016, 28430, 28460)
somey <- c(10, 20, 30, 40, 50)
summary(aov(somey ~ postal.code))

##             Df Sum Sq Mean Sq F value Pr(>F)
## postal.code  1  782.5   782.5   10.79 0.0462 *
## Residuals    3  217.5    72.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Lo anterior está haciendo algo tonto: está ajustando una regresión lineal, porque está tomando postal.code como un valor numérico legítimo. Pero sabemos que no tiene sentido que 28009 y 28016 (dos distritos de Madrid) estén separados por 7 unidades mientras que 28430 y 28410 estén separados por 20 unidades (dos pueblos cercanos

al norte de Madrid), ni esperamos encontrar relaciones lineales con (el número del) código postal en sí.

A veces, al leer datos, una variable se convierte en factor, pero en realidad es una variable numérica. ¿Cómo convertirla en el conjunto original de números? Esto no funciona:

```
x <- c(34, 89, 1000)
y <- factor(x)
y

## [1] 34   89   1000
## Levels: 34 89 1000

as.numeric(y)

## [1] 1 2 3

y

## [1] 34   89   1000
## Levels: 34 89 1000
```

Los valores se han recodificado. Una forma sencilla de hacerlo es la siguiente:

```
as.numeric(as.character(y))

## [1]   34   89 1000
```

VI.7.1. Factores y símbolos, colores, etc en gráficos

Muchas veces se puede ver código como el siguiente:

```
plot(y ~ x, col = c("red", "blue")[group])
```

donde group es un factor de la longitud de x o y con dos niveles (si tuviese más, habría que proporcionar más colores).

Otro ejemplo:

```
legend(1, 2, legend = c("A", "B"), pch = c(1, 2),
       col = c("red", "blue")[factor(levels(group))])
```

En este caso, los colores se van a sacar en el mismo orden que los puntos. Aunque sea enreversado, lo que se pide son los niveles del grupo y convertirlo en un factor. Así,

los niveles se ponen en el orden que se tienen, y los colores se adjudican en ese mismo orden.

Un último ejemplo:

```
gr <- c("B", "A", "A", "B", "A")
group <- factor(gr)
c("red", "blue")[gr]

## [1] NA NA NA NA NA

c("red", "blue")[group]

## [1] "blue" "red"  "red"  "blue" "red"

c("red", "blue")[levels(group)]

## [1] NA NA

c("red", "blue")[factor(levels(group))]

## [1] "red"  "blue"
```

VI.8. Matrices

Los vectores son unidimensionales, mientras que las matrices son bidimensionales, y los arrays pueden tener un número arbitrario de dimensiones. Aquí nos quedaremos en las matrices. Como en vectores, todos los elementos de una matriz o de un array son del mismo tipo.

Las matrices se pueden crear desde un vector:

```
matrix(1:10, ncol = 2)

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

matrix(1:10, nrow = 5)

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

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

```
matrix(1:10, ncol = 2, byrow = TRUE)
```

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

```
matrix(1:15, nrow = 5, ncol = 2)
```

```
## Warning in matrix(1:15, nrow = 5, ncol = 2): data length [15] is
not a sub-multiple or multiple of the number of columns [2]
```

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

Por defecto, R rellena la matriz por columnas, pero se puede especificar que sea por fila.

VI.8.1. Combinar vectores para crear una matriz: `cbind`, `rbind`

Se pueden combinar vectores en horizontal o vertical para crear una matriz:

```
v1 <- 1:5
v2 <- 11:15
rbind(v1, v2)
```

```
##    [,1] [,2] [,3] [,4] [,5]
## v1    1    2    3    4    5
## v2   11   12   13   14   15
```

```
cbind(v1, v2)
```

```
##      v1 v2
## [1,]  1 11
## [2,]  2 12
## [3,]  3 13
## [4,]  4 14
## [5,]  5 15
```

También se puede hacer lo mismo con matrices siempre que tengan las dimensiones apropiadas:

```
A <- matrix(1:10, nrow = 5)
B <- matrix(11:20, nrow = 5)
cbind(A, B)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```
rbind(A, B)
```

```
##       [,1] [,2]
##  [1,]    1    6
##  [2,]    2    7
##  [3,]    3    8
##  [4,]    4    9
##  [5,]    5   10
##  [6,]   11   16
##  [7,]   12   17
##  [8,]   13   18
##  [9,]   14   19
## [10,]   15   20
```

VI.8.2. Indexación y subsetting en matrices

Una matriz tiene dos dimensiones, pero por lo demás funciona de forma similar a vectores. La primera dimensión son filas, y la segunda son columnas. Si no se especifica nada para una dimensión, se devuelve en su totalidad.

```
A <- matrix(1:15, nrow = 5)
A[1, ] ## first row
```

```
## [1]  1  6 11
```

```
A[, 2] ## second column

## [1]  6  7  8  9 10

A[4, 2] ## fourth row, second column

## [1] 9

A[3, 2] <- 999
A[1, ] <- c(90, 91, 92)
A < 4

##       [,1]  [,2]  [,3]
## [1,] FALSE FALSE FALSE
## [2,]  TRUE FALSE FALSE
## [3,]  TRUE FALSE FALSE
## [4,] FALSE FALSE FALSE
## [5,] FALSE FALSE FALSE
```

El operador `which` puede no hacer lo que uno espera por defecto. Si se quieren los índices, se debe especificar.

```
which(A == 999)

## [1] 8

which(A == 999, arr.ind = TRUE)

##      row col
## [1,]   3   2
```

También se puede indexar mediante los nombres de filas y columnas:

```
B <- A
colnames(B) <- c("A", "E", "I")
rownames(B) <- letters[1:nrow(B)]
B[, "E"]

##   a   b   c   d   e
##  91   7 999   9  10

B["c", ]

##   A   E   I
##   3 999  13
```

Se puede utilizar una matriz para indexar otra. Esto es algo más avanzado, pero puede venir muy bien:

```
(m1 <- cbind(c(1, 3), c(2, 1)))

##      [,1] [,2]
## [1,]    1    2
## [2,]    3    1

A[m1]

## [1] 91  3

## compare with
A[c(1, 3), c(2, 1)]

##      [,1] [,2]
## [1,]   91   90
## [2,]  999    3
```

Al indexar con una matriz, se devuelven tantos elementos como filas tiene la matriz.

Cuando se obtiene una sola columna, se pierde una dimensión y, en lugar de conseguir una matriz, el resultado es un vector. Para evitar esto, se puede emplear `drop = FALSE`

```
A[c(2, 4), 1, drop = FALSE]

##      [,1]
## [1,]    2
## [2,]    4
```

VI.8.3. Operaciones con matrices

Hay muchas operaciones matriciales disponibles en R (abre tu libro de álgebra matricial e intenta encontrarlas, si quieres). Y muchas funciones operan directamente, por defecto, sobre toda la matriz, o sobre filas/columnas de la matriz:

```
sum(B)

## [1] 1366

mean(B)

## [1] 91.06667
```

```
colSums(B) #rowSums

##    A    E    I
##  104 1116  146

rowMeans(B) #colMeans

##        a        b        c        d        e
##  91.0000   7.0000 338.3333   9.0000  10.0000
```

También se pueden seleccionar filas y columnas utilizando esas operaciones:

```
B[rowMeans(B) > 9, ]

##    A   E  I
## a 90  91 92
## c  3 999 13
## e  5  10 15
```

VI.9. Listas

Una lista es un contenedor general donde se pueden mezclar cosas de distintos tipos. De hecho, no debe por qué tener una estructura rectangular. Hay muchas formas de acceder a elementos de una lista.

```
listA <- list(a = 1:5, b = letters[1:3])
listA[1]

## $a
## [1] 1 2 3 4 5

listA[[1]]

## [1] 1 2 3 4 5

listA[["a"]]

## [1] 1 2 3 4 5

listA$a

## [1] 1 2 3 4 5

listA[[1]][2]

## [1] 2
```

Una lista más compleja sería la siguiente:

```
(listB <- list(one.vector = 1:10,  hello = "Hola",
               one.matrix = matrix(rnorm(20), ncol = 5),
               another.list =
               list(a = 5,
                    b = factor(c("male",
                      "female", "female")))))
```

```
## $one.vector
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $hello
## [1] "Hola"
##
## $one.matrix
##            [,1]       [,2]       [,3]       [,4]       [,5]
## [1,] -1.9859602 0.04951268  0.4858839  0.1783097  0.5981655
## [2,]  0.4183266 0.02503637 -1.1496227 -1.1748653  1.0635857
## [3,] -2.2278836 0.48155524  0.1188615  0.9027103 -0.9890780
## [4,] -0.2960181 0.27107194 -2.1135222 -0.6880739  0.7295614
##
## $another.list
## $another.list$a
## [1] 5
##
## $another.list$b
## [1] male   female female
## Levels: female male
```

```
listB[[c(3, 11)]]
```

```
## [1] 0.1188615
```

```
listB[[3]][11]
```

```
## [1] 0.1188615
```

```
listB[[3]][3, 3]
```

```
## [1] 0.1188615
```

```
listB[[3]][c(3, 3)]
```

```
## [1] -2.227884 -2.227884
```

```
listB[c(3, 4)]

## $one.matrix
##            [,1]       [,2]       [,3]       [,4]       [,5]
## [1,] -1.9859602 0.04951268  0.4858839  0.1783097  0.5981655
## [2,]  0.4183266 0.02503637 -1.1496227 -1.1748653  1.0635857
## [3,] -2.2278836 0.48155524  0.1188615  0.9027103 -0.9890780
## [4,] -0.2960181 0.27107194 -2.1135222 -0.6880739  0.7295614
##
## $another.list
## $another.list$a
## [1] 5
##
## $another.list$b
## [1] male   female female
## Levels: female male
```

VI.10. Dataframes

Un dataframe es una lista de vectores con la misma longitud y que pueden contener distintos tipos de objetos. La estructura es rectangular. Se pueden acceder a los elementos como si fueran matrices y como si fueran listas. Además, se pueden convertir dataframes en matrices con `data.matrix(df)` y `as.matrix(df)`. Muchas operaciones de matrices, concretamente `rbind` y `cbind`, también funcionan con dataframes.

```
(AB <- data.frame(ID = c("a1", "a2", "a3", "a4", "a5"),
                  Age = c(12, 14, 12, 16, 19),
                  Sex = c("M", "F", "F", "M", "F"),
                  Y = c(11, 14, 15, 12, 19)))

##   ID Age Sex  Y
## 1 a1  12   M 11
## 2 a2  14   F 14
## 3 a3  12   F 15
## 4 a4  16   M 12
## 5 a5  19   F 19

(AC <- data.frame(ID = "a9", Age = 14, Sex = "M", Y = 17))

##   ID Age Sex  Y
## 1 a9  14   M 17

(AB2 <- rbind(AB, AC))
```

```
##   ID Age Sex  Y
## 1 a1  12   M 11
## 2 a2  14   F 14
## 3 a3  12   F 15
## 4 a4  16   M 12
## 5 a5  19   F 19
## 6 a9  14   M 17
```

```
as.matrix(AB) #convierte todo en strings
```

```
##      ID   Age  Sex Y
## [1,] "a1" "12" "M" "11"
## [2,] "a2" "14" "F" "14"
## [3,] "a3" "12" "F" "15"
## [4,] "a4" "16" "M" "12"
## [5,] "a5" "19" "F" "19"
```

```
data.matrix(AB) #convierte todo en números
```

```
##      ID Age Sex  Y
## [1,]  1  12   2 11
## [2,]  2  14   1 14
## [3,]  3  12   1 15
## [4,]  4  16   2 12
## [5,]  5  19   1 19
```

Es muy fácil añadir nuevas variables a los dataframes:

```
AB2$status <- rep(c("Z", "V"), 3)
```

Capítulo VII

Números aleatorios y semillas

Los generadores de números aleatorios hacen lo que indica su nombre: generan números de forma aleatoria cada vez que se ejecutan. No obstante, hay veces en los que se buscan números aleatorios, pero también permitir la reproducción del código. En esos casos, se emplean semillas. En R, la forma más sencilla de fijar una semilla es con `set.seed()`.

Capítulo VIII

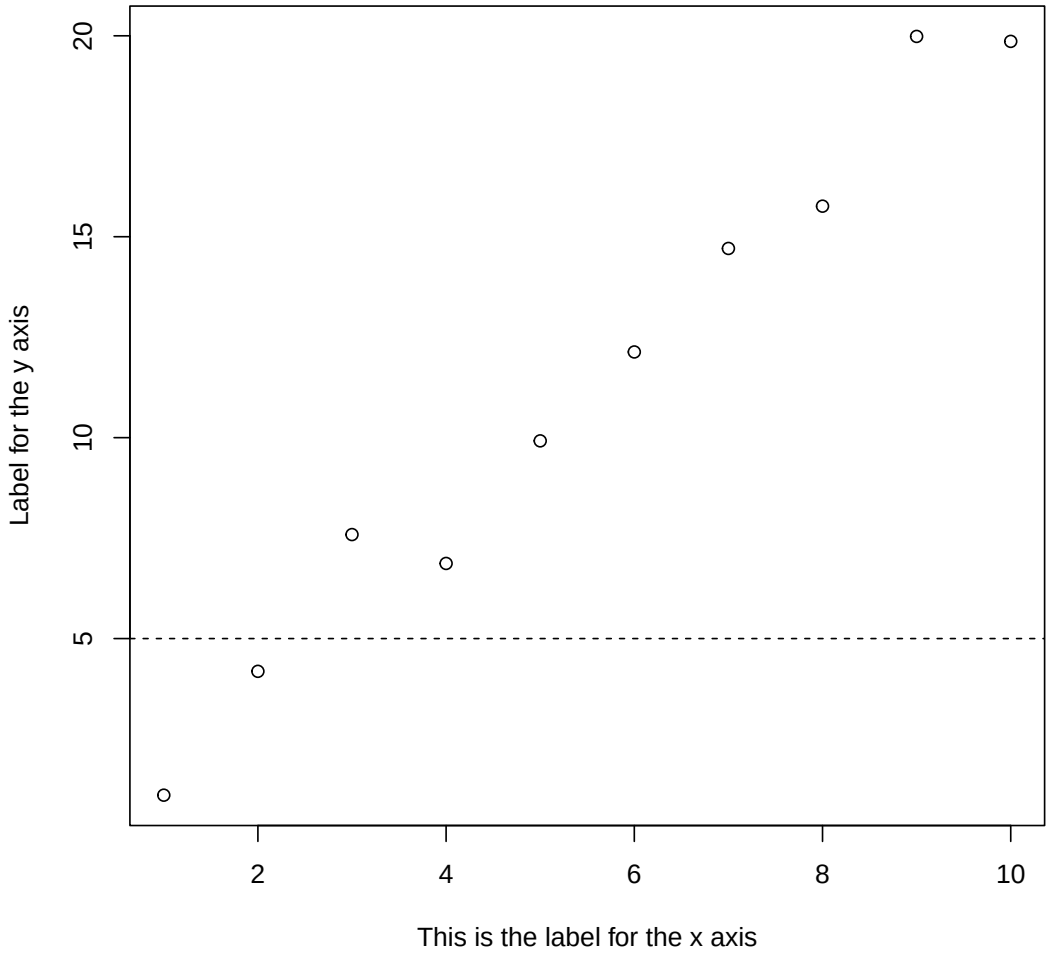
Plots (gráficos)

R puede producir una variedad de gráficos y se pueden modificar al gusto.

VIII.1. Lo más básico

La función de gráficos básica es `plot`. Su página de ayuda puede ser ligeramente engañosa y muchos argumentos adicionales se explican en `par`. Una buena analogía para empezar es la de un lienzo en el que se van añadiendo elementos. Veamos este sencillo ejemplo:

```
set.seed(2) ## for reproducibility
x <- 1:10
y <- 2 * x + rnorm(length(x))
plot(x, y, xlab = "This is the label for the x axis",
     ylab = "Label for the y axis")
## And now, we add a horizontal line:
abline(h = 5, lty = 2)
```



VIII.2. Personalización de plots: colores, tipos de línea y de puntos

Se pueden personalizar los gráficos añadiendo colores específicos, modificando el tipo de línea y de puntos.

```
plot(c(1, 21), c(1, 2.3),
     type = "n", axes = FALSE, ann = FALSE)
## show pch
points(1:20, rep(1, 20), pch = 1:20)
text(1:20, 1.2, labels = 1:20)
text(11, 1.5, "pch", cex = 1.3)

## show colors for rainbow palette
points(1:20, rep(2, 20), pch = 16, col = rainbow(20))
```

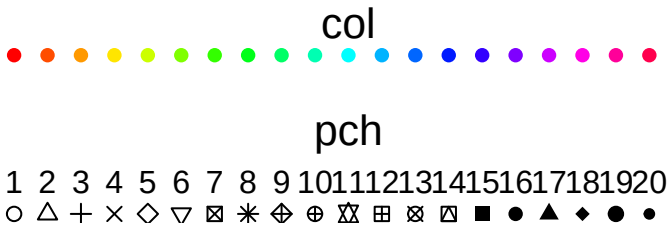


Figura VIII.1: *pch and col*

```
text(11, 2.2, "col", cex = 1.3)
```

```
plot(c(0.2, 5), c(0.2, 5), type = "n", ann = FALSE, axes = FALSE)
for(i in 1:6) {
    abline(0, i/3, lty = i, lwd = 2)
    text(2, 2 * (i/3), labels = i, pos = 3)
}
```

VIII.2.1. Un ejemplo de cómo mejorar gráficos

El gráfico básico sería el siguiente:

```
hit <- read.table("data/hit-table-500-text.txt")
## We know, from the header of the file, that
## alignment length is the fifth column,
## score is the 13th and percent identity the 3rd
hist(hit[, 5]) ## the histogram
```

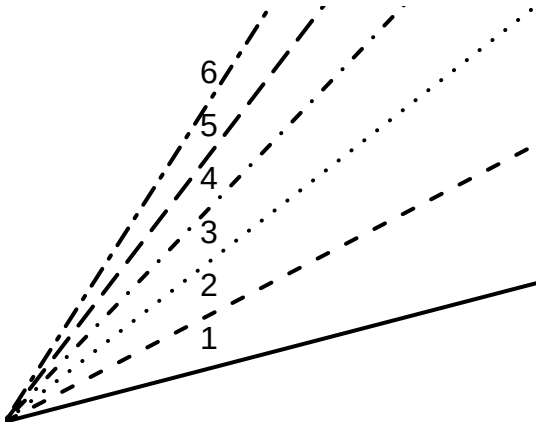
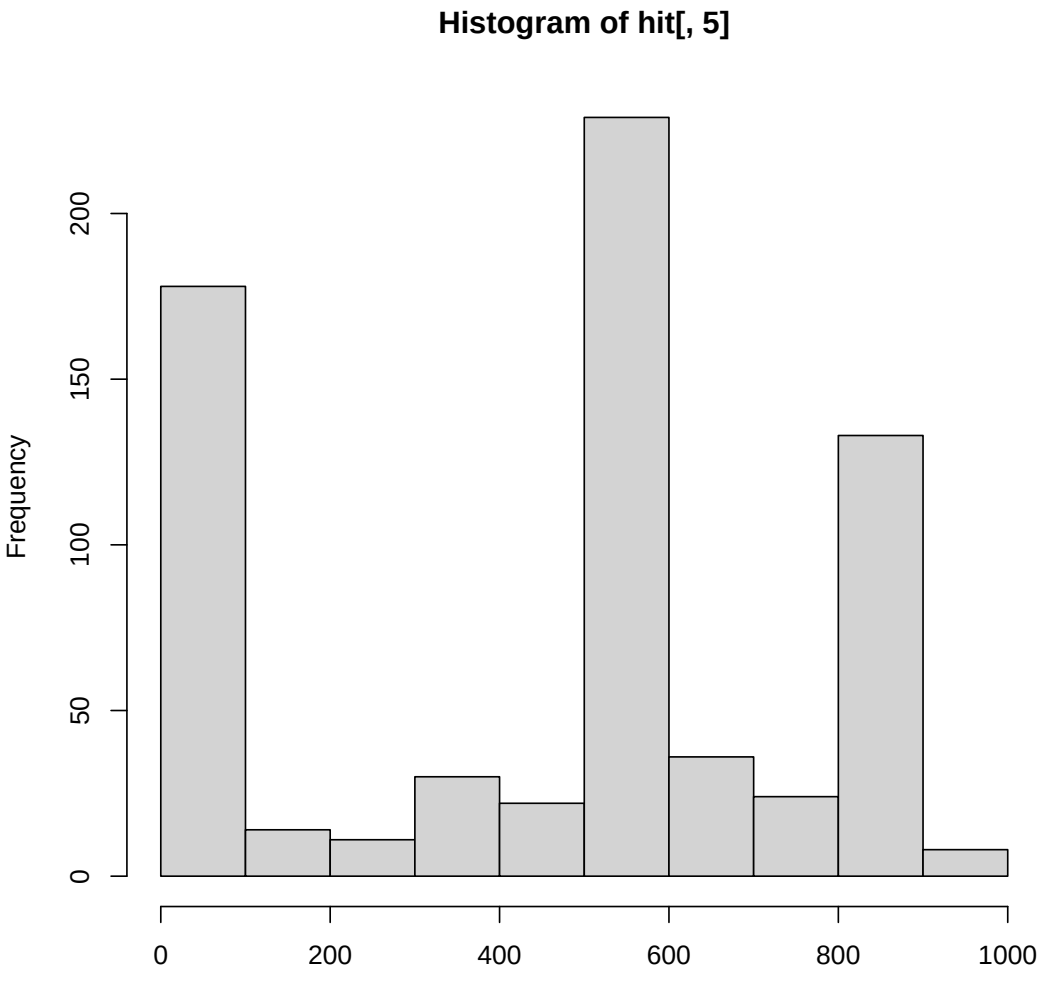
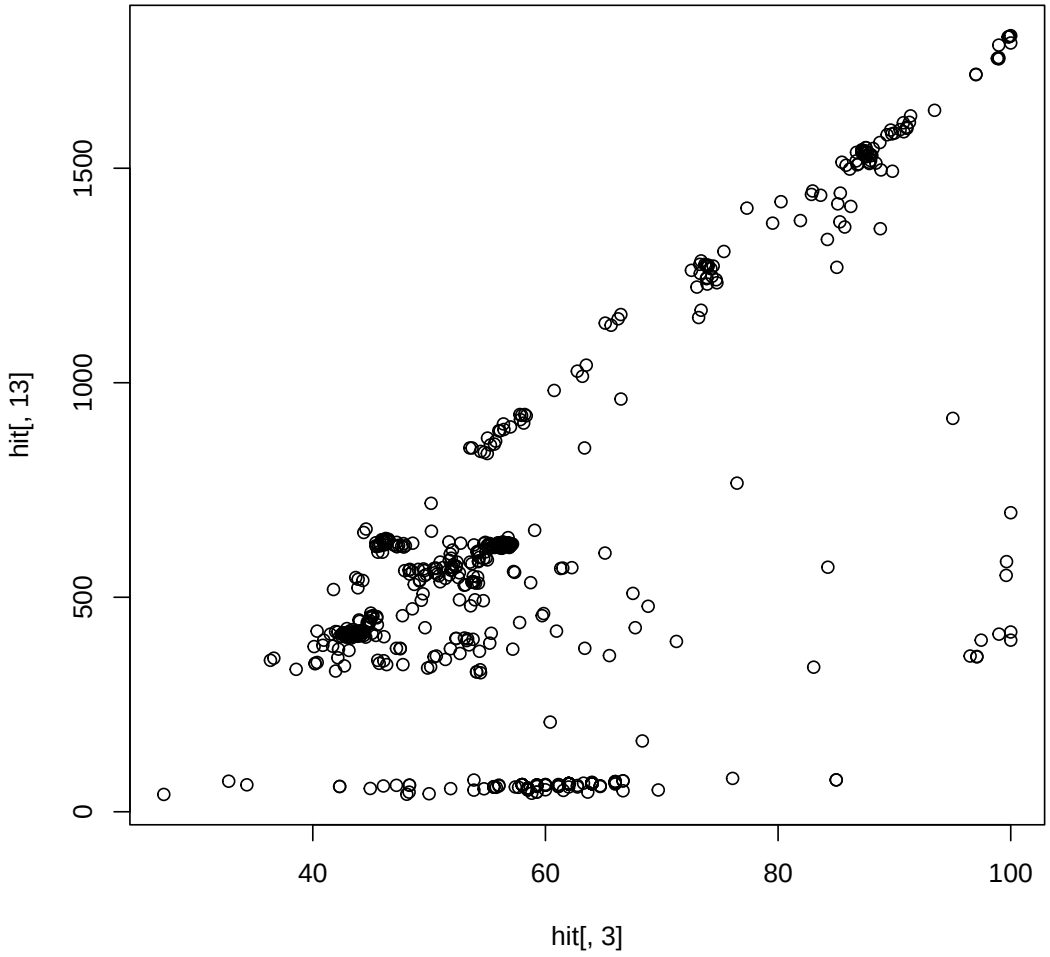


Figura VIII.2: *lty for values 1 to 6*



```
plot(hit[, 13] ~ hit[, 3]) ## the scatterplot
```




```
## plot(y ~ x) == plot(x, y)
```

Pero esto es fácilmente mejorable:

```
par(mfrow = c(1, 2)) ## two figures side by side
hist(hit[, 5], breaks = 50, xlab = "", main = "Alignment length")
plot(hit[, 13] ~ hit[, 3], xlab = "Percent. identity",
     ylab = "Bit score")
```

Por simetría, se podría añadir un título al segundo gráfico. También se pueden generar gráficos interactivos con la librería `car`.

```
library(car)
scatter3d(hit[, 13] ~ hit[, 3] + hit[, 5], xlab = "Ident",
          zlab = "Length", ylab = "Score")
```

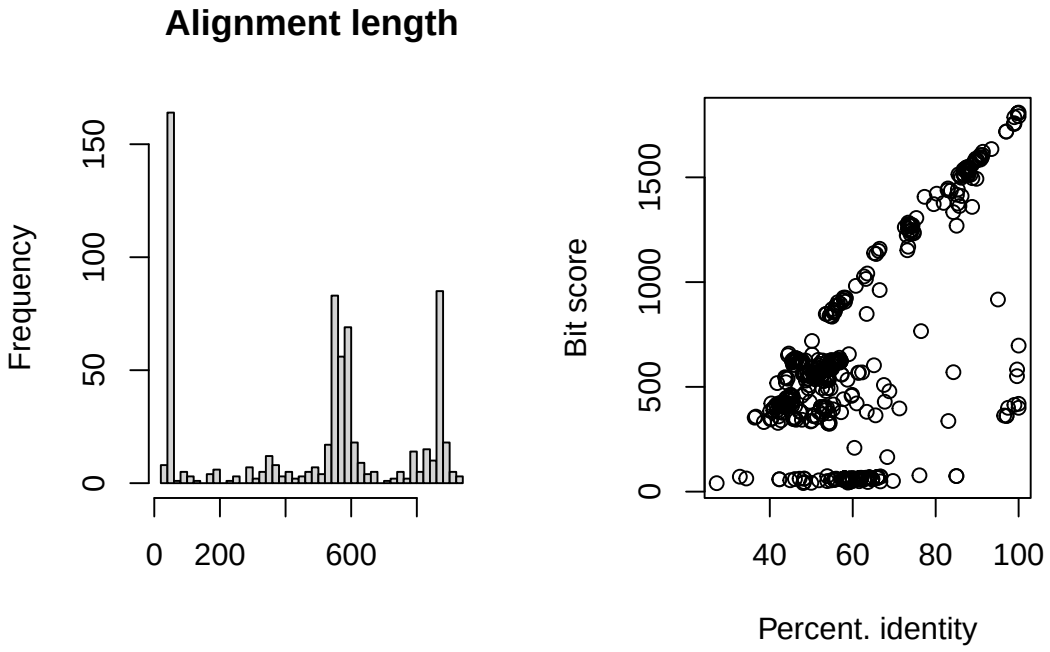


Figura VIII.3: *A quick look at the alignment results*

VIII.3. Guardar plots

Se pueden guardar las gráficas como PDF, png, etc. Desde RStudio hay una ventana de gráficos. Sin embargo, es mejor especificar y determinar unas características como tamaño, extensión, etc. Se pueden utilizar las funciones `pdf()` y `png()`: `pdf(file="plot.pdf")`. El paquete ggplot tiene la función `ggsave()`.

En el siguiente ejemplo se abre un PDF, se generan dos gráficos y hasta que no se ejecuta `dev.off()` no se guarda el contenido en el PDF. Además, cada gráfico se guarda en una página distinta del PDF.

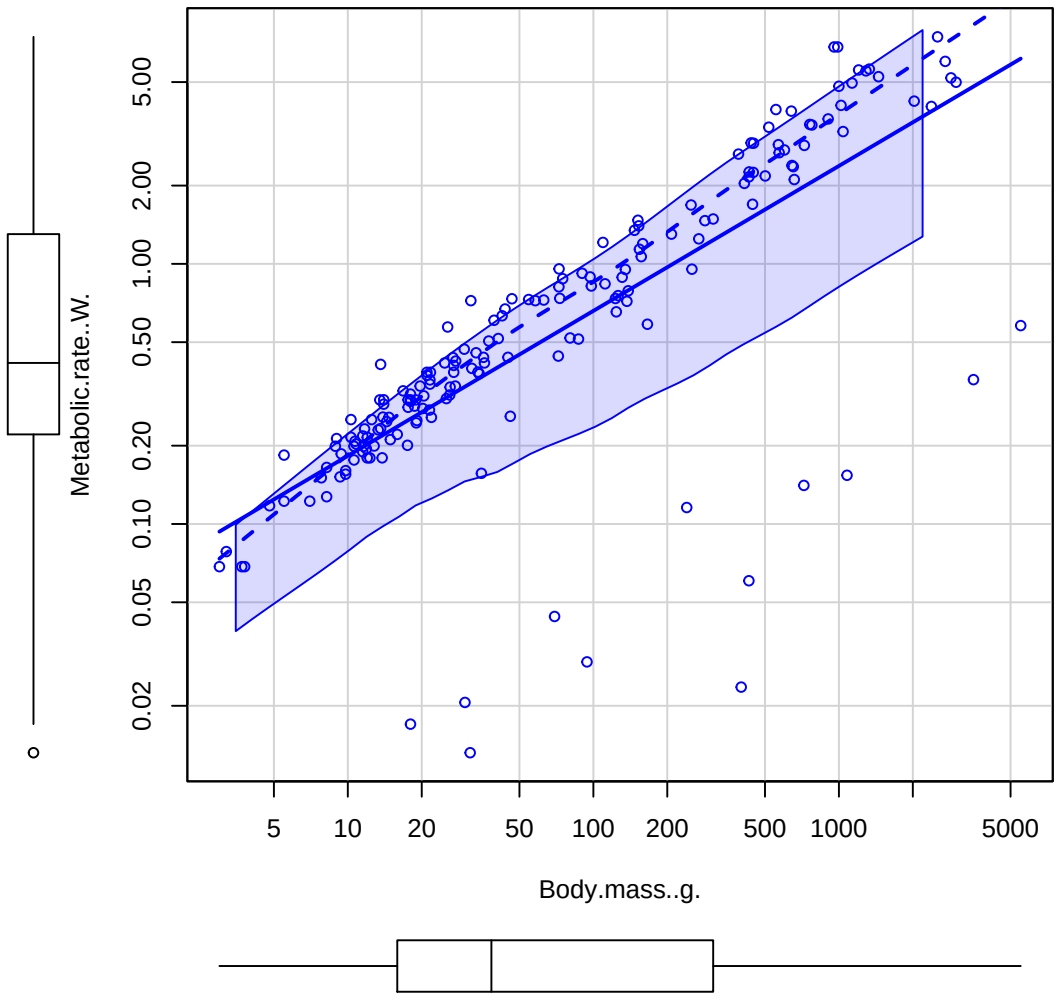
```
pdf(file = "file1.pdf", width = 2, heigh = 3)
plot(1:10)
abline(h = 4, lty = 2, col = "blue")
hist(rnorm(25))
dev.off()
```

VIII.4. Tipos de gráficos

Hemos visto que `plot` genera un gráfico simple de puntos, pero hay más tipos. Por ejemplo, `hist` genera un histograma. En el paquete de `ggplot2` hay más opciones de tipos de gráficos con una mayor posibilidad de personalización.

El paquete `car` también cuenta con varios tipos de gráficos, como scatter3d mencionado anteriormente. También tiene una función llamada `scatterplot`:

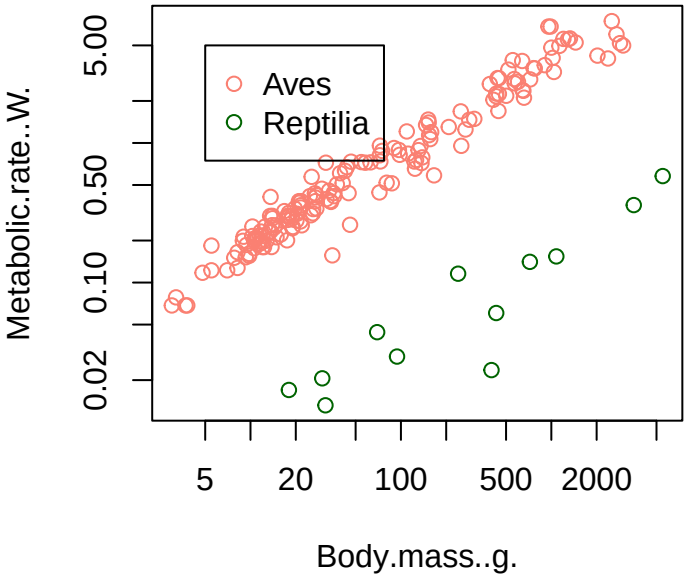
```
library(car)

## Cargando paquete requerido: carData

load("data/anage.RData")
scatterplot(Metabolic.rate..W. ~ Body.mass..g., log="xy",
            data = anage)
```



Dependiendo de la figura final que se quiera, se puede ir añadiendo elementos desde plot o utilizar scatterplot y eliminar cosas.

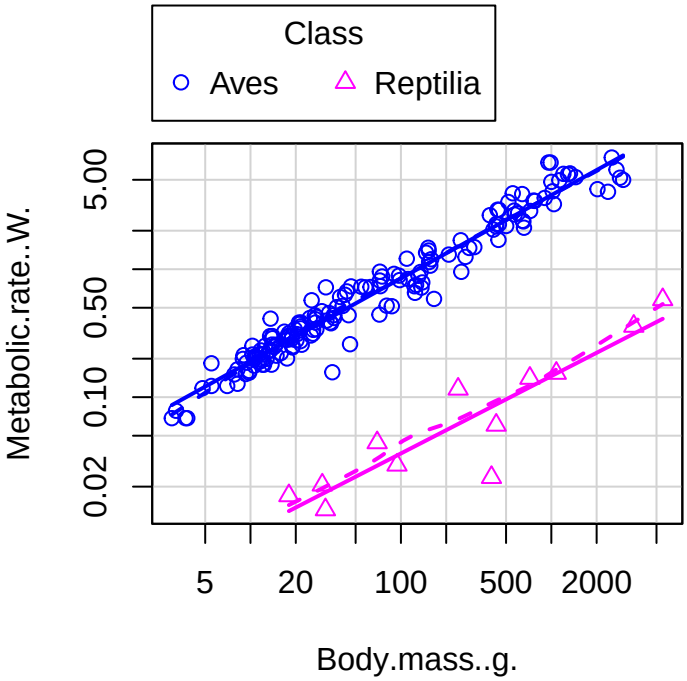
Las leyendas se introducen con la función `legend` en caso de utilizar plot.

```
plot(Metabolic.rate..W. ~ Body.mass..g., log="xy",
     col = c("salmon", "darkgreen")[Class], data = anage)
legend(5, 5, legend = levels(anage$Class),
       col = c("salmon", "darkgreen"),
       pch = 1)
```



Si se utiliza scatterplot, la sintaxis es diferente y añade directamente la línea de regresión:

```
scatterplot(Metabolic.rate..W. ~ Body.mass..g.|Class, log="xy",
            data = anage)
```



Capítulo IX

Tablas

Tabular datos es una operación muy común. Hay fundamentalmente dos formas de realizarlo con `table` (la más sencilla) y `xtabs` (con uso más genérico):

```
table(AB2$Sex, AB2$status)

##
##     V Z
##   F 1 2
##   M 2 1

with(AB2, table(Sex, status)) ## note "with"

##    status
## Sex V Z
##   F 1 2
##   M 2 1

xtabs( ~ Sex + status, data = AB2)

##    status
## Sex V Z
##   F 1 2
##   M 2 1
```

Tabular un dataframe completo saca varias tablas 2x2 en función de las combinaciones de los valores de las otras variables.

IX.1. Más de dos dimensiones y `ftable`

Cuando hay más de dos dimensiones, utilizar las funciones anteriores saca el mismo resultado.

```r
(x <-  data.frame(a = c(1,2,2,1,2,2,1),
                  b = c(1,2,2,1,1,2,1),
                  c = c(1,1,2,1,2,2,1)))
```

```
##   a b c
## 1 1 1 1
## 2 2 2 1
## 3 2 2 2
## 4 1 1 1
## 5 2 1 2
## 6 2 2 2
## 7 1 1 1
```

```r
## Equivalent
table(x)
```

```
## , , c = 1
##
##    b
## a   1 2
##   1 3 0
##   2 0 1
##
## , , c = 2
##
##    b
## a   1 2
##   1 0 0
##   2 1 2
```

```r
xtabs(~ a + b + c, data = x)
```

```
## , , c = 1
##
##    b
## a   1 2
##   1 3 0
##   2 0 1
##
## , , c = 2
##
##    b
## a   1 2
##   1 0 0
##   2 1 2
```

Sin embargo, hay veces en las que buscamos una tabla plana, es decir, encajar una de las variables dentro de otra:

```
ftable(xtabs(~ a + b + c, data = x))

##     c 1 2
## a b
## 1 1   3 0
##   2   0 0
## 2 1   0 1
##   2   1 2

## same as ftable(table(x))
```

IX.2. Recuperar una tabla de un dataframe

Si una tabla se nos ha convertido en un dataframe, se puede volver a convertir en tabla:

```
## create a data frame with a "Freq" column:
## put the table in a data frame
(dfx <- as.data.frame(table(x)))

##   a b c Freq
## 1 1 1 1    3
## 2 2 1 1    0
## 3 1 2 1    0
## 4 2 2 1    1
## 5 1 1 2    0
## 6 2 1 2    1
## 7 1 2 2    0
## 8 2 2 2    2

## We can recover the table
xtabs(Freq ~ a + b + c, data = dfx)

## , , c = 1
##
##    b
## a   1 2
##   1 3 0
##   2 0 1
##
## , , c = 2
```

```
##
##    b
## a   1 2
##   1 0 0
##   2 1 2

## of course, this is the same as
## xtabs(~ a + b + c, data = x)
## or table(x)
```

Capítulo X

La familia `apply`

Una de las grandes ventajas de R es poder operar sobre vectores, arrays, listas, etc enteros. Algunas funciones son `apply`, `lapply`, `sapply`, `tapply`, `mapply`.

X.1. apply

apply es una forma de utilizar una función sobre una cierta cantidad de elementos. Es una forma más elegante que realizando un bucle.

```
(Z <- matrix(c(1, 27, 23, 13), nrow = 2))

##      [,1] [,2]
## [1,]    1   23
## [2,]   27   13

apply(Z, 1, median)

## [1] 12 20

apply(Z, 2, median)

## [1] 14 18

apply(Z, 2, min)

## [1]  1 13
```

X.2. lapply

Con listas se utiliza `lapply`, y aplica una función definida a esa lista.

```
(listA <- list(one.vector = 1:10,  hello = "Hola",
               one.matrix = matrix(rnorm(20), ncol = 5),
               another.list = list(a = 5,
                 b = factor(c("male", "female", "female")))))

## $one.vector
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $hello
## [1] "Hola"
##
## $one.matrix
##            [,1]        [,2]       [,3]         [,4]       [,5]
## [1,]  0.4176508  1.78222896  1.0128287  1.589638200  0.4772373
## [2,]  0.9817528 -2.31106908  0.4322652  1.954651642 -0.5965582
## [3,] -0.3926954  0.87860458  2.0908192  0.004937777  0.7922033
## [4,] -1.0396690  0.03580672 -1.1999258 -2.451706388  0.2896367
##
## $another.list
## $another.list$a
## [1] 5
##
## $another.list$b
## [1] male   female female
## Levels: female male

lapply(listA, function(x) x[[1]])

## $one.vector
## [1] 1
##
## $hello
## [1] "Hola"
##
## $one.matrix
## [1] 0.4176508
##
## $another.list
## [1] 5
```

X.3. tapply y by

Se utiliza `tapply` cuando tenemos unos datos (una columna o varias) que se pueden utilizar para estratificar o seleccionar otros datos. Los dos vectores u objetos deben tener la misma longitud.

```
(one.dataframe <- data.frame(age = c(12, 13, 16, 25, 28),
                             sex = factor(c("male", "female",
                                            "female", "male", "male")))
)
```

```
##   age    sex
## 1  12   male
## 2  13 female
## 3  16 female
## 4  25   male
## 5  28   male
```

```
one.dataframe <- rbind(one.dataframe, one.dataframe)
one.dataframe$age[6:10] <- one.dataframe$age[6:10] + 2
one.dataframe$country <- rep(c("A", "B"), c(5, 5))
one.dataframe$Y <- rnorm(10)
one.dataframe
```

```
##    age    sex country          Y
## 1   12   male       A  0.7389386
## 2   13 female       A  0.3189604
## 3   16 female       A  1.0761644
## 4   25   male       A -0.2841577
## 5   28   male       A -0.7766753
## 6   14   male       B -0.5956605
## 7   15 female       B -1.7259798
## 8   18 female       B -0.9025845
## 9   27   male       B -0.5590619
## 10  30   male       B -0.2465126
```

```
tapply(one.dataframe$age, one.dataframe$sex, mean)
```

```
##   female     male
## 15.50000 22.66667
```

Se pueden formar grupos con dos o más variables siempre y cuando se proporcionen en forma de lista:

```
tapply(one.dataframe$age,
       list(one.dataframe$sex, one.dataframe$country),
       mean)
```

```
##                A        B
## female 14.50000 16.50000
## male   21.66667 23.66667
```

De igual forma, se puede utilizar una función que devuelva más que un solo valor:

```
tapply(one.dataframe$age,
       one.dataframe$sex,
       function(x) c(Mean = mean(x), Var = var(x)))

## $female
##      Mean       Var
## 15.500000  4.333333
##
## $male
##     Mean      Var
## 22.66667 59.06667
```

El problema con esto viene cuando se quiere realizar lo anterior en base a dos o más variables. Para estos casos, se debería emplear `aggregate`.

La función `by` es similar a `tapply`, pero para dataframes. Las salidas de ambas funciones también son diferentes:

```
by(one.dataframe,
   list(one.dataframe$sex, one.dataframe$country),
   function(x) c(Mean_Age = mean(x$age), SD_Age = sd(x$age),
                 Median_Y = median(x$Y)))

## : female
## : A
##   Mean_Age     SD_Age   Median_Y
## 14.5000000  2.1213203  0.6975624
## ------------------------------------------------
## : male
## : A
##   Mean_Age     SD_Age   Median_Y
## 21.6666667  8.5049005 -0.2841577
## ------------------------------------------------
## : female
## : B
##  Mean_Age    SD_Age  Median_Y
## 16.500000  2.121320 -1.314282
## ------------------------------------------------
## : male
## : B
##   Mean_Age     SD_Age   Median_Y
## 23.6666667  8.5049005 -0.5590619
```

X.4. aggregate

La función aggregate suele devolver la salida en un formato más conveniente. En este caso, el segundo argumento debe ser siempre una lista:

```
aggregate(one.dataframe$age, list(one.dataframe$sex), mean)

##   Group.1        x
## 1  female 15.50000
## 2    male 22.66667

## make the aggregating variable explicit,
## and give it another name
aggregate(one.dataframe$age,
          list(Sexo = one.dataframe$sex), mean)

##     Sexo        x
## 1 female 15.50000
## 2   male 22.66667

## or use the name of the column/variable
aggregate(one.dataframe$age,
          one.dataframe[2], mean)

##      sex        x
## 1 female 15.50000
## 2   male 22.66667
```

Se puede utilizar con dos o más variables:

```
aggregate(one.dataframe$age,
          list(Sex = one.dataframe$sex,
               Country = one.dataframe$country), mean)

##      Sex Country        x
## 1 female       A 14.50000
## 2   male       A 21.66667
## 3 female       B 16.50000
## 4   male       B 23.66667
```

También se puede utilizar para devolver varios valores:

```
aggregate(one.dataframe$age,
          list(Sex = one.dataframe$sex,
               Country = one.dataframe$country),
          function(x) c(Mean = mean(x), SD = sd(x))
          )
```

```
##      Sex Country    x.Mean      x.SD
## 1 female       A 14.500000  2.121320
## 2   male       A 21.666667  8.504901
## 3 female       B 16.500000  2.121320
## 4   male       B 23.666667  8.504901
```

aggregate también se puede llamar con una sintaxis de tipo fórmula, que puede ser más intuitiva:

```
aggregate(age ~ sex + country, data = one.dataframe,
          function(x) c(Mean = mean(x), SD = sd(x)))
```

```
##      sex country  age.Mean    age.SD
## 1 female       A 14.500000  2.121320
## 2   male       A 21.666667  8.504901
## 3 female       B 16.500000  2.121320
## 4   male       B 23.666667  8.504901
```

Esto también funciona para funciones con múltiples columnas o vectores:

```
(ag1 <- aggregate(cbind(age, Y) ~ sex + country,
                  data = one.dataframe,
                  function(x) c(Mean = mean(x), SD = sd(x))))
```

```
##      sex country  age.Mean    age.SD     Y.Mean       Y.SD
## 1 female       A 14.500000  2.121320  0.6975624  0.5354240
## 2   male       A 21.666667  8.504901 -0.1072981  0.7731305
## 3 female       B 16.500000  2.121320 -1.3142821  0.5822284
## 4   male       B 23.666667  8.504901 -0.4670783  0.1918901
```

```
aggregate(one.dataframe[, c("age", "Y")],
          list(Sex = one.dataframe$sex,
               Country = one.dataframe$country),
          function(x) c(Mean = mean(x), SD = sd(x)))
```

```
##      Sex Country  age.Mean    age.SD     Y.Mean       Y.SD
## 1 female       A 14.500000  2.121320  0.6975624  0.5354240
## 2   male       A 21.666667  8.504901 -0.1072981  0.7731305
## 3 female       B 16.500000  2.121320 -1.3142821  0.5822284
## 4   male       B 23.666667  8.504901 -0.4670783  0.1918901
```

Es importante mencionar que el resultado no es un dataframe de 6 columnas, si no de 4: Mean y SD forman una matriz de dos columnas dentro de una misma columna. Para que el dataframe de salida sí tenga las 6 columnas, se puede utilizar `do.call`:

```
do.call(data.frame,
        aggregate(cbind(age, Y) ~ sex + country,
                  data = one.dataframe,
                  function(x) c(Mean = mean(x), SD = sd(x)))
        )

##      sex country age.Mean   age.SD     Y.Mean      Y.SD
## 1 female       A 14.50000 2.121320  0.6975624 0.5354240
## 2   male       A 21.66667 8.504901 -0.1072981 0.7731305
## 3 female       B 16.50000 2.121320 -1.3142821 0.5822284
## 4   male       B 23.66667 8.504901 -0.4670783 0.1918901
```

X.5. split

La función `split` sirve para dividir un dataframe en varios en función de una variable

```
split(one.dataframe, one.dataframe$sex)

## $female
##   age    sex country          Y
## 2  13 female       A  0.3189604
## 3  16 female       A  1.0761644
## 7  15 female       B -1.7259798
## 8  18 female       B -0.9025845
##
## $male
##    age  sex country          Y
## 1   12 male       A  0.7389386
## 4   25 male       A -0.2841577
## 5   28 male       A -0.7766753
## 6   14 male       B -0.5956605
## 9   27 male       B -0.5590619
## 10  30 male       B -0.2465126

split(one.dataframe, c(one.dataframe$sex, one.dataframe$country))

## Warning in split.default(x = seq_len(nrow(x)), f = f, drop = drop,
...): largo de datos no es múltiplo de la variable de separación

## $`1`
##   age    sex country          Y
## 2  13 female       A  0.3189604
## 3  16 female       A  1.0761644
## 7  15 female       B -1.7259798
```

```
## 8  18 female       B -0.9025845
##
## $`2`
##    age  sex country          Y
## 1   12 male       A  0.7389386
## 4   25 male       A -0.2841577
## 5   28 male       A -0.7766753
## 6   14 male       B -0.5956605
## 9   27 male       B -0.5590619
## 10  30 male       B -0.2465126
##
## $A
## [1] age     sex     country Y
## <0 rows> (o 0- extensión row.names)
##
## $B
## [1] age     sex     country Y
## <0 rows> (o 0- extensión row.names)
```

Esto se puede combinar con *apply:

```
lapply(split(one.dataframe,
             list(one.dataframe$sex,
                  one.dataframe$country)),
       function(x) lm(Y ~ age, data = x)) #or lm(x$Y ~ x$age)

## $female.A
##
## Call:
## lm(formula = Y ~ age, data = x)
##
## Coefficients:
## (Intercept)          age
##     -2.9623       0.2524
##
##
## $male.A
##
## Call:
## lm(formula = Y ~ age, data = x)
##
## Coefficients:
## (Intercept)          age
##     1.84109     -0.08993
##
##
## $female.B
```

```
##
## Call:
## lm(formula = Y ~ age, data = x)
##
## Coefficients:
## (Intercept)          age
##     -5.8430       0.2745
##
##
## $male.B
##
## Call:
## lm(formula = Y ~ age, data = x)
##
## Coefficients:
## (Intercept)          age
##    -0.84879      0.01613
```

El procedimiento anterior está relacionado con los enfoques split-apply-combine y map-reduce. Y `by`, `aggregate`, y amigos pueden ser considerados como formas especialmente prácticas de hacer la combinación anterior de `split` con `*apply` y alguna(s) función(es) de resumen particular(es).

X.6. apply y dejar caer dimensiones en matrices

A menos que utilicemos `drop = FALSE`, si seleccionamos sólo una fila o una columna, el resultado no es una matriz, sino un vector. Pero a veces necesitamos que permanezcan como matrices. Ese es a menudo el caso en muchas operaciones matriciales, y también cuando se utiliza `apply` y afines.

```
(E <- matrix(1:9, nrow = 3))
```

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

```
E[, 1]
```

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

```
E[, 1, drop = FALSE]
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
```

```r
E[1, ]

## [1] 1 4 7

E[1, , drop = FALSE]

##      [,1] [,2] [,3]
## [1,]    1    4    7
```

Esto suele ser importante cuando se escribe código genérico y una variable solo tenga una dimensión.

X.7. Algunas apreciaciones

Hay otros tipos de `apply` que veremos más adelante, tales como `vapply`, `sapply`, `mapply`. Además, las operaciones con `apply` son fácilmente paralelizables (librería `parallel`).

Capítulo XI

Programación en R

XI.1. Flow control

R tiene las típicas construcciones condicionales y estructuras de control: `if`, `ifelse`, `for`, `while`, `repeat`, `switch`, `break`. Un `for` loop rara vez es la opción adecuada, normalmente es mejor utilizar `apply`.

```
names.of.friends <- c("Ana", "Rebeca", "Marta",
                      "Quique", "Virgilio")
for(friend in names.of.friends) {
  cat("\n I should call", friend, "\n")
}
```

```
##
##  I should call Ana
##
##  I should call Rebeca
##
##  I should call Marta
##
##  I should call Quique
##
##  I should call Virgilio
```

```
x <- y <- 0
iteration <- 1
while( (x < 10) && (y < 2)) {
  cat(" ... iteration", iteration, "\n")
  x <- x + runif(1)
  y <- y + rnorm(1)
  iteration <- iteration + 1
}
```

```
##  ... iteration 1
```

```
##  ... iteration 2
##  ... iteration 3
##  ... iteration 4
##  ... iteration 5
##  ... iteration 6
##  ... iteration 7
##  ... iteration 8

x

## [1] 3.183051

y

## [1] 3.020543
```

`while` normalmente se combina con `break` para salidr del bucle en cuanto pasa algo (normalmente detectado mediante `if`). Break sirve para salir del bloque de llaves del bucle en el que está metido, no para todo.

```
iteration <- 0
while(TRUE) {
  iteration <- iteration + 1
  cat(" ... iteration", iteration, "\n")
  x <- rnorm(1, mean = 5)
  y <- rnorm(1, mean = 7)
  z <- x * y
  if (z < 15) break
}
```

```
aa <- 9

if (aa < 95) {
  cat("\n aa is < 95\n")
} else if (aa > 100) {
  cat("\n hummm.... larger than a 100\n")
} else {
  cat("\n between 95 and a 100\n")
}

##
##  aa is < 95
```

XI.2. Definir funciones

Se pueden crear funciones en R mediante `function`:

```
multByTwo <- function(x) {
  z <- 2 * x
  return(z)
}

a <- 3
multByTwo(a)

## [1] 6

multByTwo(45)

## [1] 90
```

Si no se incluye `return`, la función devuelve el último valor generado, pero es recomendable añadirlo para facilitar la lectura.

Las funciones pueden tener varios argumentos, y es posible que tengan valores por defecto:

```
plotAndLm <- function(x, y, title = "A figure") {
  lm1 <- lm(y ~ x)
  cat("\n Printing the summary of x\n")
  print(summary(x))
  cat("\n Printing the summary of y\n")
  print(summary(y))
  cat("\n Printing the summary of the linear regression\n")
  print(summary(lm1))
  plot(y ~ x, main = title)
  abline(lm1)
  return(lm1)
}

x <- 1:20
y <- 5 + 3 *x + rnorm(20, sd = 3)
plotAndLm(x, y)
plotAndLm(x, y, title = "A user specified title")
```

XI.3. Orden de los argumentos, argumentos con y sin nombre

R es bastante flexible a la hora de llamar a una función y el orden en el que se pasan los argumentos, pero hay formas mejores y peores de hacerlo. En general, se utiliza la posición de llamada solo para los primeros dos argumentos, y se recomienda evitar pasar argumentos sin nombre después de haber nombrado a algunos:

```
f1 <- function(one, two, three) {
    cat("one = ", one,
        " two = ", two,
        " three = ", three, "\n")}

## We are OK
f1(1, 2, 3)

## one =  1  two =  2  three =  3

## We are OK, but this is getting risky
f1(two = 2, three = 3, 1)

## one =  1  two =  2  three =  3

## We are no longer OK. Nothing "strange" happened
## but we would need to be very careful. So avoid it.
f1(two = 2, 3, 1)

## one =  3  two =  2  three =  1
```

XI.4. Scoping, frames y entornos

R puede tener variables globales y locales.

```
f1 <- function(x) {
    x + z
}

z <- -100 #variable global

f11 <- function(y) {
    z <- 10 #variable local
    f1(y)
}
```

```
f11(4)


## [1] -96
```

En este caso, z podría adquirir el valor donde se definió f1 (el entorno global) o usando el valor del entorno local en el que se llamó a f1. R utiliza la primera opción: resuelve donde se definió f1, tomando el valor de z de ese entorno. Esto es igual en otros lenguajes como Python.

Este es otro ejemplo en el que, como se define una función dentro de otra, al llamarla hereda los valores de las variables del entorno local.

```
v <- 1000
f3 <- function(x, y) {
    v <- 3 * x
    f2 <- function(u) {
        u + v
    }
    f2(y)
}

f3(2, 9)
```

binding En `y <- 9`, y está unida al valor 9.

free variable z es una variable libre en la función f1 de arriba. No está unida a nada (al menos en ese frame)

frame Una serie de bindings (y a 9, x a 77, etc.).

environment Puedes pensar en ello como una secuencia de frames Cuando f2 (bueno, R) busque el valor de v lo hará a través de una secuencia de frames De hecho, un entorno tiene dos componentes: un frame y una referencia a otro entorno, su entorno padre (o su entorno adyacente); puesto que cada entorno tiene una referencia a otro entorno, ahora puedes entender fácilmente la idea de un entorno como una secuencia de frames

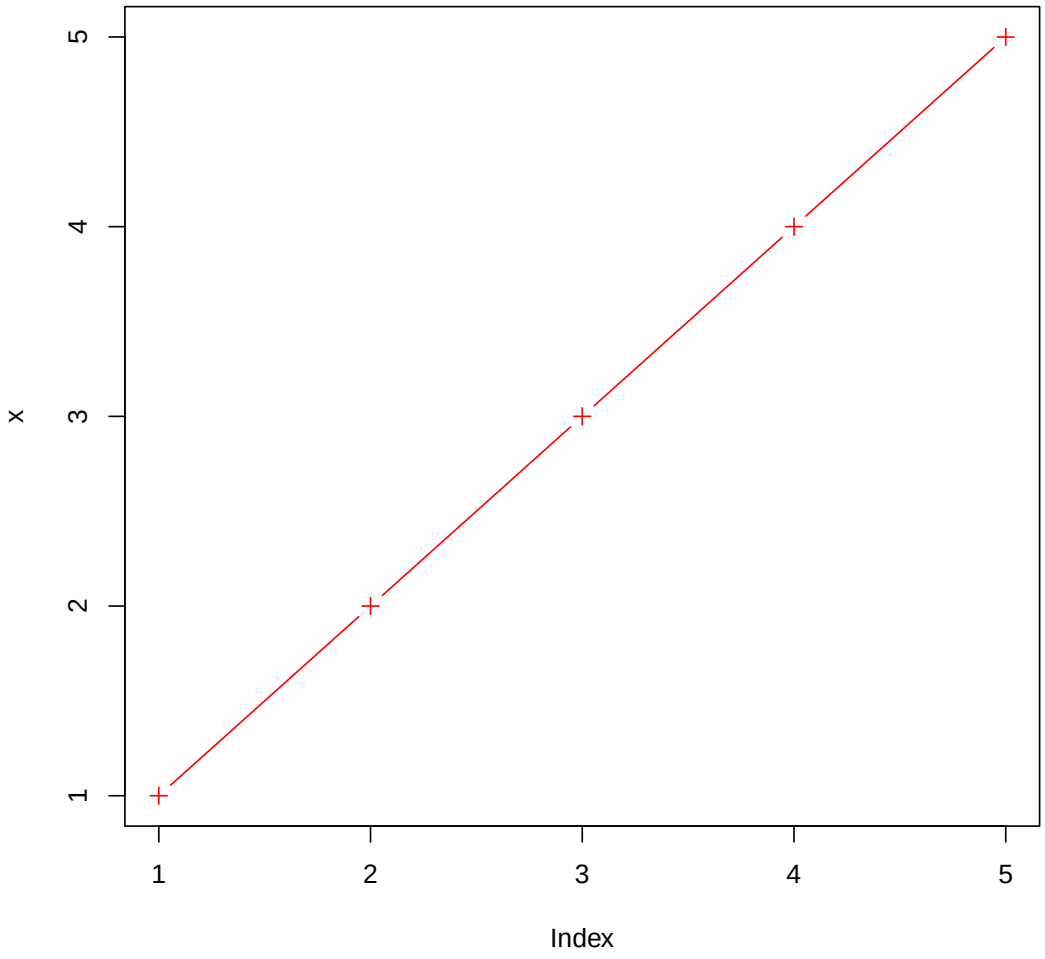
```
search()
```

Esto se utiliza implícitamente o explícitamente en gran parte del código. Lo que hace es listar los distintos entornos que hay y su orden. Así, cuando se cargan librerías o se utilizan variables, se utiliza `search` para localizar lo que se está pidiendo.

XI.5. Los ...

Los ... permiten pasar argumentos adicionales en funciones que deben manejarlas.

```r
f0 <- function(x, pch = 3, ...) {plot(x, pch = pch, ...)}

f0(1:5, col = "red", type = "b")
```



Aquí, col y type que no se han especificado en f0, se pasan directamente a la función. Como plot acepta muchos argumentos adicionales, no es necesario especificar todos, si no que se pueden poner ....

```r
fa <- function(x, col = "red", ...) {plot(x, col = col, ...)}
fa(1:5, "blue", pch = 8)

fb <- function(x, col = "red", ...) {plot(x, col = col)}
fb(1:5, "blue", pch = 8)

fc <- function(x, col = "red") {plot(x, col = col, ...)}
fc(1:5, "blue", pch = 8)
```

La función fa recoge pch dentro de los tres puntos. Además, el color se sobrescribe (el plot resultante tiene los puntos en azul). Tanto fb como fc no hacen lo que se espera, ya que les falta ...en la función y en el plot respectivamente. Esa ausencia hace que las funciones no hagan nada con ese argumento.

XI.6. local

La función `local` permite crear un entorno local en el que trabajar y luego, al salir, que no tenga guardado el valor de las variables.

```
i <- 2
local({cat("i ", i); i <- 99; cat(";  i = ", i)})

## i  2;  i =  99

i

## [1] 2

try(rm(vv))
local({vv <- 99; cat("vv = ", vv)})

## vv =  99

try(vv)

## Error in eval(expr, envir) : objeto 'vv' no encontrado
```

XI.7. Evaluación vaga

La siguiente función toma 2 argumentos, pero solo utiliza 1. Por ello, cuando solo se le pasa un argumento, no se produce ningún error.

```
flazy <- function(x, y) {return(2 * x)}
flazy(4)

## [1] 8
```

Esto no es algo a hacer de forma habitual, pero es importante entender por qué el código no se rompe. En otras palabras, la evaluación vaga es la evaluación de los valores cuando se van a utilizar, no cuando se definen.

Capítulo XII

Debugging y capturar excepciones

Debugging consiste en recorrer el código para comprobar que funciona como se espera y no se estropee.

XII.1. `traceback`

`traceback` muestra la última llamada y ayuda a identificar dónde se rompió el código para ver qué función tiene un problema.

```
f1 <- function(x) 3 * x

f2 <- function(x) 5 + f1(x)

f3 <- function(z, u) {
    v <- runif(z)
    a <- f2(u)
    b <- f2(3 * v)
    return(a + b)
}

f3(3, 7)

## [1] 36.97035 35.67900 33.98585

f3(-5, 6)

## Error in runif(z): invalid arguments

traceback()

## No traceback available

f3(5, "a")
```

```
## Error in 3 * x: argumento no-numérico para operador binario

traceback()

## No traceback available
```

XII.2. `debug` and `browser`

El comando debug permite ir paso a paso ejecutando cada línea del código. Cuando se quiera parar, hay que poner simplemente undebug.

```
debug(f3)
f3(3, 5)
undebug(f3) ## stop debugging
f3(3, 5)
```

El comando `browser` para la ejecución de la expresión actual y permite acceder al intérprete de R. También se puede realizar de forma condicional.

```
## just browser
f3 <- function(z, u) {
    v <- runif(z)
    a <- f2(u)
    browser()
    b <- f2(3 * v)
    return(a + b)
}

## with conditional browser
f3 <- function(z, u) {
    v <- runif(z)
    if (z > 5) browser()
    a <- f2(u)
    b <- f2(3 * v)
    return(a + b)
}
```

Desde `browser`, hay una serie de expresiones:

- n o enter permite ejecutar la siguiente línea.
- c: salir del browser y continuar la ejecución del siguiente statement.
- s: evalúa el siguiente statement entrando en las siguientes funciones.
- Q: salir de la evaluación actual e ir al sitio desde donde se llamó.

debug es como poner `browser` al inicio del código.

XII.3. `trace` para ver funciones arbitrarias en sitios arbitrarios

Se puede utilizar `debug` con funciones que no hayamos escrito nosotros (por ejemplo, `debug(lm)`). Sin embargo, la función lm es muy larga y quizás no queremos empezar por arriba, si no que sospechamos que nuestros problemas están por el medio. Para eso, podemos utilizar `trace`.

```
trace("lm", edit = TRUE)
```

```
as.list(body(lm))
trace(lm, tracer = browser, at = 5)
y <- runif(100)
x <- 1:100
lm(y ~ x)
## stop tracing
untrace(lm)
```

XII.4. Warnings

En R puede haber algunas funciones que no den error, pero muestren warnings. En algunos casos, los warnings pueden indicar que algo no esté funcionando, por lo que se pueden convertir los warnings en errores para que la función no se ejecute.

```
opt <- options(warn = 2)
```

Este código hace que los warnings se comporten como errores. Una vez terminado, se puede reestablecer a los valores predeterminados mediante:

```
options(opt)
```

XII.5. `where` para cuando uno está perdido en dónde está

A veces, cuando se hace debugging, especialmente cuando se está dentro de varias funciones que se llaman unas a otras, uno puede perderse y no saber dónde está. En estos casos, se utiliza `where`, que devuelve la función en la que nos encontramos.

```
debug(f1); debug(f2); debug(f3)
f3(4, 5) ## now, keeping pressing enter or n
         ## and you'll get deeper and deeper
         ## while in browser mode, type where

#Return things to normal
undebug(f3)
undebug(f2)
undebug(f1)
```

XII.6. Protección frente a posibles fallos

Hay un manejo excepcional en R mediante `try`. Esto permite evitar que el código no falle. Cuando se va a dar un error, la variable adquiere la clase try-error.

```
ft <- function(x) {
    tmp <- try(log(x), silent = TRUE)
    if(inherits(tmp, "try-error")) {
        warning(paste("It looks like something did not work:\n",
                      "   ", tmp))
    } else{
        return(tmp)
    }
}

ft(9)

## [1] 2.197225

ft("a")

## Warning in ft("a"): It looks like something did not work:
##      Error in log(x) : Argumento no numérico para una función matemática
```

XII.7. Funciones de debugging que no son exportadas

Si cargamos un paquete, sirve con `library(paquete)`. No obstante, puede haber algunas funciones no exportadas (no se ve directamente al teclear el nombre).

```
trace(randomForest:::predict.randomForest, edit = TRUE)
```

Las **funciones exportadas** son aquellas que el paquete pone a disposición del usuario final. Esto significa que cualquier persona que cargue el paquete puede llamar a estas funciones directamente. Así, cuando la función está exportada, el usuario solo necesita escribir el nombre de la función para ejecutarla, siempre que el paquete esté cargado.

Las **funciones no exportadas** son aquellas que están presentes en el paquete pero no están pensadas para ser utilizadas por el usuario final. Estas funciones suelen ser de uso interno, y los desarrolladores del paquete las utilizan para realizar tareas auxiliares o para construir las funciones exportadas de manera modular. No aparecen en la lista de funciones del paquete y no son accesibles directamente.

Capítulo XIII

Programación orientada a objetos: clases S3 y S4

En R hay varios sistemas de programación orientada a objetos. Los sistemas originales en R son los sistemas S3 y S4, siendo los más extendidos.

XIII.1. `methods`

```
methods('plot')
getAnywhere(plot.TukeyHSD)
#stats:::plot.TukeyHSD
```

`getAnywhere` permite obtener las funciones no exportadas. `plot` realmente no hace nada, solo determina el tipo de objeto que se le ha pasado y llama a la función específica para ese objeto.

Lo que se ve con `methods` depende de los paquetes que haya cargados y, por tanto, lo que haya en nuestro espacio de búsqueda.

En POO en R, no se define una clase dentro de la que definir métodos (como en Python). Los métodos no pertenecen a la clase, si no que hay que definirlos por separado.

Se pueden buscar todos los métodos de una clase con:

```
methods(class = 'lm')
methods(class = 'lm', byclass = FALSE)
```

El argumento `byclass` muestra el nombre completo de los métodos y si están o no exportados.

El código fuente de todas las funciones está disponible. Para todas las funciones S3 exportadas desde el namespace, se puede escribir el nombre del método en la línea de comando como generic.class. Para las funciones no exportadas, se puede utilizar `getAnywhere` o `getS3method`. Sabiendo el namespace, también se puede utilizar :::.

```
add1.lm
getAnywhere('add1.lm')
stats:::add1.lm
getS3method('add1', 'lm')
```

XIII.2. Creación de clases y métodos

Vamos a suponer que queremos trabajar con unos data frames especiales con información sobre colesterol, la expresión de un gen y el tipo de experimento que lo midió. Lo primero que se quiere es convertir data frames en objetos de mi clase (y más adelante convertir matrices o vectores a objetos), crear un summary de los objetos y ajustar funciones de plot. Finalmente, hay que testear el código mediante la librería `testthat`.

Empezamos con una función genérica de conversión al objeto y luego un método que funcione para los data frames. El objeto será Cholest_Gene object, por lo que un conversor genérico (y sus comentarios) sería:

```
# object -> Cholest_Gene object
# General converter to Cholest_Gene object.
to_CG <- function(x, ...) {
  UseMethod("to_CG")
}
```

El primer método será convertir un data frame al objeto:

```
# data.frame -> Cholest_Gene object
# Take a data frame and return (if possible) a Cholest_Gene object.
to_CG.data.frame <- function(x) {
  cns <- c("Cholesterol", "Gene", "Kind")
  if (!(all(colnames(x) %in% cns)))
        stop(paste("Column names are not ", cns))
    tmp <- x[, cns]
    ## Notice I do not set this to be of data.frame class
    class(tmp) <- c("Cholest_Gene")
    return(tmp)
}
```

Y se debe probar:

```
uu <- to_CG(data.frame(Cholesterol = 1:10, Gene = 11:20, Kind = "Cl1"))
uu
```

El resultado de la visualización es muy feo y se debe mejorar. Esto se debe a que la clase es Cholest_Gene y, por ello, utiliza print default. Por ello, se debe cambiar la clase del objeto a la clase Cholest_Gene, adjuntando la clase que tenía previamente:

```r
# data.frame -> Cholest_Gene object
# Take a data frame and return (if possible) a Cholest_Gene object.
to_CG.data.frame <- function(x) {
    cns <- c("Cholesterol", "Gene", "Kind")
    if (!(all(colnames(x) %in% cns)))
        stop(paste("Column names are not ", cns))
    tmp <- x[, cns]
    tmp$Kind <- factor(tmp$Kind)
    class(tmp) <- c("Cholest_Gene", class(tmp)) ## "data.frame")
    return(tmp)
}
```

De esta forma, la visualización del objeto está bien, al igual que la salida de summary, reutilizando así las funciones existentes.

```r
uu <- to_CG(data.frame(Cholesterol = 1:10, Gene = 11:20, Kind = "Cl1"))
summary(uu)
print(uu)
uu
```

La siguiente función es más sofisticada:

```r
# Cholest_Gene object -> printed Cholest_Gene object
# Print a Cholest_Gene object.
print.Cholest_Gene <- function(x) {
    u <- x[, c(1, 2)]
    class(u) <- "data.frame"
    print(u)
    cat("\n Printing summary of first column \n")
    print(summary(x[, 1]))
}
```

En este caso, se asigna la clase data.frame (y solo esa clase) para evitar que se llame a sí mismo.

Como por el momento solo se ha creado el método para convertir data frames a nuestro objeto, se debe comprobar que el objeto que se pase sea de una clase soportada (data frame) y no se ejecute cuando la clase (por ejemplo, matriz) no está soportada. Además, esto muestra un mensaje de error personalizado.

```r
# arbitrary object -> failure message if no method
# Return error message if there is no specific method to convert
# from that class to Cholest_Gene class
to_CG.default <- function(x) {
    stop("For now, only methods for data.frame are available.")
}
```

XIII.3. Testeo y test-driven development

El último paso es el testeo, y es algo fundamental. En los tests se van poniendo casos en los que se encuentran bugs y se arreglan. Como mínimo, se debe comprobar que se puede crear un objeto legítimo de un data frame, que falla (como esperamos) cuando al data frame le faltan columnas y que falla (como esperamos) cuando no se proporciona un data frame. El testeo se puede llevar a cabo con el paquete `testthat`, el cual tiene varios bloques de salidas que se pueden esperar (`expect_s3_class`, `expect_error`, `expect_equal`, `...`)

```
library(testthat)
test_that("minimal conversions and failures", {

    expect_s3_class(to_CG(data.frame(Cholesterol = 1:10, Gene = 11:20,
                                     Kind = "Cl1")), "Cholest_Gene")


    expect_error(to_CG(cbind(Cholesterol = 1:10, Gene = 11:20)),
                 "For now, only methods for data.frame are available",
                 fixed = TRUE)

    expect_error(to_CG(data.frame(Cholesterol = 1:10, Geni = 11:20,
                                     Kind = "Cl1")),
                 "Column names are not ",
                 fixed = TRUE)

    expect_s3_class(to_CG(data.frame(Cholesterol = 1:10, Gene = 11:20,
                                     Kind = "Cl1",
                                     whatever = "abcd")),
                    "Cholest_Gene")

})
```

El último bloque de test ha fallado, por lo que hay que hacer debugging.

```
debugonce(to_CG.data.frame)

dummy <- to_CG(data.frame(Cholesterol = 1:10, Gene = 11:20,
                                  Kind = "Cl1",
                                  whatever = "abcd"))
```

En este entorno se va viendo qué va pasando en cada línea de código, y se verifica dónde está el problema. En este caso, el if comprueba si todos los nombres de columnas están en cns, cuando debería ser al revés: que todos los nombres de cns estén en el data frame (aunque haya otras columnas adicionales). Por tanto, hay que reescribir la función invirtiendo eso:

```
to_CG.data.frame <- function(x) {
    cns <- c("Cholesterol", "Gene", "Kind")
    if (!(all(cns %in% colnames(x))))
        stop(paste("Column names are not ",
                   paste(cns, collapse = " ")))
    tmp <- x[, cns]
    tmp$Kind <- factor(tmp$Kind)
    class(tmp) <- c("Cholest_Gene", class(x))
    return(tmp)
}
```

Y volvemos a ejecutar el bloque de comprobaciones:

```
test_that("minimal conversions and failures", {
    expect_s3_class(to_CG(data.frame(Cholesterol = 1:10,
                                     Gene = 11:20,
                                     Kind = "Cl1")),
                    "Cholest_Gene")
    expect_error(to_CG(cbind(Cholesterol = 1:10, Gene = 11:20)),
                 "For now, only methods for data.frame are available",
                 fixed = TRUE)
    expect_error(to_CG(data.frame(Cholesterol = 1:10, Geni = 11:20,
                                  Kind = "Cl1")),
                 "Column names are not",
                 fixed = TRUE)
    expect_s3_class(to_CG(data.frame(Cholesterol = 1:10, Gene = 11:20,
                                     Kind = "Cl1",
                                     whatever = "abcd")), "Cholest_Gene")
})
```

XIII.4. Creación de función de plot

```
## Cholest_Gene object -> ggplot object
## Produce a ggplot of a Cholest_Gene object.
plot.Cholest_Gene <- function(x, ...) {
  class(x) <- "data.frame"
  require(ggplot2)
  ## FIXME: should I explicitly print? Hummm.. return, as orthodox?
  if (nlevels(x$Kind) >= 2)
    p1 <- ggplot(aes(y = Cholesterol, x = Gene, col = Kind),
                 data = x) +
      facet_grid(~ Kind)
  else
    p1 <- ggplot(aes(y = Cholesterol, x = Gene), data = x)
  p1 <- p1 + geom_point()
```

```
  return(p1)
}
```

En R, cuando se pasa un argumento, tan pronto como se utiliza en el interior de la función, se hace una copia. Así, cuando se modifica la clase dentro de una función, no se altera el argumento original, solo la copia interna.

XIII.5. Clases S4

Las clases S4 se utilizan en algunos paquetes de BioConductor. Funcionan de forma similar a las clases S3, pero son más formales y rigurosas.

```
library(Matrix)
m1 <- Matrix(1:9, nrow = 3)
m2 <- Diagonal(5)

x <- 0:10
y <- c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8)
fit1 <- lm(y ~ x)

class(fit1)

## [1] "lm"

is.list(fit1)

## [1] TRUE

isS4(fit1)

## [1] FALSE

print(fit1)

##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##      19.955       -1.682

stats:::print.lm(fit1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##      19.955       -1.682

fit1

##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##      19.955       -1.682

names(fit1)

##  [1] "coefficients"  "residuals"     "effects"
##  [4] "rank"          "fitted.values" "assign"
##  [7] "qr"            "df.residual"   "xlevels"
## [10] "call"          "terms"         "model"

fit1$coefficients

## (Intercept)           x
##   19.954545   -1.681818

## don't do that for real. Use coefficients
coefficients(fit1)

## (Intercept)           x
##   19.954545   -1.681818

isS4(m1)

## [1] TRUE

is.list(m1)

## [1] FALSE

class(m1)
```

```
## [1] "dgeMatrix"
## attr(,"package")
## [1] "Matrix"

slotNames(m1)

## [1] "Dim"      "Dimnames" "x"        "factors"

slotNames(m2)

## [1] "diag"     "Dim"      "Dimnames" "x"

m1

## 3 x 3 Matrix of class "dgeMatrix"
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

m1@Dim

## [1] 3 3

m1@x

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

m2@Dim

## [1] 5 5
```

XIII.6. Resumen sobre la programación orientada a objetos en R

Es recomendable familiarizarse con las clases S3 en R. En BioConductor, es posible encontrarse con las clases S4, pero en general se puede ejecutar todo con clases S3. Hay otras clases, como las R6, pero tienen un uso muy concreto en situaciones muy específicas.

Parte II

Estadística con R

Capítulo XIV

Fundamentos y preparativos

XIV.1. Introducción a la comparación entre dos grupos

Alguien del laboratorio ha medido la expresión de varios genes de un conjunto de pacientes con y sin cáncer. Nosotros somos el encargado de los datos y responder a la pregunta "¿Difiere la expresión de los genes entre los pacientes con y sin cáncer?"

```
dp53 <- data.frame(p53 = round(rnorm(23, c(rep(2, 13), rep(2.8, 10))), 3),
                   pten = round(c(rlnorm(13, 1), rlnorm(10, 1.35)), 3),
                   brca1 = round(rnorm(23, c(rep(2, 13), rep(5.8, 10))), 3),
                   brca2 = round(c(rep(c(1, 2, 3), length.out = 13),
                       rep(c(2, 3, 4), length.out = 10))),
                   cond = rep(c("Cancer", "NC"), c(13, 10)),
                   id = replicate(23, paste(sample(letters, 10), collapse = "")))
```

XIV.2. Tipos de datos

Tenemos que aclarar este punto, ya que nos referiremos a él con frecuencia. Los datos pueden medirse en diferentes escalas. De "menos información a más información" podemos organizar las escalas de esta manera:

Escala nominal o categórica Utilizamos una escala que simplemente diferencia las distintas clases. Por ejemplo, podemos clasificar algunos objetos por aquí, "ordenador", "pizarra", "lápiz", y podemos asignarles números (1 al ordenador, 2 a la pizarra, etc.), pero los números no tienen significado *per se*.

Binario los datos están en una escala nominal con sólo dos clases: muerto o vivo (y podemos dar un 0 o un 1 a cualquiera de ellas), hombre o mujer, etc.

Muchos datos biológicos están en una escala nominal. Por ejemplo, supongamos que nos fijamos en los tipos de elementos repetitivos del genoma y damos un 1 a las SINEs, un 2 a las LINEs, etc. O numera los aminoácidos del 1 (alanina) al

20 (valina). Por supuesto, se puede contar cuántos son del tipo 1 (cuántos son alaninas), etc., pero no tendría sentido hacer promedios y decir «su composición media de AA es de 13,5».

Escala ordinal Los datos pueden ordenarse en el sentido de que se puede decir que algo es mayor o menor que otra cosa. Por ejemplo, puedes ordenar tu preferencia por la comida como: "chocolate > jamón serrano > grillos tostados > hígado". Podrías asignar el valor 1 al chocolate (tu alimento preferido) y un 4 al hígado (el menos preferido), pero las diferencias o proporciones entre esos números no tienen ningún significado.

Escala de intervalos o de proporciones Se pueden tomar diferencias y proporciones, y sí que tienen significado. Si un sujeto tiene un valor de 6 para la expresión del gen PTEN, otro un valor de 3, y otro un valor de 1, entonces el primero tiene seis veces más ARN de PTEN que el último, y dos veces más que el segundo.

XIV.3. Visualizació inicial de datos

El primer paso siempre es mirar los datos. De hecho, aquí podemos ver todos los datos originales. Así que echa un vistazo a los datos.

XIV.3.1. Plots a hacer

Para todos los conjuntos de datos, excepto los más pequeños, debemos utilizar gráficos. Asegúrate de hacer los siguientes gráficos:

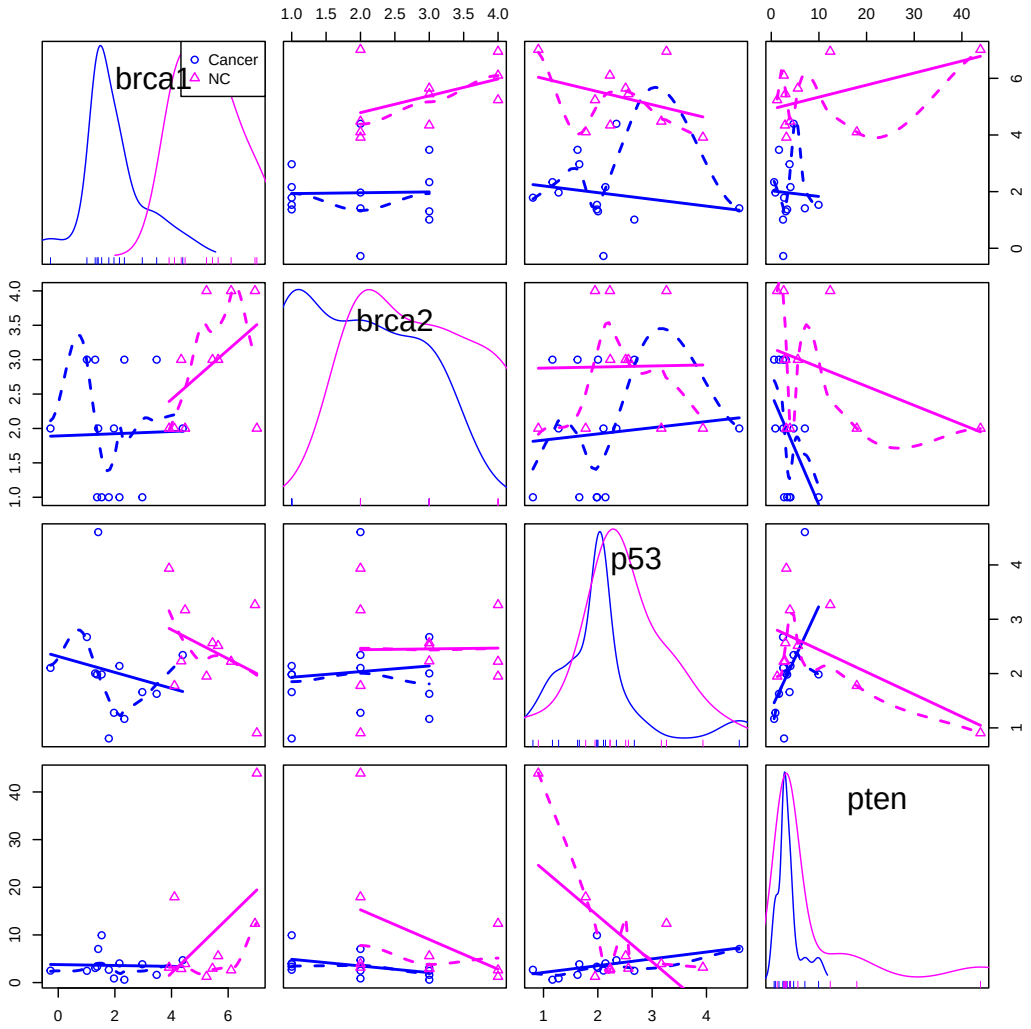
- Histograma de cada gen
- Boxplot
- Plots de medias
- Stripchart con jitter
- Density plots

XIV.3.2. Relación entre variables

Nos centraremos en comparar dos grupos. Pero tenemos varias variables (genes). Una cosa obvia a hacer es: (i) mirar cómo se relacionan y (ii) mostrar los diferentes (dos, en este caso) grupos.

```
library(RcmdrMisc)

## Cargando paquete requerido: sandwich
```

```
scatterplotMatrix( ~ brca1 + brca2 + p53 + pten | cond,
                   data = dp53)
```



No vamos a seguir con esto. Pero se puede y probablemente desee mirar a este tipo de gráficos de forma rutinaria.

Capítulo XV

Comparación entre dos grupos

XV.1. T-test para dos grupos

La forma más sencilla de realizar un test de la t es mediante:

```
t.test(p53 ~ cond, data = dp53)
```

```
##
##  Welch Two Sample t-test
##
## data:  p53 by cond
## t = -1.1376, df = 20.206, p-value = 0.2686
## alternative hypothesis: true difference in means between group Cancer and group
## 95 percent confidence interval:
##  -1.2018502  0.3532041
## sample estimates:
## mean in group Cancer     mean in group NC
##             2.028077             2.452400
```

La prueba t estándar asume que las varianzas de los dos grupos son iguales, mientras que la prueba de Welch no requiere que las varianzas de ambos grupos sean iguales. En la prueba de Welch, los grados de libertad pueden ser un número no entero (como sucede en este caso). Con este estadístico, el programa ha utilizado la distribución t correspondiente y ha calculado el área en ambas colas de la distribución.

XV.1.1. Grados de libertad

Supongamos que tenemos los números 0, 1 y 2, y sabemos que su media es 1. Dado que tenemos tres números, ¿a cuántos de ellos podemos asignar libremente un valor? A dos de ellos, ya que el tercer número debe ajustarse para que el promedio sea 1. Así, el número de grados de libertad es el número de observaciones menos el número de parámetros que debemos estimar. En el caso de tener dos grupos, los grados de libertad se calcularían como:

$$N = N_1 + N_2 = N - 2$$

o de manera equivalente:

$$(N_1 - 1) + (N_2 - 1) = N - 2$$

XV.1.2. Test de Welch vs test de la t

Si las varianzas no son iguales y se realiza una prueba t estándar asumiendo que son iguales, se incurre en un error. En cambio, si las varianzas son realmente iguales, pero se usa la prueba de Welch, el error cometido es menor. Por ello, es preferible utilizar la prueba de Welch cuando existe incertidumbre sobre la igualdad de varianzas. Esta es la razón por la cual, por defecto, se suele optar por la prueba de Welch.

El test de la t sirve para **comparar medias**. La fórmula es:

$$t = \frac{(\bar{x_1} - \bar{x_2}) - (\mu_1 - \mu_2)}{S_{\bar{x_1} - \bar{x_2}}}$$

XV.1.3. Desviación estándar vs error estándar

La desviación estándar es una medida de la dispersión de los datos alrededor de la media, mostrando cuán alejados están los valores individuales del promedio. Es útil para entender la variabilidad dentro de una sola muestra o población. La desviación (σ: poblacional; s: muestral) disminuye cuadráticamente con el tamaño poblacional o muestral, respectivamente:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \qquad s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

En cambio, el error estándar mide la precisión de la media muestral como estimador de la media poblacional. A diferencia de la desviación estándar, que refleja la variabilidad en una muestra específica, el error estándar indica cuánto podrían variar las medias de diferentes muestras si se extrajeran repetidamente de la misma población. Cuanto mayor es el tamaño de la muestra, menor será el error estándar, ya que la media muestral se aproxima más a la media poblacional.

$$E[\bar{X}] = \mu; \qquad E[S^2] \neq \sigma^2$$

XV.1.4. Ideas clave sobre el test de la t

Es importante comprender algunas ideas clave sobre el uso de los p-valores en la prueba de hipótesis. Primero, cuando el resultado de un análisis no es estadísticamente significativo, no estamos confirmando la hipótesis nula; simplemente es posible que estemos fallando en rechazarla, lo que significa que no hemos encontrado suficiente evidencia en su contra. Además, el p-valor no representa la probabilidad de que la hipótesis nula sea cierta, ni de que la hipótesis alternativa lo sea. En cambio, el p-valor sirve como una métrica que indica la fuerza de la evidencia **en contra** de la hipótesis nula. Si el p-valor es bajo, la interpretación es que, *"o bien la hipótesis nula es falsa, o*

*bien hemos observado un evento tan **im**probable como el p-valor calculado, dado que la hipótesis nula es cierta".*

Estas tres preguntas son distintas: 1) ¿Qué dice la evidencia? Es una prueba estadística, y lo responde el p-valor. 2) ¿Qué debo creer? Quizás hay evidencia adicional; se calcula mediante la inferencia bayesiana. 3) ¿Qué debo hacer? Esto refleja otra relación de coste-beneficio, y resulta en la toma de la decisión que se realiza (aceptar o rechazar la hipótesis).

Es fundamental recordar que los p-valores se calculan bajo ciertos supuestos de modelo, y cualquier violación de estos supuestos puede afectar la validez del resultado. Por eso, utilizar los p-valores de manera cuidadosa es más adecuado que interpretar resultados en términos absolutos de "significativo" o "no significativo". Además, comparar valores extremadamente pequeños de p (como $p = 10^{-13}$ frente a $p = 10^{-16}$) no tiene un significado práctico adicional, ya que ambos ya representan un nivel de evidencia considerablemente fuerte en contra de la hipótesis nula. También es esencial reconocer que el p-valor no es la única herramienta de inferencia estadística; los intervalos de confianza proporcionan información valiosa sobre el rango de valores plausibles para el parámetro de interés, complementando el análisis de los p-valores y ayudando a interpretar mejor los resultados.

Para comprender la inferencia estadística, es esencial distinguir entre una muestra y la población. La población es el conjunto completo de elementos sobre el cual queremos obtener conclusiones, mientras que una muestra es un subconjunto de esa población que se selecciona para su análisis. Mayoritariamente, se trabaja con muestras porque estudiar toda una población suele ser impracticable; a partir de los datos de la muestra, hacemos inferencias sobre las características de la población.

Un concepto fundamental en estadística es el de un estadístico, que es cualquier valor numérico que se puede calcular a partir de una muestra. Un tipo específico de estadístico es un estimador, que se usa para aproximar un parámetro de la población. Por ejemplo, la media muestral, calculada como $\sum x/N$, es un estimador que proporciona una aproximación de la media poblacional verdadera utilizando datos de una muestra.

Un tipo particular de estadístico es el estadístico t, utilizado en la prueba t para contrastar hipótesis sobre las medias de dos grupos. Tanto los estadísticos en general como los estimadores específicos tienen distribuciones propias, que describen cómo se distribuyen sus valores posibles si el muestreo se repitiera muchas veces. Esta variabilidad introducida por el muestreo afecta las conclusiones y debe tenerse en cuenta.

Otro aspecto clave es entender la diferencia entre desviación estándar y error estándar. La desviación estándar mide la variabilidad de los datos dentro de la muestra, mientras que el error estándar refleja la variabilidad de la media muestral con respecto a la media poblacional.

En cuanto al p-valor, es una medida de la evidencia en contra de la hipótesis nula (H0), que plantea que no hay efecto o diferencia. Al calcular el p-valor, se supone que los estadísticos siguen una distribución específica bajo la hipótesis nula, lo cual permite evaluar la probabilidad de obtener un resultado tan extremo como el observado.

La lógica de un test estadístico radica en decidir entre la hipótesis nula y la alternativa basándose en los datos. A diferencia de un procedimiento de estimación, que busca obtener un valor aproximado de un parámetro, una prueba de hipótesis se centra en determinar si la evidencia es suficientemente fuerte para rechazar la hipótesis nula. Esta diferencia entre estimación y prueba de hipótesis es fundamental para realizar inferencias estadísticas bien informadas.

XV.1.5. Intervalos de confianza

Un intervalo de confianza del 95 % alrededor de una estimación, como una media, no debe interpretarse como que existe una probabilidad del 95 % de que la media poblacional esté entre los límites del intervalo, por ejemplo, entre 1 y 2. Esta interpretación es incorrecta. La interpretación correcta de un intervalo de confianza del 95 % es que, si repitiéramos el muestreo y el cálculo del intervalo de confianza muchas veces, aproximadamente el 95 % de esos intervalos generados contendrían la media poblacional real. El intervalo refleja la precisión de la estimación dada la variabilidad del muestreo, no una probabilidad sobre la ubicación de la media en un intervalo específico para una muestra concreta.

En el contexto de un test de hipótesis, si el test es justamente significativo (es decir, si el p-valor es 0,05), uno de los límites del intervalo de confianza tocará el valor de 0, indicando que no se puede rechazar la hipótesis nula con un nivel de confianza superior al 95 %. Cuando el valor t calculado aumenta (es decir, la evidencia contra la hipótesis nula se vuelve más fuerte), el intervalo de confianza se amplía, reflejando una mayor certeza en la estimación. Por ejemplo, un valor t de 18 corresponde a un área bajo la curva mucho mayor que un valor t de 4, lo que implica una estimación mucho más precisa y una evidencia más fuerte en favor de rechazar la hipótesis nula.

XV.1.6. Supuestos del test de la t

Un supuesto clave en la prueba t es la **independencia de los datos**. Este requisito no solo es esencial para la prueba t, sino también para muchas otras pruebas estadísticas. La falta de independencia entre observaciones es un problema grave y común en los estudios estadísticos. Una forma de dependencia, conocida como pseudorreplicación, ocurre cuando las observaciones no son realmente independientes, lo que puede sesgar los resultados y llevar a interpretaciones incorrectas.

Cuando se comparan dos medias, otro supuesto importante es la **igualdad de varianzas**. Sin embargo, detectar diferencias en las varianzas no siempre es sencillo. Dos soluciones prácticas ante la posible desigualdad de varianzas son el uso de la prueba de Welch (predeterminada en software estadístico como R) y la aplicación de transformaciones de datos. No obstante, antes de continuar con la comparación, conviene preguntarse si realmente tiene sentido comparar medias cuando las varianzas de los grupos difieren considerablemente, ya que diferencias amplias en la variabilidad pueden afectar la interpretación de las medias.

En cuanto a la **normalidad** de los datos, este supuesto es menos restrictivo, especialmente a medida que aumenta el tamaño de la muestra. Es importante notar que, al hablar de normalidad, simetría y otros aspectos de la distribución, se hace

referencia a la **distribución de cada grupo por separado**. Las desviaciones de la normalidad debido a la *asimetría* pueden tener un efecto significativo en los resultados, mientras que las desviaciones relacionadas con una mayor o menor *curtosis* (colas más pesadas o ligeras que la normal) suelen tener un impacto menor. Por eso, comúnmente se acepta que los datos estén "suficientemente cerca de la normalidad," prestando especial atención a la asimetría de la distribución. Con tamaños de muestra grandes, la normalidad de los datos suele ser menos preocupante gracias al *teorema del límite central*, que establece que, a medida que aumenta el tamaño de la muestra, la distribución de la media muestral se aproxima a una distribución normal. ¿Cuándo una muestra es lo suficientemente grande? La respuesta depende de cuánto difieran los datos de la normalidad. En muchas situaciones, un tamaño de muestra de 10 puede ser suficiente; 50 generalmente es adecuado y, en algunos casos, incluso muestras de 100 observaciones podrían no ser suficientes si la distribución es extremadamente no normal.

Por último, los **valores atípicos o outliers** pueden ser una preocupación seria en el análisis de datos. De hecho, los valores atípicos, o los valores potencialmente atípicos según alguna definición, son identificados por la función `Boxplots` en R. En general, los puntos que están muy alejados del resto de los datos pueden tener efectos graves sobre la media calculada, pero no sobre la mediana (esto es uno de los motivos por los cuales los procedimientos no paramétricos suelen ser más robustos frente a valores atípicos). Sin embargo, decidir qué hacer con los valores atípicos no es una tarea sencilla. Un valor atípico podría ser el resultado de un error en el registro de los datos, pero también podría ser un dato perfectamente válido y, de hecho, podría ser lo "interesante" del análisis. En algunos casos, se realizan análisis con y sin el valor atípico para comparar los resultados (y, por supuesto, se debe informar explícitamente de esto). A veces, se llegan a las mismas conclusiones cualitativas, pero otras veces no. Por tanto, antes de decidir cómo tratar los valores atípicos, es fundamental reflexionar cuidadosamente sobre lo que se considera un valor atípico en el contexto del análisis y el objetivo del estudio. No se debe caer en la tentación de eliminar automáticamente los valores atípicos sin una justificación sólida. Y, en cualquier caso, cualquier decisión sobre cómo tratar los valores atípicos debe ser documentada y comunicada de manera transparente.

XV.2. Tests de una y dos colas

Hasta ahora, hemos trabajado con tests de dos colas. Sin embargo, en algunas situaciones es posible limitar el análisis a una sola cola. En un test de dos colas, la hipótesis nula plantea que las medias son iguales, y cualquier desviación en ambas direcciones puede llevar al rechazo de la hipótesis nula. En contraste, un test de una cola permite especificar una dirección para la hipótesis. Por ejemplo, podemos plantear como hipótesis nula que $\mu_1 \geq \mu_2$, y como hipótesis alternativa que $\mu_1 < \mu_2$, concentrándonos solo en una dirección de la desviación.

Para un mismo estadístico t, un test de una cola tendrá un p-valor igual a la mitad del p-valor de un test de dos colas, ya que se considera únicamente una de las colas de la distribución. Sin embargo, por convención y para evitar sesgos, lo normal es realizar

un test de dos colas, especialmente si no existe una razón científica sólida para anticipar la dirección del efecto.

Algunos tests, como el ANOVA, utilizan la distribución F, la cual tiene una sola cola de manera natural, ya que evalúa si existe variabilidad significativa entre varios grupos en cualquier dirección sin considerar una dirección específica. En el caso del test de la t, se debe decidir entre un test de una o dos colas en función de la hipótesis científica planteada y siempre antes de observar los datos, para evitar que los resultados influyan en la elección del tipo de test.

XV.3. Consideraciones sobre potencia estadística de un test

Si existe una verdadera diferencia de medias, nos gustaría detectarla. La potencia se refiere a nuestra capacidad para rechazar el nulo cuando es falso. Esta figura puede ayudar; las filas se refieren al estado real del Universo y las columnas a la decisión que se toma.

	Hipótesis nula no se rechaza	Hipótesis nula se rechaza
Medias no difieren (H_0 *es cierta*)	Correcto	Type I error
Medias difieren (H_0 *es falsa*)	Type II error	Correcto

No es posible realizar un test con un error de tipo I extremadamente pequeño sin aumentar el error de tipo II, ya que reducir al mínimo la probabilidad de un error de tipo I generalmente incrementa la probabilidad de un error de tipo II. Por ello, es necesario encontrar un equilibrio adecuado entre ambos tipos de error. Al diseñar un test, se debe establecer un nivel de significancia o error de tipo I nominal, generalmente expresado como α, que refleje la probabilidad aceptable de rechazar la hipótesis nula cuando en realidad es cierta. Este valor nominal permite controlar de forma explícita la tasa de error de tipo I, manteniendo el test en un nivel de confianza apropiado para los objetivos del estudio.

La potencia es $1 - Type\ II\ error$. La potencia es la probabilidad de rechazar la hipótesis nula cuando la hipótesis nula es falsa.

La probabilidad de que se detecte una diferencia que realmente existe (potencia) depende de:

- El umbral que se utilice para decir que "las medias difieren" (α o error de tipo I).[1]
- El tamaño de la muestra
- El tamaño del efecto (distancia de medias)

[1] El valor p es una función de los datos, es algo que se calcula con un procedimiento determinado para un conjunto de datos determinado; el nivel α o la tasa de error de tipo I es una propiedad del procedimiento.

- La desviación estándar de la población

La potencia se puede calcular de antemano para saber si es probable encontrar una diferencia en caso de que la haya (dado el tamaño de la muestra y los tamaños de efecto y las desviaciones estándar estimados) y averiguar si el tamaño de la muestra es adecuado para la potencia deseada (y los tamaños del efecto y las desviaciones estándar estimados). Es importante recalcar que no tiene mucho sentido calcular la potencia del test después de haberlo calculado, ya que no aporta nada de valor.

XV.3.1. Maldición del ganador

En estudios con baja potencia, las estimaciones de los efectos de las pruebas que resultan "significativas" tienden a estar sesgadas al alza, es decir, a ser mayores de lo que realmente deberían ser. Esto significa que, para un mismo fenómeno, cuando solo se consideran estudios de baja potencia con valores p significativos, las estimaciones del efecto suelen ser excesivamente grandes (en términos absolutos). Así, el sesgo de publicación, junto con la baja potencia, puede llevar a una sobreestimación sistemática de los tamaños del efecto reportados en la literatura.

Además, el tamaño de la muestra afecta el valor de p asociado a un estadístico t dado. Para un mismo valor de t, un tamaño de muestra grande se traduce en un p-valor más pequeño que el que obtendríamos con un tamaño de muestra pequeño, lo que significa que la significancia estadística es más fácil de alcanzar con muestras grandes, incluso si el efecto real es pequeño. Este fenómeno subraya la importancia de interpretar los valores p en contexto, considerando tanto el tamaño de muestra como la potencia del estudio para obtener una estimación realista del efecto.

Capítulo XVI

Inferencia estadística

XVI.1. (Bio)equivalencia

Hemos configurado las cosas de modo que **necesitamos pruebas suficientemente sólidas para rechazar el nulo** y **utilizamos p-valores de medidas de fuerza de las pruebas CONTRA el nulo**. Esto es a menudo lo que queremos en la ciencia, pero no siempre. Y en muchos casos, en particular en cuestiones relacionadas con la salud pública, es posible que queramos seguir un principio de precaución.

Por ejemplo, tal vez queramos decir: «Sólo permitiremos verter cloro en el río si hay pruebas suficientemente sólidas de que tal acción no causará daños, por ejemplo, no aumentará la mortalidad de los peces». Esto no es algo que se pueda resolver con valores p tal y como los hemos utilizado.

¿Qué podemos hacer? Queremos darle la vuelta al proceso. Querríamos un procedimiento para responder a la siguiente pregunta: «¿Existen pruebas suficientemente sólidas de que, si el cloro tiene un efecto, éste no es mayor que un aumento de la mortalidad de los peces del 1 %?». Esto es como invertir la carga de la prueba: es como si ahora quisiéramos pruebas a favor de una hipótesis que dice que las cosas no difieren en más de un valor dado, pequeño (es decir, parece que ahora queremos pruebas a favor de lo que a menudo es el nulo). En otras palabras, queremos pruebas sólidas de que el valor verdadero está dentro de los límites de equivalencia, los límites que dicen que «las cosas son similares o equivalentes» (hemos simplificado las cosas aquí, preocupándonos sólo por los aumentos en la mortalidad de los peces, pero a menudo nos preocupamos por las desviaciones tanto hacia arriba como hacia abajo).

Podemos enfocar este problema como la búsqueda de pruebas contra la hipótesis (nueva nula) de que las cosas difieren en más de la tolerancia especificada, en nuestro caso ese 1 % de aumento en la mortalidad de los peces; en otras palabras, que el valor verdadero cae fuera de los límites de equivalencia. Si podemos rechazar nuestra nueva hipótesis nula de que los grupos difieren en más de un umbral determinado (que la diferencia real queda fuera de los límites de equivalencia), habremos establecido que son equivalentes. En algunos casos es relativamente sencillo hacerlo (como con el procedimiento TOST; realizando dos tests de una cola), pero en muchos otros no lo es.

XVI.2. Inferencia bayesiana

El teorema de Bayes es una fórmula fundamental en probabilidad condicional que permite calcular la probabilidad de un evento dado que otro evento ha ocurrido. Su expresión general es:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

En estadística, el teorema de Bayes se aplica de la siguiente manera:

$$P(H_0|\bar{x_A} - \bar{x_B} = 3) = \frac{P(\bar{x_A} - \bar{x_B} = 3|H_0) \cdot P(H_0)}{P(\bar{x_A} - \bar{x_B} = 3)}$$

Sin embargo, una dificultad importante en la aplicación de la inferencia bayesiana en este contexto es la estimación de la probabilidad previa de la hipótesis nula, $P(H_0)$, antes de realizar el test. Asignar un valor adecuado a esta probabilidad previa es crucial, pero puede ser complicado y, en algunos casos, controvertido.

El teorema de Bayes es ampliamente utilizado en estadística sin controversias en áreas como el diagnóstico médico. Por ejemplo, calcular la probabilidad de padecer una enfermedad dado un resultado positivo en un test diagnóstico es una aplicación común. Sin embargo, la interpretación de un mismo resultado depende del contexto que se tenga. En el caso de un test de sangre en heces para detectar cáncer de colon, un resultado positivo no necesariamente implica que la persona tenga la enfermedad, debido a la posibilidad de falsos positivos. En estos casos, el teorema de Bayes nos ayuda a comprender la probabilidad real de la enfermedad, considerando tanto la precisión del test como la prevalencia de la enfermedad en la población.

XVI.3. Intervalos de confianza e interpretación de p-valores

Al interpretar intervalos de confianza, es crucial considerar tanto la posición de la media como el rango en el que se concentra la mayor parte de los valores posibles.

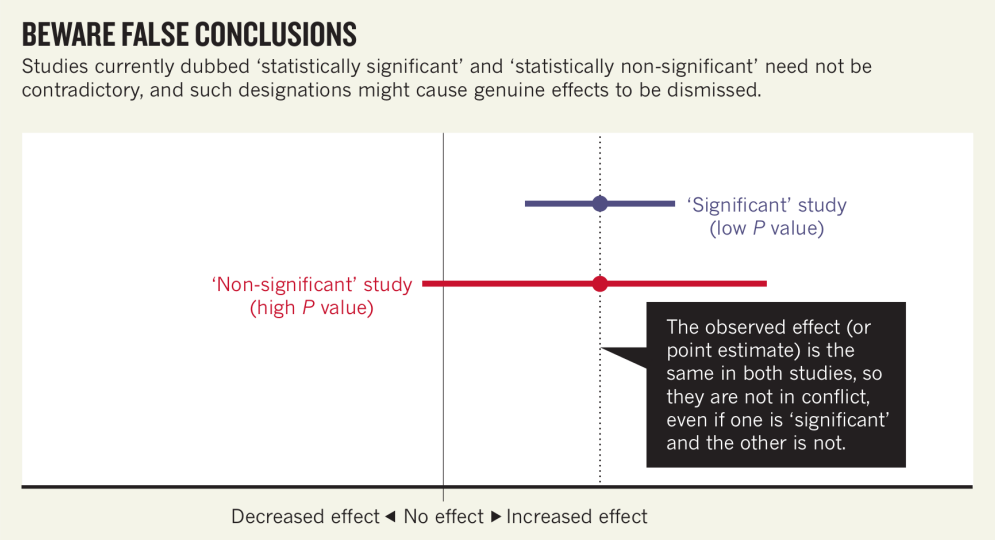
Supongamos los siguientes intervalos de confianza en los que la hipótesis nula es $H_0 = \mu_1 - \mu_2 = 0$.

En el caso del primer intervalo de confianza, aunque incluye el valor 0 y, por tanto, no se rechaza la hipótesis nula, la media estimada está bastante alejada de 0, lo que sugiere que muchos de los valores dentro del intervalo no son consistentes con la hipótesis nula. Esto podría ser indicativo de un tamaño de muestra pequeño, y no rechazar la hipótesis nula sin más podría no ser adecuado. En el segundo caso, la hipótesis nula se rechaza, y el intervalo de confianza, que es pequeño y distante de 0, respalda una diferencia clara. Sin embargo, si el intervalo estuviera cerca de 0, rechazar la hipótesis nula podría tener menos relevancia práctica, ya que los valores observados indicarían una diferencia mínima.

Es importante recordar que los intervalos de confianza del 99 % son más amplios que los del 95 %, y estos, a su vez, son más amplios que los del 90 %. Cuanto mayor es el

nivel de confianza, más amplio será el intervalo, lo que refleja una mayor incertidumbre en la estimación.

Hay que evitar conclusiones erróneas al interpretar significación estadística. No siempre es contradictorio que un estudio resulte "significativo" mientras otro no lo sea, incluso si el efecto observado es el mismo en ambos. Por ejemplo, un estudio con un p-valor bajo (significativo) y otro con un p-valor alto (no significativo) pueden tener medias similares si el tamaño de muestra o la variabilidad difieren entre los estudios.



Los intervalos de confianza, en general, ofrecen más información que los p-valores, ya que muestran los valores que son consistentes con lo observado y permiten evaluar el rango de posibles efectos. En el gráfico, ambos intervalos muestran valores compatibles con la hipótesis alternativa, sugiriendo una diferencia entre grupos.

Finalmente, la interpretación de un p-valor adecuado depende del contexto. Aunque históricamente se ha utilizado un umbral de 0,05, en ciertos contextos este nivel puede ser demasiado alto, y puede ser necesario establecer un criterio más estricto o considerar otras métricas adicionales según la naturaleza del estudio.

Capítulo XVII

Comparación de datos emparejados

XVII.1. Pruebas estadísticas para datos emparejados

Los tests apareados son un tipo de análisis estadístico diseñado para comparar dos medidas tomadas sobre el mismo grupo de individuos o unidades experimentales bajo condiciones distintas. Su uso es especialmente común en estudios donde se desea evaluar el efecto de un tratamiento o intervención midiendo a los mismos sujetos en dos momentos diferentes (pre y post intervención) o bajo dos condiciones diferentes. Al comparar cada sujeto consigo mismo, estos tests ayudan a controlar la variabilidad intrasujeto y, por tanto, pueden aumentar la precisión y potencia estadística en comparación con un test de muestras independientes.

En los tests apareados, el análisis se enfoca en las diferencias intrasujeto (o intraunidad), lo que permite aislar el efecto de la condición o el tiempo sobre cada individuo. El test de la t apareado, una de las pruebas más usadas para este tipo de análisis, evalúa si la media de las diferencias entre dos medidas es significativamente distinta de cero, lo cual indicaría una diferencia sistemática entre las condiciones evaluadas.

Para que los resultados de un test apareado sean válidos, es crucial que las medidas sean independientes entre sujetos y que cada par de medidas esté correctamente ordenado para cada sujeto. Esto asegura que cada par se refiera al mismo individuo en ambas condiciones, de manera que el test pueda evaluar directamente las diferencias intrasujeto.

```r
set.seed(15)
s <- rnorm(12, 4, 25)

s <- c(s, s)
cond <- rep(c(0, .5), c(12, 12))
y <- rnorm(24) + s + cond
y <- y - min(y) + 0.3
```

```r
id <- replicate(12, paste(sample(letters, 10), collapse = ""))
id <- c(id, id)
dmyc <- data.frame(myc = round(y, 3),
                   cond = rep(c("Cancer", "NC"), c(12, 12)),
                   id = id)
```

XVII.1.1. Test de la t apareados

```r
myc.cancer <- dmyc$myc[dmyc$cond == "Cancer"]
myc.nc <- dmyc$myc[dmyc$cond == "NC"]
t.test(myc.nc, myc.cancer, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  myc.nc and myc.cancer
## t = 4.079, df = 11, p-value = 0.001823
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  0.432056 1.444777
## sample estimates:
## mean difference
##       0.9384167
```

En este análisis, se mide a 12 sujetos en dos condiciones diferentes, generando un total de 24 observaciones. Sin embargo, al tratarse de un test apareado, el análisis se enfoca en las 12 diferencias intrasujeto entre ambas condiciones, lo que implica que hay 11 grados de libertad.

El resultado del test de R muestra que se ha realizado un test apareado y proporciona el valor de t, los grados de libertad (df) y el p-valor asociado. Además, señala que la hipótesis alternativa es que la diferencia entre las medias de las dos condiciones no es igual a 0, refiriéndose a la diferencia intrasujeto.

El output incluye un intervalo de confianza del 95 % para la media de las diferencias, que en este caso está desplazado respecto al 0 (lo cual puede sugerir una diferencia significativa entre las condiciones). La media de las diferencias (*mean differences*) indica el promedio de la variación intrasujeto entre ambas condiciones.

Es crucial que los datos estén correctamente ordenados para cada sujeto en ambas condiciones. Esto significa que los dos vectores pasados al test deben tener las observaciones de cada sujeto en el mismo orden, ya que el test apareado compara las diferencias exactas entre las condiciones para cada sujeto.

XVII.1.2. Remodelación de los datos para un test emparejado

Cuando se va a realizar un test de la t emparejado, se pueden organizar los datos en estructuras como las siguientes:

SubjectID	Tumor	Non-Tumor
pepe	23	45
maria	29	56
...	...	...

Tabla XVII.1: *Paired data in a "unstacked or wide" shape/format.*

SubjectID	Myc	Condition
pepe	23	tumor
pepe	45	nontumor
maria	29	tumor
maria	56	nontumor
...	...	...

Tabla XVII.2: *Paired data in a "stacked or long" shape/format.*

En general, es más útil tener los datos organizada de forma "apilada".

```
(merged3 <- reshape(dmyc, direction = "wide", idvar = "id",
                    timevar = "cond", v.names = "myc"))

##            id myc.Cancer myc.NC
## 1  bqysitlvpm     38.289 39.634
## 2  zuhxmiyfos     76.188 78.361
## 3  bpkmxwhtsg     24.621 24.396
## 4  qsmyexkcnw     54.079 53.902
## 5  uhbkifsnvw     43.832 44.679
## 6  efzpcboidt      0.300  1.675
## 7  trsyacmejh     31.055 32.260
## 8  hyqjownkue     58.402 59.427
## 9  ejmkobsqrh     29.723 30.300
## 10 mculjayvhw      6.190  6.030
## 11 ytwgsplaef     52.626 54.494
## 12 dchlnopykg     22.089 23.497

dmycWide <- reshapeL2W(dmyc, within="cond", id="id", varying="myc")
```

XVII.1.3. El test de la t emparejado - plots

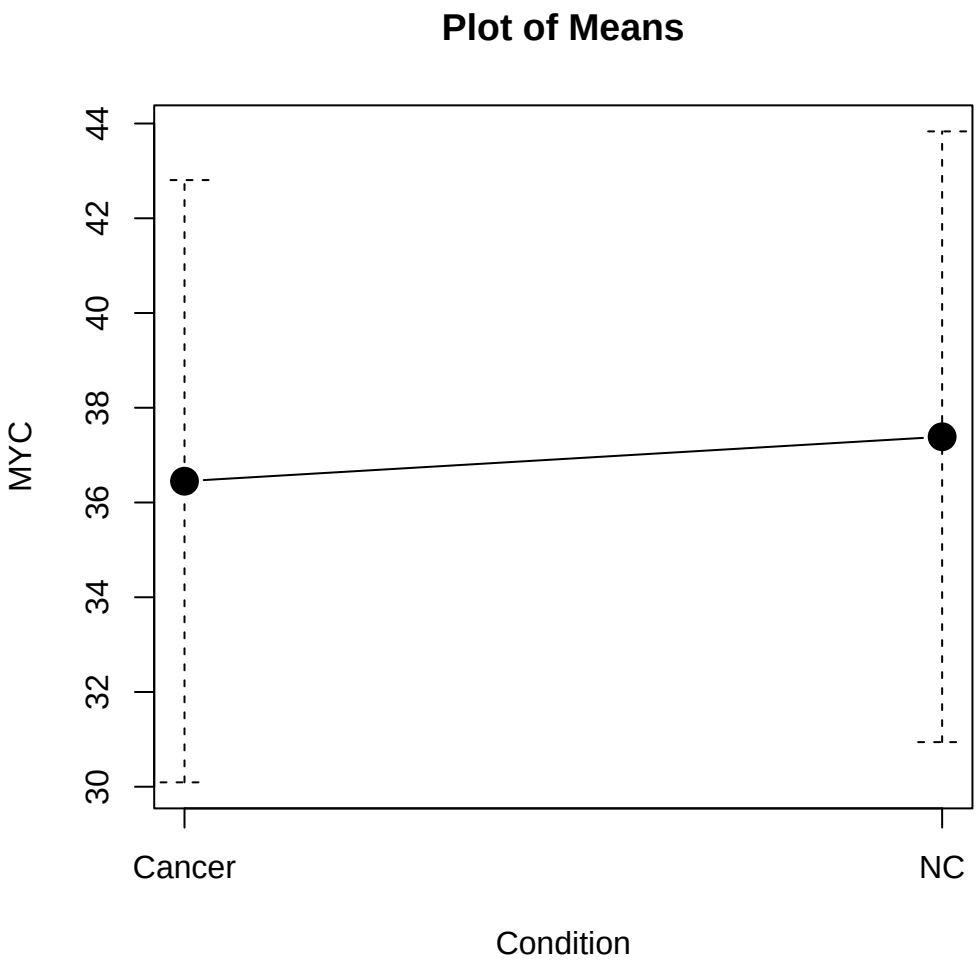
```r
## Paired
t.test(merged3$myc.NC, merged3$myc.Cancer, alternative='two.sided',
       conf.level=.95,  paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  merged3$myc.NC and merged3$myc.Cancer
## t = 4.079, df = 11, p-value = 0.001823
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  0.432056 1.444777
## sample estimates:
## mean difference
##       0.9384167
```

```r
t.test(myc ~ cond, alternative = 'two.sided', conf.level=.95,
       var.equal=FALSE,  data=dmyc)
```

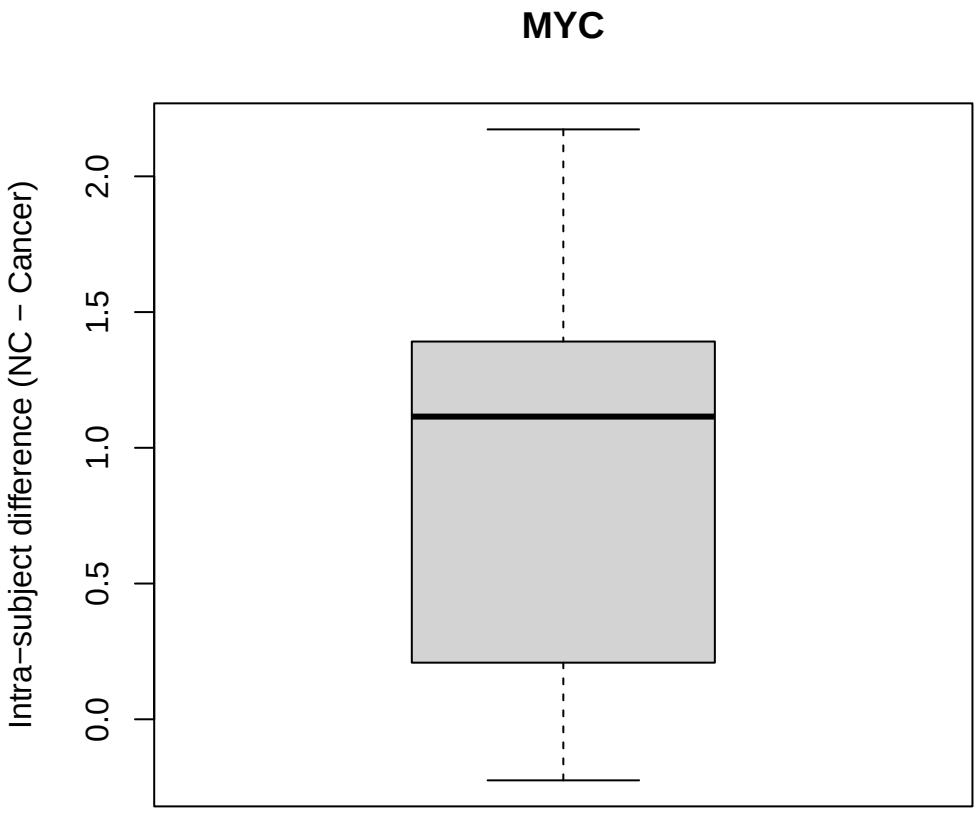
```
##
##  Welch Two Sample t-test
##
## data:  myc by cond
## t = -0.10365, df = 21.996, p-value = 0.9184
## alternative hypothesis: true difference in means between group Cancer and group
## 95 percent confidence interval:
##  -19.71435  17.83752
## sample estimates:
## mean in group Cancer     mean in group NC
##             36.44950             37.38792
```

```r
plotMeans(dmyc$myc, dmyc$cond, error.bars = "se", ylab = "MYC",
          xlab = "Condition")
```



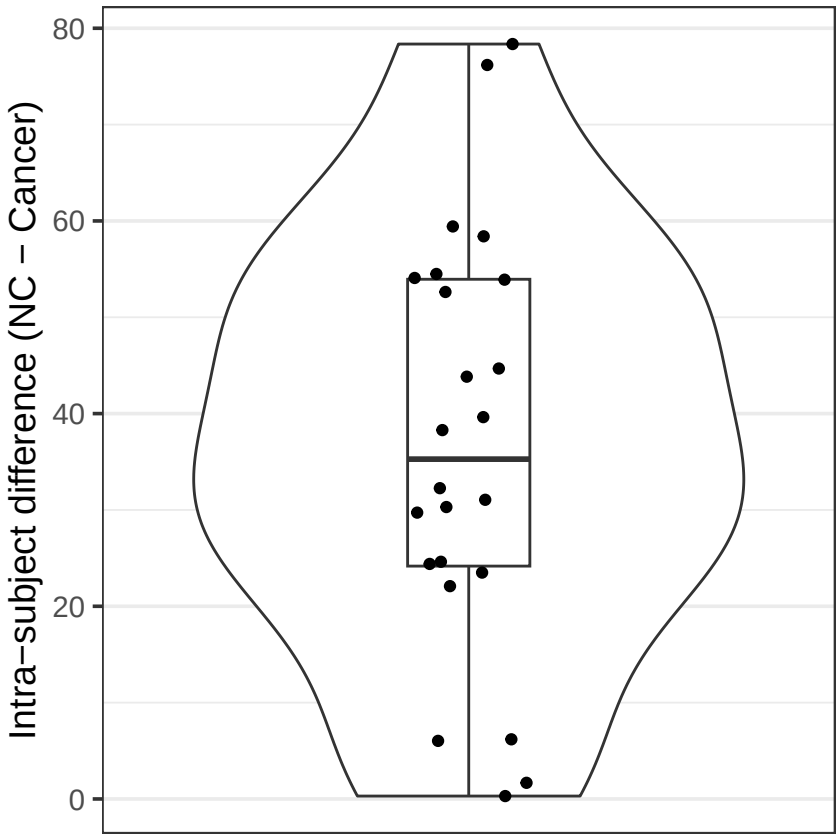
Los intervalos de confianza solapan a lo largo de sus recorridos. En general, es un mal plot para datos emparejados al no reflejarse que cada sujeto se ha medido en dos condiciones.

```
diff.nc.c <- (myc.nc - myc.cancer)
t.test(diff.nc.c)

##
##  One Sample t-test
##
## data:  diff.nc.c
## t = 4.079, df = 11, p-value = 0.001823
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.432056 1.444777
## sample estimates:
## mean of x
## 0.9384167

Boxplot( ~ diff.nc.c, data = merged3, xlab = "",
         ylab = "Intra-subject difference (NC - Cancer)", main = "MYC")
```



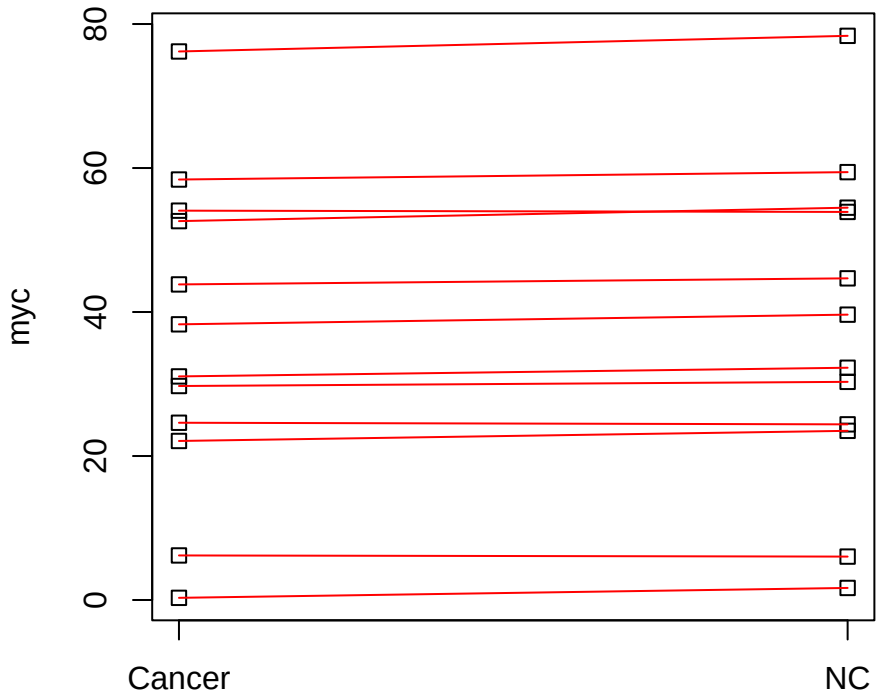
Aquí se calcula la diferencia intrasujeto y se muestra en el plot. Es una mejor representación, ya que el grueso de los pares son desviaciones positivas.

```
library(ggplot2)

dftmp <- data.frame(y = dmycWide$diff.nc.c)
theplot <- ggplot(data = dftmp, aes(x = factor(1), y = y)) +
  geom_violin() + geom_boxplot(width = 0.2) +
  geom_jitter(colour = "black", width = 0.1, height = 0) +
  scale_x_discrete(breaks = NULL) +
  xlab("") +
  ylab("Intra-subject difference (NC - Cancer)") +
  theme_bw(base_size = 14, base_family = "sans") +
  theme(axis.title.x = element_blank(), axis.text.x = element_blank())
print(theplot)
rm(dftmp, theplot)
```



Aquí, el número de observaciones es pequeño, por lo que el plot violín no está justificado; pero sería la mejor representación. Los puntos son las diferencias intrasujeto.

```
stripchart(myc ~ cond, vertical = TRUE, data = dmyc)
for(i in unique(dmyc$id))
  segments(x0 = 1, x1 = 2,
           y0 = dmyc$myc[dmyc$cond == "Cancer" & dmyc$id == i],
           y1 = dmyc$myc[dmyc$cond == "NC" & dmyc$id == i],
           col = "red")
```



Este plot es bastante feo. El objetivo está en mostrar por qué el test de la t muestra grandes diferencias, pero el plot no. En casi todos los casos, a diferencia intrasujeto es positiva (NC-cancer da un resultado positivo). Además, la variabilidad intersujeto es muy grande en relación con la magnitud del efecto. Los dos valores de un sujeto están altamente correlacionados, pero los grados de libertad son menores.

XVII.2. Procedimientos no paramétricos

Los procedimientos no paramétricos son métodos estadísticos que no dependen de supuestos específicos sobre la distribución de los datos, como la normalidad. En lugar de trabajar directamente con los valores originales, a menudo convierten los datos en rangos, lo que les permite ser menos sensibles a valores atípicos o desviaciones significativas de los supuestos paramétricos. Aunque ofrecen robustez frente a ciertas violaciones de los supuestos, estos métodos pueden estar evaluando hipótesis nulas ligeramente diferentes en comparación con los procedimientos paramétricos, lo que debe considerarse al elegir la metodología.

El uso de procedimientos no paramétricos puede ser una decisión compleja, y su aplicación depende de diversos factores:

- ¿La naturaleza de los datos justifica un enfoque no paramétrico? Si los datos presentan distribuciones altamente sesgadas, valores atípicos extremos o una

estructura que claramente viola los supuestos paramétricos, un procedimiento no paramétrico puede ser más adecuado.

- Eficiencia relativa del procedimiento no paramétrico: La eficiencia relativa mide el tamaño de la muestra necesario para que dos procedimientos con una tasa de error de tipo I similar alcancen la misma potencia estadística. Cuando se cumplen los supuestos de los métodos paramétricos, estos suelen tener mayor potencia estadística. Sin embargo, cuando los supuestos no se cumplen, los métodos no paramétricos pueden ser más robustos y ofrecer una potencia comparable o incluso superior. Cabe señalar que los métodos no paramétricos tienen limitaciones en la detección de valores p extremadamente pequeños, lo que puede ser un desafío en contextos como experimentos ómicos donde se aplican correcciones para pruebas múltiples.
- Flexibilidad del método: Algunos métodos no paramétricos tienen limitaciones para incorporar factores experimentales adicionales o interacciones complejas. Por ello, es importante evaluar si un procedimiento no paramétrico puede adaptarse al diseño del experimento.

!!!! Aquí nos centraremos en la prueba de Wilcoxon.Esta suele ser la forma de proceder cuando tenemos medidas de escala ordinal y queremos comparar dos grupos independientes. Sin embargo, la prueba de Wilcoxon **requiere datos de escala de intervalo** para el Wilcoxon de una sola muestra y el Wilcoxon pareado (esto es algo que muchos sitios web y algunos libros de texto hacen mal). La razón detrás de esta restricción es que la prueba de Wilcoxon para datos pareados comienza calculando las diferencias intra-sujeto. Este cálculo implica restar valores entre pares de observaciones, lo que requiere que los datos tengan una distancia significativa y consistente entre los valores. Por lo tanto, no es adecuado aplicar esta prueba a datos de escala ordinal, ya que en tales casos no existe una medida de "distancia real" entre los niveles ordinales.

Además, aunque la prueba de Wilcoxon es más robusta frente a ciertos supuestos que los métodos paramétricos, la hipótesis de independencia de las observaciones sigue siendo fundamental. Es decir, las observaciones dentro y entre los grupos deben ser independientes. Este supuesto es tan crucial para el Wilcoxon como lo es para la prueba t.

Es importante recordar que las pruebas no paramétricas no son completamente «libres de supuestos». Ningún método estadístico lo es, ni en teoría ni en la práctica. Si bien estas pruebas pueden ser más flexibles frente a ciertas violaciones de los supuestos paramétricos, aún requieren que se cumplan otros supuestos esenciales para garantizar la validez de los resultados. Por ello, la elección de la prueba debe basarse no solo en las características de los datos, sino también en el cumplimiento de estos supuestos.

XVII.2.1. Wilcoxon rank-sum test or Mann-Whitney U test: 2 muestras independentes

La prueba Wilcoxon rank-sum, también conocida como Mann-Whitney U, se utiliza para comparar dos muestras independientes. Es adecuada para datos de escala ordinal o de intervalo, y su objetivo principal es evaluar si existe una diferencia significativa entre las distribuciones de los dos grupos.

La lógica básica del test consiste en combinar las observaciones de ambos grupos en una sola lista, clasificar las observaciones asignando rangos, calcular las sumas de los rangos correspondientes a cada grupo por separado y evaluar si la suma de los rangos de un grupo es significativamente mayor (o menor) que la del otro.

La hipótesis nula indica que las dos muestras provienen de la misma población (o poblaciones con la misma distribución). La hipótesis alternativa es que las dos muestras provienen de poblaciones con distribuciones diferentes (la forma específica depende de si la prueba es de una cola o dos colas).

Es una prueba no paramétrica, por lo que no requiere asumir normalidad en los datos. Es robusta frente a valores atípicos y puede utilizarse con datos de escala ordinal. Aunque no requiere normalidad, la independencia entre las observaciones de los dos grupos es fundamental. En algunos casos, esta prueba no detecta diferencias en las medias sino en la posición o distribución de los valores, lo que puede interpretarse como diferencias en la mediana si las distribuciones son similares. En presencia de muchos empates (valores idénticos en los datos), se deben hacer ajustes para calcular correctamente el estadístico de la prueba.

```
wilcox.test(p53 ~ cond, alternative="two.sided", data=dp53)
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  p53 by cond
## W = 41, p-value = 0.1475
## alternative hypothesis: true location shift is not equal to 0
```

Mucha gente dice «usaré una prueba de Wilcoxon para comparar las medias». Pues bien, la prueba de Wilcoxon no es una prueba de comparación de medias. A menudo ni siquiera es una prueba de medianas, a menos que se supongan varias cosas sobre los datos. La prueba de Wilcoxon puede rechazar la nulidad incluso si las medianas son iguales, y la prueba de Wilcoxon puede no rechazar la nulidad incluso si las medianas difieren.

```
## Will accept, means differ
x <- c(rep(10, 1000), 1e9, rep(1000, 1000))
y <- c(rep(10, 1000), -1e9, rep(1000, 1000))
summary(x)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 1.000e+01 1.000e+01 1.000e+03 5.003e+05 1.000e+03 1.000e+09
```

```
summary(y)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.
## -1.000e+09  1.000e+01  1.000e+01 -4.992e+05  1.000e+03
##       Max.
##  1.000e+03
```

```r
wilcox.test(x, y)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  x and y
## W = 2004001, p-value = 0.9496
## alternative hypothesis: true location shift is not equal to 0
```

```r
## Will reject, medians the same
x <- c(rep(10, 1000), 11, rep(12, 1000))
y <- c(rep(10, 1000), 11, rep(13, 1000))
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      10      10      11      11      12      12
```

```r
summary(y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.0    10.0    11.0    11.5    13.0    13.0
```

```r
wilcox.test(x, y)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  x and y
## W = 1502001, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

¿Qué prueba el test de Wilcoxon? Como dice la entrada de Wikipedia, «la hipótesis nula de que, para valores X e Y seleccionados aleatoriamente de dos poblaciones, la probabilidad de que X sea mayor que Y es igual a la probabilidad de que Y sea mayor que X».

Resumen: no utilices un Wilcoxon esperando que pruebe diferencias de medias. Y a continuación haremos hincapié en otro mensaje relacionado: no utilices un Wilcoxon por algún temor mal motivado a utilizar la prueba t.

XVII.2.2. Wilcoxon signed-rank test: matched-pairs or single sample test

El Wilcoxon signed-rank test es una prueba no paramétrica que se utiliza para datos emparejados o para comparar una sola muestra con un valor hipotético. Este

test evalúa si las diferencias dentro de cada par están distribuidas de manera simétrica en torno a un valor central, generalmente cero.

Este test requiere datos en una escala de intervalo, ya que el cálculo de diferencias dentro de los pares supone que las distancias entre valores son significativas y constantes. La hipótesis subyacente supone que las diferencias dentro de los pares están distribuidas simétricamente. Además, las observaciones deben ser independientes entre pares.

La hipótesis nula es que las diferencias dentro de los pares se distribuyen simétricamente en torno a un valor especificado (generalmente cero). La hipótesis alternativa es que las diferencias no se distribuyen simétricamente en torno al valor especificado, o lo hacen en torno a un valor distinto. Dicho de otro modo, si rechazamos la nulidad, podríamos estar rechazándola por diferentes razones (asimetría, simetría en torno a un valor distinto del especificado por la nulidad), o combinaciones de esas razones.

En este test, primero se calculan las diferencias, restando los valores dentro de cada par. Después, se asignan rangos absolutos, ignorando el signo de las diferencias y ordenando los valores absolutos. A continuación se reasigna a cada rango el signo de la diferencia original. Se calculan dos sumas, una para los rangos positivos y otra para los negativos. El estadístico de prueba se basa en la suma de los rangos, evaluando si es suficientemente extremo para rechazar la hipótesis nula.

Un ejemplo de aplicación es un diseño antes-después. En un diseño donde se mide a los participantes antes y después de una intervención, las diferencias entre las medidas suelen ser el foco del análisis. Aunque estas diferencias suelen ser simétricas incluso si el promedio cambia, siempre es útil inspeccionar la simetría con gráficos.

```
wilcox.test(myc.nc, myc.cancer, alternative = 'two.sided', paired = TRUE)
```

```
##
##  Wilcoxon signed rank exact test
##
## data:  myc.nc and myc.cancer
## V = 72, p-value = 0.006836
## alternative hypothesis: true location shift is not equal to 0
```

XVII.2.3. Una mala forma de elegir entre un procedimiento paramétrico y no paramétrico

Una práctica inapropiada pero común es realizar un test de normalidad en los datos y, en función de los resultados, decidir si usar un procedimiento paramétrico (como el test de la t) o uno no paramétrico (como el test de Wilcoxon). Por ejemplo: (I) si el test de normalidad no rechaza la hipótesis nula (los datos son consistentes con la normalidad), entonces se procede con el test paramétrico; (II) si el test de normalidad rechaza la hipótesis nula (los datos no son normales), entonces se elige un test no paramétrico.

Hay varios problemas con esta aproximación. El primero es que el test de normalidad tiene una potencia insuficiente. Los tests de normalidad tienen una potencia limitada, especialmente con tamaños de muestra pequeños. Esto significa que pueden no detectar desviaciones significativas de la normalidad en esos casos, lo que podría llevar al uso indebido de un test paramétrico cuando no es adecuado. Además, puede dar lugar a errores en la interpretación. Si el test de normalidad no detecta una desviación de la normalidad, esto no garantiza que los datos sean normales; simplemente indica que no hay suficiente evidencia para rechazar la normalidad. Y si se rechaza la normalidad, esto no significa automáticamente que el test no paramétrico sea la mejor opción, ya que otros supuestos (como independencia) también podrían estar en juego. Finalmente, da lugar a inconsistencias en la toma de decisiones: cambiar el procedimiento analítico basado en los resultados de un test de normalidad introduce un sesgo en el análisis. Este enfoque puede aumentar el error de tipo I (rechazar una hipótesis nula verdadera) o tipo II (no rechazar una hipótesis nula falsa).

Hay otras alternativas más adecuadas:

- **Evaluar los supuestos con gráficos:** Inspecciona visualmente la distribución de los datos usando histogramas, gráficos de caja y diagramas Q-Q. Estas herramientas permiten identificar problemas evidentes, como asimetría o colas pesadas, que podrían invalidar los métodos paramétricos.
- **Confiar en la robustez de los tests paramétricos:** Los tests paramétricos, como el test de la t, son sorprendentemente robustos a desviaciones moderadas de la normalidad, especialmente cuando el tamaño de la muestra es mayor a 30, gracias al Teorema del Límite Central.
- **Considerar directamente métodos no paramétricos:** Si tienes razones para sospechar que los datos no cumplen con los supuestos paramétricos (como observaciones extremas o una distribución claramente no normal), usar directamente un procedimiento no paramétrico puede ser más apropiado.

En conclusión, usar un test de normalidad como criterio decisivo para elegir entre un análisis paramétrico o no paramétrico no es un enfoque recomendado debido a sus limitaciones inherentes. En su lugar, utiliza un enfoque más holístico, considerando tanto la robustez de los métodos como las características específicas de tus datos.

XVII.2.4. Wilcoxon's paired test and interval data

La distinción clave entre la prueba de Wilcoxon para dos muestras independientes y su versión emparejada radica en el tratamiento de los datos y el tipo de escala de medida requerida.

Para la prueba de dos muestras, basta con tener datos ordinales porque el método implica clasificar todas las observaciones juntas y comparar los rangos entre los grupos. Estos rangos siguen siendo consistentes bajo cualquier transformación monotónica de los datos (por ejemplo, logarítmica, exponencial). Por lo tanto, la prueba es resistente a los cambios en la escala de medición, siempre que se mantenga el orden de los valores.

En cambio, la prueba por pares de Wilcoxon opera sobre las **diferencias dentro de los pares**. Esto requiere datos de intervalo porque el método asume que las

diferencias entre observaciones emparejadas son significativas y pueden clasificarse adecuadamente. Si aplicáramos una transformación monotónica no lineal a los datos originales, las diferencias entre pares podrían cambiar de forma que afectaran a su clasificación. Por ejemplo, una transformación logarítmica podría alterar de forma desproporcionada las diferencias pequeñas en comparación con las grandes, distorsionando así los resultados de la prueba pareada.

Así pues, la prueba por parejas depende fundamentalmente de la escala de los datos para garantizar que las diferencias dentro de las parejas conserven su significado y puedan analizarse correctamente. Este requisito no se aplica a la prueba de dos muestras, ya que tiene en cuenta la clasificación general de las observaciones individuales en lugar de las diferencias entre pares.

Esta distinción pone de relieve la importancia de comprender los supuestos subyacentes y los requisitos de los métodos estadísticos para aplicarlos adecuadamente.

```
## Without logs
wilcox.test(dmyc$myc[1:12], dmyc$myc[13:24], paired = TRUE)

##
##  Wilcoxon signed rank exact test
##
## data:  dmyc$myc[1:12] and dmyc$myc[13:24]
## V = 6, p-value = 0.006836
## alternative hypothesis: true location shift is not equal to 0

## After taking logs
wilcox.test(log(dmyc$myc[1:12]), log(dmyc$myc[13:24]), paired = TRUE)

##
##  Wilcoxon signed rank exact test
##
## data:  log(dmyc$myc[1:12]) and log(dmyc$myc[13:24])
## V = 9, p-value = 0.01611
## alternative hypothesis: true location shift is not equal to 0
```

XVII.3. Simetría y el test de la t emparejado

En la prueba t pareada, la simetría de las diferencias entre sujetos ($W = U - V$) es crítica porque la prueba asume que estas diferencias se muestrean a partir de una población con una distribución aproximadamente normal. Se trata de un supuesto clave para la validez de la prueba t, ya que afecta directamente al cálculo del estadístico de la prueba y a su comparación con la distribución t.

La prueba t apareada se centra en las diferencias (W) en lugar de las distribuciones originales de (U) ("después") y (V) ("antes"). Por lo tanto, la relevancia de las distribuciones de U y V radica en cómo se combinan para formar W: Si U y V

tienen distributiones similares (por ejemplo, formas, medias o varianzas parecidas), W tiene más probabilidaes de ser simétrico alrededor de su media. Si U y V difieren sustancialmente en forma, escala o dispersión, W podría ser asimétrico, lo que viola el supuesto de simetría de la prueba t apareada.

Aunque U y V no sean normalmente distribuidos, las diferencias W podrían aproximarse a la normalidad debido al **teorema del límite central**, especialmente si el tamaño de muestra es razonablemente grande. Sin embargo, si U y V tienen distribuciones muy no normales o colas pesadas, W podría no ser suficientemente normal para que la prueba t sea válida.

Las diferencias W reflejan los cambios intraindividuales entre condiciones (por ejemplo, "después" frente a "antes"). La prueba t apareada evalúa si estos cambios, en promedio, son significativamente diferentes de cero. Si W es notablemente asimétrico o multimodal, podría indicar heterogeneidad en el efecto que se está midiendo.

XVII.4. Datos no independientes

Los datos emparejados no son independientes: se asocian a través del sujeto, o id. Existen otras formas de dependencia, siendo la más común la toma de múltiples medidas por sujeto.

En este caso, la unidad de observación son sujetos, no las medidas. Por ello, realizar un test de la t es erróneo (mira los grados de libertad):

```
t.test(brca2 ~ cond, data = dbrca)


##
##  Welch Two Sample t-test
##
## data:  brca2 by cond
## t = -2.1969, df = 28.061, p-value = 0.03645
## alternative hypothesis: true difference in means between group Cancer and group
## 95 percent confidence interval:
##  -3.9309162 -0.1377338
## sample estimates:
## mean in group Cancer     mean in group NC
##             5.953208             7.987533
```

Capítulo XVIII

Modelos lineares: ANOVA, regresión, ANCOVA

XVIII.1. Introducción a los modelos lineares

En un test apareado, el modelo puede representarse de la siguiente manera:

$$Expression.of.MYC = function(subject\ and\ condition) + \varepsilon$$

Este modelo describe cómo se distribuyen los valores observados (en este caso, la expresión de MYC) en función de los factores que afectan la medición, como el sujeto y la condición experimental, junto con un término de error estadístico (ε) que captura la variabilidad no explicada por estos factores. Para simplificarlo, se puede expresar como:

$$Expression.of.MYC = effect.of.subject + effect.of.condition + \varepsilon$$

Los componentes del modelo son:

- **Efecto del sujeto:** Este término representa las diferencias inherentes entre los sujetos en la población. Por ejemplo, algunos sujetos podrían tener una mayor o menor expresión basal de MYC debido a factores biológicos individuales.
- **Efecto de la condición:** Este término captura las diferencias causadas por las condiciones experimentales, como un tratamiento o un cambio en las circunstancias (por ejemplo, "antes" frente a "después", o "cáncer" frente a "control no canceroso").
- **Error estadístico:** Representa la variabilidad no explicada por los efectos del sujeto o la condición. Esto incluye mediciones imprecisas, factores no modelados o ruido inherente en los datos.

El modelo es apropiado para un diseño apareado porque considera explícitamente el efecto del sujeto. Al realizar el análisis sobre las diferencias intra-sujeto, se elimina el efecto individual del sujeto, dejando únicamente el efecto de la condición y el error. Este

enfoque aumenta la potencia estadística del análisis al reducir la variabilidad atribuible a las diferencias entre sujetos.

Por tanto, el análisis apareado se centra en las diferencias entre las condiciones dentro de cada sujeto, aislando el efecto que queremos evaluar (en este caso, el efecto de la condición experimental sobre la expresión de MYC).

```
LinearModel.1 <- lm(myc ~ id + cond, data = dmyc)
summary(LinearModel.1)
```

```
##
## Call:
## lm(formula = myc ~ id + cond, data = dmyc)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6173 -0.2224  0.0000  0.2224  0.6173
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   24.0393     0.4147  57.961 4.98e-15 ***
## idbqysitlvpm  14.4530     0.5635  25.647 3.66e-11 ***
## iddchlnopykg  -1.7155     0.5635  -3.044  0.01116 *
## idefzpcboidt -23.5210     0.5635 -41.739 1.82e-13 ***
## idejmkobsqrh   5.5030     0.5635   9.765 9.37e-07 ***
## idhyqjownkue  34.4060     0.5635  61.054 2.82e-15 ***
## idmculjayvhw -18.3985     0.5635 -32.649 2.65e-12 ***
## idqsmyexkcnw  29.4820     0.5635  52.316 1.53e-14 ***
## idtrsyacmejh   7.1490     0.5635  12.686 6.56e-08 ***
## iduhbkifsnvw  19.7470     0.5635  35.041 1.23e-12 ***
## idytwgsplaef  29.0515     0.5635  51.553 1.80e-14 ***
## idzuhxmiyfos  52.7660     0.5635  93.634  < 2e-16 ***
## condNC         0.9384     0.2301   4.079  0.00182 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5635 on 11 degrees of freedom
## Multiple R-squared:  0.9997, Adjusted R-squared:  0.9993
## F-statistic:  2840 on 12 and 11 DF,  p-value: < 2.2e-16
```

```
t.test(myc.nc, myc.cancer, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  myc.nc and myc.cancer
## t = 4.079, df = 11, p-value = 0.001823
## alternative hypothesis: true mean difference is not equal to 0
```

```
## 95 percent confidence interval:
##  0.432056 1.444777
## sample estimates:
## mean difference
##       0.9384167
```

XVIII.2. ANOVAs

Los modelos lineales y sus extensiones (que incluyen la regresión logística, pero también el análisis de supervivencia, muchos problemas de clasificación, modelos no lineales, análisis de experimentos, tratamiento de muchos tipos de datos dependientes, etc.) son un tema fundamental de la estadística. Aquí sólo arañaremos la superficie, pero estos métodos son extremadamente potentes y flexibles y pueden utilizarse para abordar una enorme variedad de cuestiones de investigación diferentes. ANOVAs, regresión, ANCOVAs, son sólo tipos especiales de modelos lineales.

XVIII.2.1. ANOVA: teoría y ejemplos prácticos

Se diferencia mean square (MS) between de MS within. MS_B mide la variabilidad entre los grupos, y se calcula como la suma de cuadrados entre grupos (mide qué tan lejos están las medias de cada grupo de la media global) dividido por los grados de libertad entre grupos.

$$MS_B = \frac{SSB}{dfb} = \frac{\sum_{j=1}^{k}(\bar{X}_j - \bar{X})^2}{n-k}$$

MS_W mide la variabilidad dentro de los grupos. Se calcula como la suma de cuadrados dentro de grupos (mide qué tan dispersos están los datos dentro de cada grupo) dividido por los grados de libertad dentro de grupos.

$$MS_W = \frac{SSW}{dfw} = \frac{\sum_{j=1}^{k}\sum_{j=1}^{l}(X - \bar{X}_j)^2}{k-1}$$

La razón F es el cociente MS_B/MS_W. Si es grande, indica que hay más variación entre grupos que dentro de los grupos. Así, nos ayuda a determinar si las diferencias entre grupos son estadísticamente significativas. Si la hipótesis nula es cierta, MS_B y MS_W estiman lo mismo y F es 1. Si no es cierta, F debería ser más grande de 1, ya que MS_B debería ser más grande que MS_W.

Queremos ver si la hora del ejercicio marca alguna diferencia. La realización de tres pruebas t no es el mejor camino a seguir aquí: nuestra hipótesis nula global es $\mu_{Madrugada} = \mu_{Almuerzo} = \mu_{Tarde}$ y eso es lo que ANOVA nos permitirá probar directamente.

```
AnovaMIT <- aov(activ ~ ftraining, data = mit)
summary(AnovaMIT)

##             Df Sum Sq Mean Sq F value  Pr(>F)
## ftraining    2  31.15   15.57   22.89 1.7e-07 ***
## Residuals   43  29.26    0.68
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

De las dos filas, la que nos interesa es la del efecto (ftraining). La columna Df indica los grados de libertad. Las dos columnas Sum Sq (Suma de cuadrados) y Mean Sq (Cuadrados medios). La suma de cuadrados es una cantidad relacionada con la varianza. La media cuadrática se obtiene a partir de la relación entre la suma cuadrática y Df. A continuación, utilizamos la Sq Media para comparar cuánta varianza hay entre los grupos en relación con la varianza dentro de los grupos: el valor F es el cociente de la Sq Media de la formación sobre la Sq Media de los residuos. Cuanto mayor sea el valor F, mayor será la evidencia de que los grupos son diferentes.

Aquí hemos utilizado aov, pero también se podría utilizar `lm` y mostrar el resumen con anova.

```
LinearModel.1 <- lm(activ ~ ftraining, data = mit)
Anova(LinearModel.1, type="II")

## Anova Table (Type II tests)
##
## Response: activ
##           Sum Sq Df F value    Pr(>F)
## ftraining 31.147  2  22.887 1.704e-07 ***
## Residuals 29.260 43
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

XVIII.2.2. Intervalos de confianza para los parámetros del modelo

Se puede realizar mediante:

```
confint(AnovaMIT)

##                          2.5 %    97.5 %
## (Intercept)          1.6014112 2.6045888
## ftrainingLunch      -0.5985849 0.7902515
## ftrainingAfternoon   1.0844974 2.3041983
```

A veces `confint` nos dará mucha información. Pero a menudo no será fácil relacionarla con nuestra pregunta científica original. Una de las razones es que los parámetros reales del modelo ajustado dependen, bueno, de la parametrización. Así que, a menudo, vamos a querer preguntar explícitamente «¿Qué medias son diferentes», y eso es lo que hacemos a continuación.

XVIII.2.3. Medias diferentes - comparación múltiple

El p-valor bajo nos llevó a rechazar la hipótesis nula $\mu_{Madrugada} = \mu_{Almuerzo} = \mu_{Tarde}$. Así, hay fuerte evidencia de que las tres medias no son iguales. ¿Pero cuál es diferente de las demás?

```
numSummary(mit$activ , groups = mit$ftraining, statistics = c("mean", "sd"))
```

```
##                mean        sd data:n
## Morning   2.103000 0.8113702     11
## Lunch     2.198833 0.6608995     12
## Afternoon 3.797348 0.9013205     23
```

Esto se puede obtener también con aggregate:

```
with(mit, aggregate(activ, list(Training = ftraining),
                    function(x) c(mean = mean(x),
                                  sd = sd(x),
                                  n = sum(!is.na(x)))
                    ))
```

```
##    Training     x.mean       x.sd        x.n
## 1   Morning  2.1030000  0.8113702 11.0000000
## 2     Lunch  2.1988333  0.6608995 12.0000000
## 3 Afternoon  3.7973478  0.9013205 23.0000000
```

La comparación de todos los pares de medias se realiza mediante el modelo de ANOVA, por lo que los resultados no son idénticos al compararlos a los tests de la t. Además, es necesario realizar corrección del testeo múltiple al realizar tres tests distintos.

```
library(multcomp) ## for glht
```

```
## Cargando paquete requerido: mvtnorm
## Cargando paquete requerido: survival
## Cargando paquete requerido: TH.data
## Cargando paquete requerido: MASS
##
## Adjuntando el paquete: 'TH.data'
```

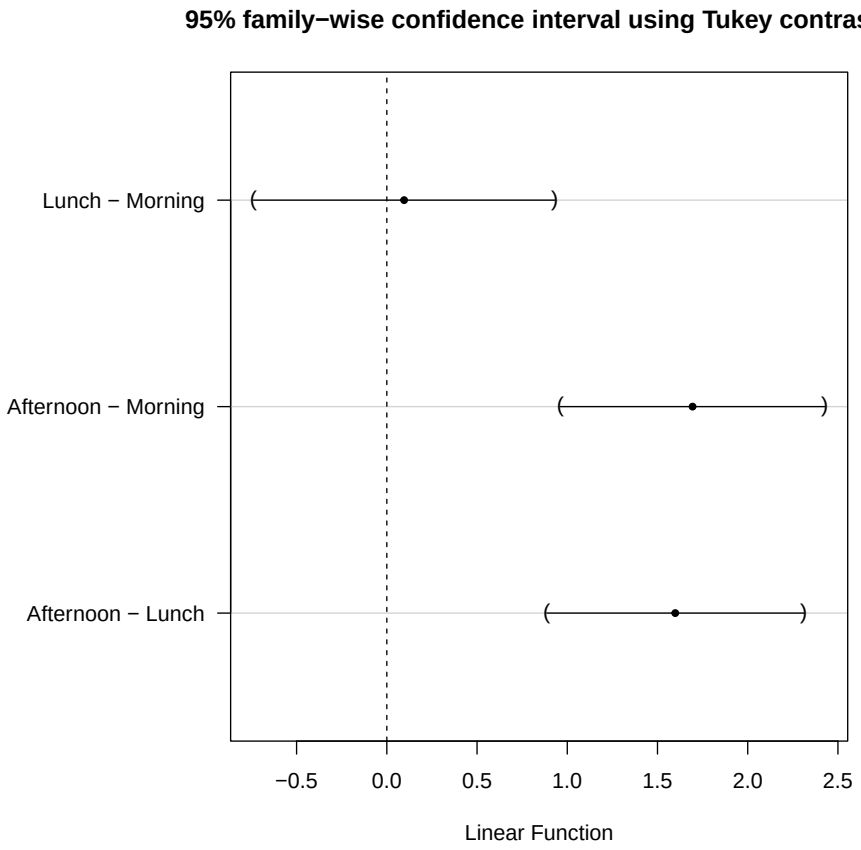
```
## The following object is masked from 'package:MASS':
##
##     geyser

## The next two lines carry out the multiple comparisons and the
## ones below plot them
Pairs <- glht(AnovaMIT, linfct = mcp(ftraining = "Tukey"))
summary(Pairs) # pairwise tests

##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: aov(formula = activ ~ ftraining, data = mit)
##
## Linear Hypotheses:
##                            Estimate Std. Error t value Pr(>|t|)
## Lunch - Morning == 0        0.09583    0.34434   0.278    0.958
## Afternoon - Morning == 0    1.69435    0.30240   5.603   <1e-04
## Afternoon - Lunch == 0      1.59851    0.29375   5.442   <1e-04
##
## Lunch - Morning == 0
## Afternoon - Morning == 0 ***
## Afternoon - Lunch == 0   ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)

confint(Pairs) # confidence intervals

##
##   Simultaneous Confidence Intervals
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: aov(formula = activ ~ ftraining, data = mit)
##
## Quantile = 2.4238
## 95% family-wise confidence level
##
##
## Linear Hypotheses:
##                            Estimate lwr      upr
## Lunch - Morning == 0        0.09583 -0.73877  0.93044
## Afternoon - Morning == 0    1.69435  0.96138  2.42731
## Afternoon - Lunch == 0      1.59851  0.88651  2.31052
```

Figura XVIII.1: *Plot of pairwise differences with Tukey contrasts*

```
cld(Pairs) # compact letter display
old.oma <- par(oma = c(0,5,0,0))
plot(confint(Pairs))
par(old.oma) ## restore graphics windows settings
```

Fíjate bien en el gráfico de la figura: para cada diferencia (para cada contraste), muestra la estimación y un intervalo de confianza del 95 % a su alrededor. El título del gráfico dice "95 % intervalo de confianza familiar", y que indica que la corrección de pruebas múltiples se ha utilizado. (Se puede hacer aún más explícito mediante el uso de un título como "95 % intervalo de confianza por familias utilizando contrastes de Tukey").

```
.Pairs <- glht(AnovaMIT, linfct = mcp(ftraining = "Tukey"))
tmp <- cld(.Pairs) ## silent assignment
old.oma <- par(oma=c(0,5,0,0))
plot(confint(.Pairs),
     main = "95% family-wise confidence interval using Tukey contrasts")
```




```
par(old.oma) ## restore graphics windows settings
```

Según la gráfica, se puede rechazar la hipótesis de que Tarde-Mañana y de que Tarde-Almuerzo sean iguales.

En resumen, el ANOVA en un sentido funciona de la siguiente forma: se obtienen los datos, se recodifica el factor (la variable independiente) en caso de ser necesario, se ejecuta el modelo y se obtienen los diagnósticos del modelo. Finalmente se realizan las comparaciones entre los pares de las medias con el ajuste apropiado para la comparación múltiple.

XVIII.3. Comparación múltiple: FWER y FDR

XVIII.3.1. Family-wise error rate (FWER)

En el caso de comparaciones múltiples, como en el ejemplo anterior donde se realizaron tres comparaciones, el principal problema es el riesgo de rechazar una hipótesis nula cuando en realidad es verdadera (error tipo I). Cuando se evalúan múltiples contrastes, la probabilidad de cometer al menos un error tipo I aumenta significativamente por encima del nivel de significancia deseado, por ejemplo, el 5 %.

El Family-Wise Error Rate (FWER) controla la probabilidad de cometer al menos un error tipo I en el conjunto completo de pruebas (la "familia" de comparaciones).

Para lograrlo: (1) Los procedimientos ajustan los p-valores individuales o ensanchan los intervalos de confianza para mantener el control global del error tipo I en la familia de pruebas, y (2) A medida que aumenta el número de pruebas, los ajustes son más estrictos, haciendo más difícil rechazar una hipótesis nula.

En procedimientos como Bonferroni, el nivel de significancia para cada prueba individual se ajusta dividiendo el nivel deseado (α) por el número total de pruebas. El método Tukey adapta los intervalos de confianza para realizar comparaciones por pares, asegurando que el error tipo I no exceda el límite especificado en el conjunto completo de comparaciones.

	H_0 not rejected	H_0 rejected
Means do not differ (H_0 *true*)	U	V
Means differ (H_0 *false*)	T	S

En este contexto:

- U: casos donde H_0 es verdadera y no se rechaza
- V: casos donde H_0 es verdadera y se rechaza (error tipo I): por ejemplo, V = 2 cuando rechazo dos hipótesis nulas cuando no debo.
- T: casos donde H_0 es falsa y no se rechaza (error tipo II)
- S: casos donde H_0 es falsa y se rechaza correctamente.

Para tres pruebas, $U + V + T + S = 3$, ya que el número total de hipótesis es igual al número de pruebas realizadas.

El objetivo de procedimientos como Tukey, Bonferroni, entre otros, es garantizar que la probabilidad de cometer al menos un error tipo I ($V \geq 1$) sea menor o igual a un valor especificado, generalmente $\alpha = 0.05$. Esto significa que se controla estrictamente la probabilidad de cometer un error dentro de la familia de pruebas.

La idea intuitiva del FWER es: "Quiero minimizar la probabilidad de rechazar falsamente cualquier hipótesis nula". En términos probabilísticos: controlar que $Pr(V \geq 1)$ esté por debajo de un valor determinado, como 0.05.

Aunque es un enfoque conservador, su rigidez puede limitar la capacidad de detectar efectos verdaderos, especialmente cuando se realizan muchas comparaciones, ya que aumenta la probabilidad de cometer errores tipo II. Por ello, en algunos casos, se prefieren métodos menos estrictos, como el control de la tasa de descubrimientos falsos (FDR).

Ten en cuenta que, en nuestro uso de Tukey, no preespecificamos el $Pr(V \geq 1)$ real. El procedimiento se ejecuta, y nos da "valores p ajustados". **El valor P ajustado para una hipótesis particular dentro de una colección de hipótesis, entonces, es el menor nivel de significación global (es decir, 'experimentalmente') al que se rechazaría la hipótesis particular.** Un valor P ajustado puede compararse directamente con cualquier nivel de significación α elegido: Si el valor P ajustado es menor o igual que α, se rechaza la hipótesis.

XVIII.3.2. False descovery rate (FDR)

Existe un enfoque diferente para el problema de las pruebas múltiples. En este enfoque nos centramos en controlar la fracción de falsos positivos. El número total de hipótesis nulas que rechazamos es $V + S$. La idea intuitiva detrás del control de la tasa de falsos descubrimientos (**FDR**) es acotar (establecer un límite superior) la proporción $\frac{V}{V+S}$[1].

Una diferencia clave es que el FDR puede mantenerse razonablemente bajo (digamos, 0,01) incluso cuando es casi seguro que $V \geq 1$. ¿Cuándo puede ocurrir esto? Por ejemplo, cuando realizamos decenas de miles de pruebas de hipótesis. De nuevo, la FDR controlará la fracción de falsos descubrimientos, mientras que el control de la tasa de error por familia (FWER) hace hincapié en que V no se convierta en 1 o más.

Al igual que hicimos con Tukey y los procedimientos FWER, en general no especificamos previamente el nivel de FDR que queremos alcanzar, sino que obtenemos "valores p ajustados". La diferencia en el significado de "ajustados" es que ahora estos valores p están ajustados para FDR (no ajustados para el control de la tasa de error por familia). Así, cuando tratamos con FDR, el p-valor ajustado de una hipótesis individual es el nivel más bajo de FDR para el que la hipótesis se incluye por primera vez en el conjunto de hipótesis rechazadas.

La FDR suele emplearse en procedimientos de cribado o screening, en los que estamos dispuestos a permitir algunos descubrimientos falsos, porque estamos cribando miles de hipótesis. El coste de exigir $V = 0$ sería pasar por alto muchos descubrimientos. ¿Un ejemplo? Supongamos que se ha medido la expresión de 20.000 genes en dos grupos de sujetos, algunos con cáncer de colon y otros no. Ahora puedes hacer el equivalente a 20.000 pruebas t. Así que obtendrás 20.000 valores p, y querrás ajustar esas 20.000 pruebas para pruebas múltiples.

Un ejemplo sencillo y con cuatro p-valores. Supongamos que hemos realizado un procedimiento de cribado, probando cuatro genes. Se obtienen los p-valores que muestro a continuación. Para utilizar un método de corrección FDR se puede utilizar `p.adjust` con el argumento `method = "BH"` (BH es uno de los varios tipos posibles de corrección FDR). Para mostrar lo que sucede, se combinan los dos, uno al lado del otro, para que se pueda ver el valor p original y el ajustado FDR.

```
p.values <- c(0.001, 0.01, 0.03, 0.05)
adjusted.p.values <- p.adjust(p.values, method = "BH")
cbind(p.values, adjusted.p.values)
```

```
##      p.values adjusted.p.values
## [1,]    0.001              0.004
## [2,]    0.010              0.020
## [3,]    0.030              0.040
## [4,]    0.050              0.050
```

[1] Hay varios enfoques diferentes. El más común es controlar $FDR = E(Q)$ donde $Q = V/(V+S)$ si $V + S > 0$ (y $Q = 0$ en caso contrario). Pero hay otros, como el $pFDR$, etc.

Por ejemplo, si mantenemos como "significativos" todos los genes con un p-valor (no p-valor ajustado, sino p-valor, así que los tres primeros últimos) $\leq 0,030$, el FDR (el número esperado de falsos descubrimientos) será 0,040 (el p-valor ajustado FDR para el gen con $p-$valor de 0,03).

XVIII.3.3. Comparación múltiple: ejemplos

Cuando se realizan muchas pruebas, algunas de ellas pueden tener valores p bajos por casualidad, por lo que es necesario realizar ajustes. Si cualquier gen con un valor p bajo se declara significativo (independientemente del tamaño de la colección de pruebas) es probable que se empiece a afirmar que muchos resultados puramente casuales son "significativos".

Recuerda que ocurren sucesos muy raros, y que es casi seguro que ocurran si el experimento se repite muchas veces (por cierto, por eso la mayoría de nosotros no tememos morir por un rayo, aunque cada año algunas personas mueran de hecho por un rayo).

Cuando se examinan 20.000 genes, se está realizando 20.000 veces el experimento del valor p y la hipótesis nula. Y recuerda las reglas: para una hipótesis nula verdadera, la probabilidad de encontrar un valor p $\leq 0,05$ es $0,05$. Ahora imagina que haces eso 20000 veces; es casi seguro que tendrás muchos p-valores $\leq 0,05$. (Lo mismo con los rayos: aunque las probabilidades de morir por un rayo sean $\leq \frac{1}{300000}$, con millones de personas en la tierra, es casi seguro que algunas morirán por un rayo).

XVIII.4. Two-way ANOVA (ANOVA de dos factores)

En un análisis de varianza de dos factores (Two-Way ANOVA), se evalúan simultáneamente dos variables predictoras (o factores) para determinar si tienen un efecto significativo en la variable de respuesta. Además, este análisis permite examinar si existe una interacción entre los dos factores.

Un aspecto clave de este modelo es la interacción entre los factores (no aditividad). La interacción ocurre cuando el efecto de un factor en la variable de respuesta depende del nivel del otro factor. En otras palabras, el impacto combinado de ambos factores no es simplemente la suma de sus efectos individuales.

Cuando no hay interacción entre los factores, los efectos de cada factor son independientes. Esto significa que el efecto de moverse entre filas (niveles de un factor) es constante, independientemente de la columna en la que te encuentres (niveles del otro factor). De manera similar, el efecto de moverse entre columnas es independiente del nivel de las filas. En este caso, los efectos pueden describirse de manera simplificada a través de los promedios marginales de las filas y columnas.

Cuando hay interacción entre los factores, el efecto de un factor cambia según el nivel del otro factor. El resultado no puede resumirse simplemente con los promedios marginales, ya que la relación entre las filas y columnas depende de la celda específica

donde te encuentres. Para describir completamente el modelo, es necesario especificar los valores en cada celda de la tabla de combinaciones de factores.

Ejemplos de interacción en la vida real:

- Genética (epistasis): En genética, la interacción entre dos genes (o loci) puede influir en un fenotipo. El efecto de un gen en el rasgo observado puede depender de la presencia o ausencia de un segundo gen.
- Medicina: Un tratamiento puede tener diferentes efectos en hombres y mujeres. Por ejemplo, la eficacia de un medicamento (factor 1) puede depender del género del paciente (factor 2).
- Marketing: El efecto de una promoción (factor 1) puede variar según la región geográfica (factor 2). Por ejemplo, un descuento puede ser más efectivo en áreas urbanas que rurales.

```
set.seed(3)
df1 <- data.frame(y = runif(8),
                  A = rep(c("a1", "a2"), 4),
                  B = rep(c("b1", "b1", "b2", "b2"), 2))


df1
```

```
##           y  A  B
## 1 0.1680415 a1 b1
## 2 0.8075164 a2 b1
## 3 0.3849424 a1 b2
## 4 0.3277343 a2 b2
## 5 0.6021007 a1 b1
## 6 0.6043941 a2 b1
## 7 0.1246334 a1 b2
## 8 0.2946009 a2 b2
```

XVIII.4.1. Modelo sin interacción (aditivo)

Un modelo aditivo supone que los efectos de los factores son independientes entre sí y no interactúan. Esto significa que el efecto de un factor no depende del nivel del otro. En R, este modelo se ajusta como:

```
m1 <- lm(y ~ A + B, data = df1)
anova(m1)


## Analysis of Variance Table
##
## Response: y
```

```
##           Df   Sum Sq  Mean Sq F value Pr(>F)
## A          1 0.071164 0.071164  1.9312 0.2233
## B          1 0.137850 0.137850  3.7410 0.1109
## Residuals  5 0.184244 0.036849
```

Los términos que representan los efectos de los factores se suman directamente: $y = \mu + A + B + \varepsilon$. Los grados de libertad de cada factor se basan en sus niveles. Para A, con k_A niveles, $df_A = k_A - 1$, y para B, con k_B niveles, $df_B = k_B - 1$

En cuanto a la parametrización del modelo, solo se estiman los parámetros del intercepto (μ) y los efectos individuales de los niveles de A y B. La interpretación de los términos es independiente, ya que no hay interacción. Por ejemplo, con dos niveles en A y B, el modelo tiene 3 parámetros ($\mu, Aa2, Bb2$).

Es importante mencionar que los números hubiesen sido diferentes si en el código hubiésemos antepuesto B a A (`lm(y ~ B + A)`).

Si los p-valores de A o B son pequeños, se concluye que esos factores tienen un efecto significativo en y.

```
summary(m1)
```

```
##
## Call:
## lm(formula = y ~ A + B, data = df1)
##
## Residuals:
##       1       2       3       4       5       6       7
## -0.28316  0.16769  0.19628 -0.04956  0.15090 -0.03544 -0.06403
##       8
## -0.08269
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.4512     0.1176   3.838   0.0121 *
## Aa2           0.1886     0.1357   1.390   0.2233
## Bb2          -0.2625     0.1357  -1.934   0.1109
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.192 on 5 degrees of freedom
## Multiple R-squared:  0.5315, Adjusted R-squared:  0.3441
## F-statistic: 2.836 on 2 and 5 DF,  p-value: 0.1502
```

XVIII.4.2. Modelo con interacción (no aditivo)

Un modelo con interacción incluye un término adicional para considerar los efectos combinados de los factores. En R, se ajusta así:

```r
m2 <- lm(y ~ A * B, data = df1)
anova(m2)
```

```
## Analysis of Variance Table
##
## Response: y
##           Df   Sum Sq  Mean Sq F value Pr(>F)
## A          1 0.071164 0.071164  1.9071 0.2394
## B          1 0.137850 0.137850  3.6942 0.1270
## A:B        1 0.034981 0.034981  0.9374 0.3878
## Residuals  4 0.149262 0.037316
```

Este modelo tiene la forma $y = \mu + A + B + A : B + \varepsilon$. La interacción A:B representa las desviaciones de la aditividad. Si es significativa, el efecto de un factor depende del nivel del otro. Los grados de libertad de la interacción son el producto de los grados de libertad de los factores involucrados: $df_{A:B} = df_A * df_B$

Se estiman parámetros para μ, los efectos individuales (A y B) y las interacciones (A:B). El modelo tiene tantos parámetros como celdas en la tabla factorial.

Un p-valor pequeño para A:B indica una interacción significativa. En este caso, no se interpretan los efectos de A y B por separado. Un modelo con interacción siempre se ajusta mejor, pero es importante evaluar si la mejora es estadísticamente significativa.

```r
summary(m2)
```

```
##
## Call:
## lm(formula = y ~ A * B, data = df1)
##
## Residuals:
##        1        2        3        4        5        6        7
## -0.21703  0.10156  0.13015  0.01657  0.21703 -0.10156 -0.13015
##        8
## -0.01657
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.3851     0.1366   2.819   0.0479 *
## Aa2           0.3209     0.1932   1.661   0.1720
## Bb2          -0.1303     0.1932  -0.674   0.5370
## Aa2:Bb2      -0.2645     0.2732  -0.968   0.3878
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1932 on 4 degrees of freedom
## Multiple R-squared:  0.6204, Adjusted R-squared:  0.3358
## F-statistic:  2.18 on 3 and 4 DF,  p-value: 0.233
```

XVIII.4.3. Ejemplo con múltiples niveles

Supongamos que tenemos un ANOVA de dos vías. El primer factor (T) tiene cuatro niveles ($df_T = 3$), el segundo factor (W) tiene siete niveles ($df_W = 6$). Podemos crear una matriz de 28 celdas (4 filas por cada nivel de T y 7 filas por cada nivel de W).

El modelo aditivo sería $y = \mu + T + W + \varepsilon$, con un total de $1 + 3 + 6 = 10$ parámetros, incluyendo el intercepto.

El modelo con interacción incluiría un término para la interacción: $y = \mu + T + W + T : W + \varepsilon$. Los grados de libertad de la interacción serían $df_{T:W} = df_T * df_W = 3 * 6 = 18$. El total de parámetros es $1+3+6+18 = 28$. Esto significa que el modelo ajusta un parámetro para cada celda de la tabla 4x7.

XVIII.4.4. ANOVA de tres vías

Tenemos tres factores: A con 3 niveles, B con 5 niveles y C con 6 niveles. Los grados de libertad son 2, 4 y 5 respectivamente. Los parámetros son A 2, B 4, C 5, A:B 8, A:C 10, B:C 20, A:B:C 40. Esto último es la interacción entre los tres factores, y se puede leer de varias maneras equivalentes: la interacción A:B cambia con niveles de C, la interacción B:C cambia con niveles de A, la interacción A:C cambia con niveles de B. Al encontrar una interacción significativa ahí, se puede decir que hay interacción entre los tres factores, dejando de lado las demás interacciones.

XVIII.4.5. Data set colesterol

```
## This fits the model. Pay attention to the "*"
cholestanova <- (lm(y ~ Diet*Drug, data=dcholest))
## This shows the ANOVA table. Notice the "Type II"
## And notice we are using function Anova with capital A
## which is a function from the car package.
Anova(cholestanova)

## Anova Table (Type II tests)
##
## Response: y
##           Sum Sq Df F value    Pr(>F)
## Diet      75.453  2  29.949 3.163e-08 ***
## Drug      32.261  1  25.610 1.433e-05 ***
## Diet:Drug 48.979  2  19.441 2.348e-06 ***
## Residuals 42.830 34
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

## Now we are shoum the 3 by 2 table of means, standard deviations, and number
## of observations
tapply(dcholest$y, list(Diet=dcholest$Diet, Drug=dcholest$Drug),
       mean, na.rm=TRUE) # means
```

```
##     Drug
## Diet         A         B
##   HF 1.7280000 -0.588400
##   M1 0.7914286  4.055714
##   M2 2.5685556  5.318250
```

```
tapply(dcholest$y, list(Diet=dcholest$Diet, Drug=dcholest$Drug),
       sd, na.rm=TRUE) # std. deviations
```

```
##     Drug
## Diet         A         B
##   HF 0.5026165 0.4956474
##   M1 0.9572549 0.7641736
##   M2 1.5181591 1.3963718
```

```
tapply(dcholest$y, list(Diet=dcholest$Diet, Drug=dcholest$Drug),
       function(x) sum(!is.na(x))) # counts
```

```
##     Drug
## Diet A B
##   HF 4 5
##   M1 7 7
##   M2 9 8
```

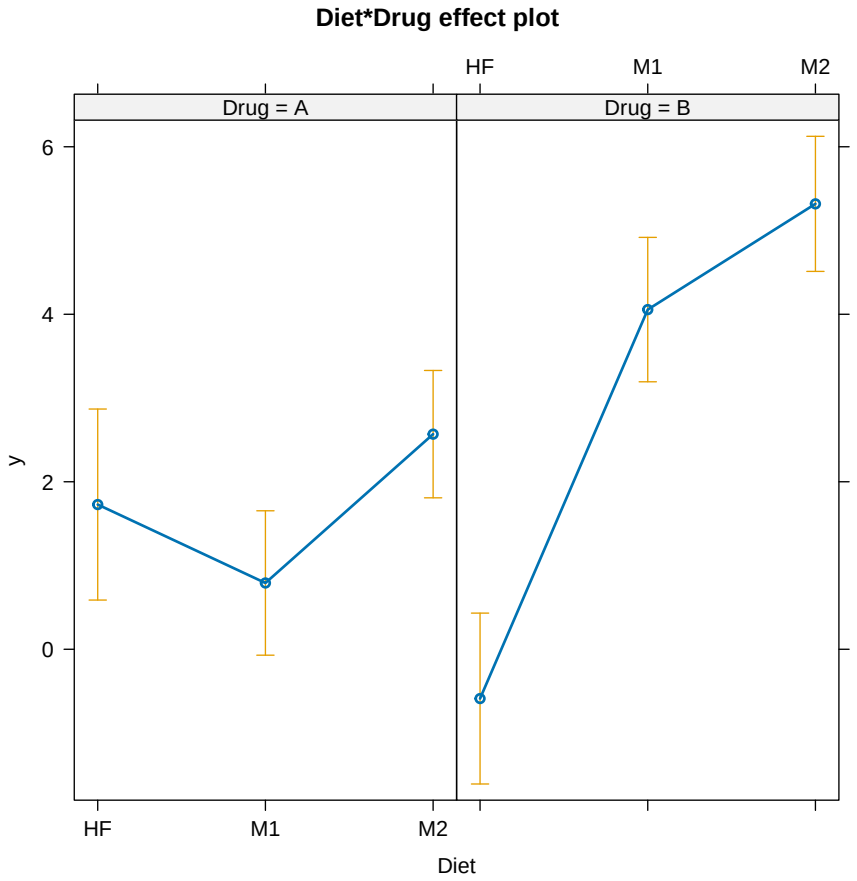
XVIII.4.5.1. `Anova, anova, aov, lm, summary`

En R a menudo podemos utilizar diferentes formas de obtener resultados de un modelo lineal, incluyendo regresión y ANOVA.

- La función `lm` ajusta modelos lineales, incluyendo regresión y ANOVA (la mayoría de ellos, no todos).
- La función `aov` también puede utilizarse para ajustar modelos ANOVA. No la utilizamos para ajustar modelos de regresión. La mayoría de los modelos que se pueden ajustar con `aov` se pueden ajustar con `lm`. En lo que respecta a este curso, su sintaxis es la misma.
- `Anova` y `anova` dan tablas ANOVA de modelos ajustados con `lm`. La principal diferencia entre los dos es que `Anova` da, por defecto, lo que se llama sumas de cuadrados y pruebas de Tipo II, que utilizaremos la mayoría de las veces para tratar cuestiones sobre el orden de los factores. Además, `anova` también se puede utilizar para comparar modelos.
- La función `summary` sobre un objeto ajustado con `aov` también dará una tabla ANOVA. Rara vez usaremos esto (aunque podría ser el código que algunos menús en R commander realmente generan, y se podría ver en el código de otras personas).

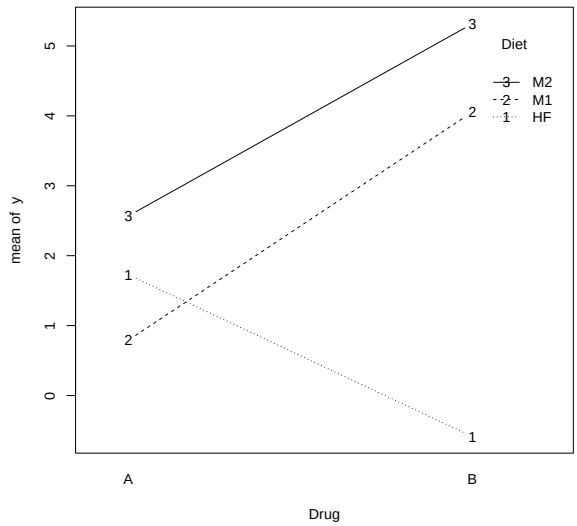
- Función `summary` sobre un objeto equipado con `lm` dará, entre otros, una tabla de coeficientes, no una tabla ANOVA.

```
library(effects)

## lattice theme set by effectsTheme()
## See ?effectsTheme for details.

plot(allEffects(cholestanova), ask = FALSE)
```



Básicamente, una interacción significa que el efecto de una variable depende del efecto de la otra. En este caso, aunque el fármaco B provoque en general un cambio mayor (disminución) del colesterol, sus efectos dependen de la dieta. Esto tiene consecuencias prácticas: ¿es el fármaco B mejor? Depende de la dieta del paciente: para la dieta HF (alta en grasas), el Fármaco B es claramente peor que el Fármaco A.

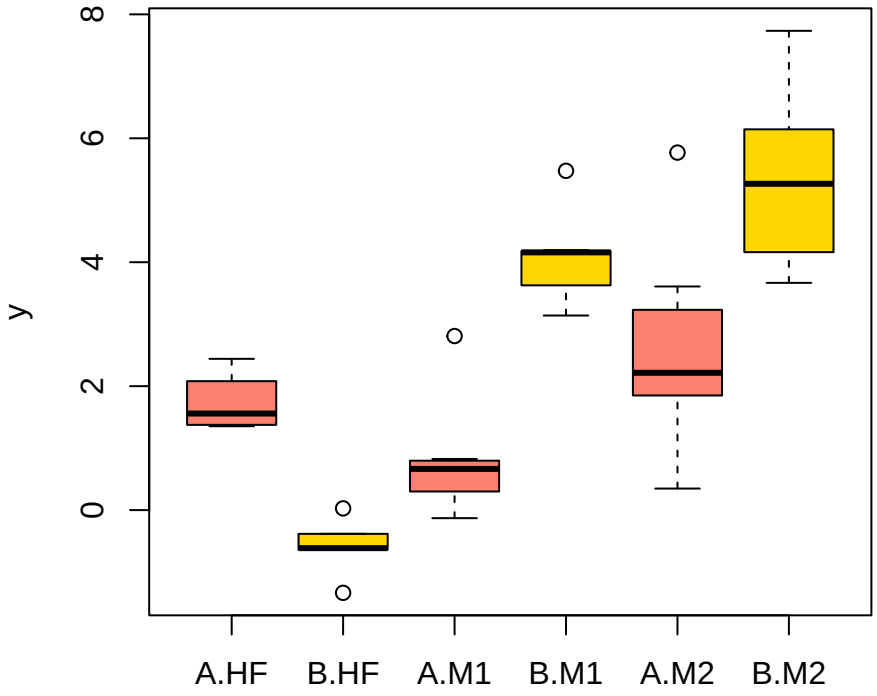
```
with(dcholest, interaction.plot(Drug, Diet, y, type = "b"))
```

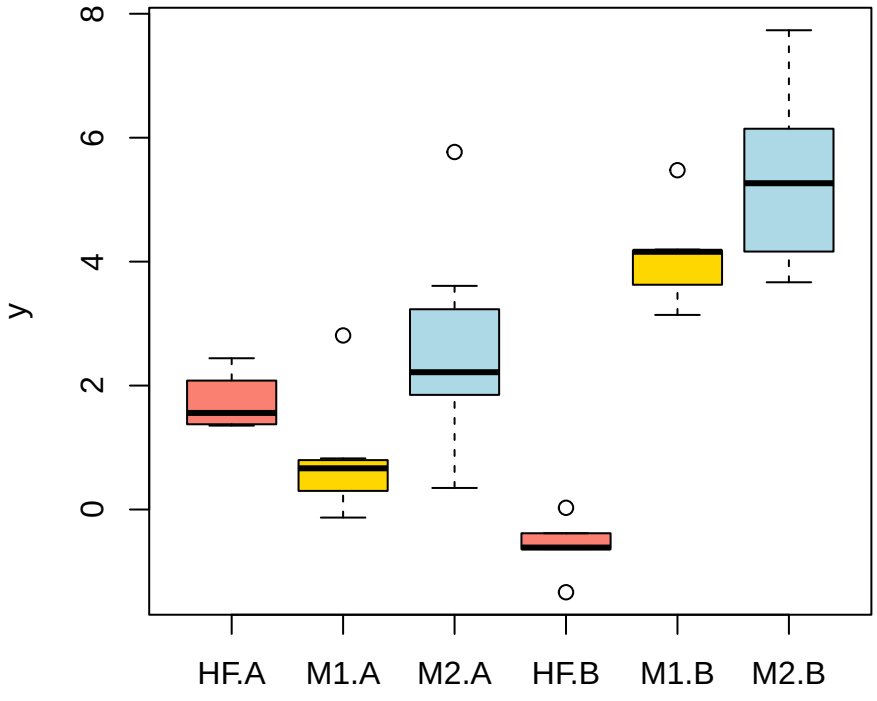


En este gráfico se ve lo que hemos mencionado antes: el fármaco A es mejor para personas que siguen una dieta alta en grasas, pero el fármaco B es mejor en los otros dos casos. Este gráfico muestra el cruce.

Un gráfico de caja también puede ayudar a mostrar la interacción. Se muestran dos diferentes, que difieren por el orden en que se especifican los factores (uno u otro puede ser más fácil de decodificar visualmente):

```
par(mfrow = c(2, 1))
boxplot(y ~ Drug * Diet, data = dcholest, col = c("salmon", "gold"))
boxplot(y ~ Diet * Drug, data = dcholest,
        col = c("salmon", "gold", "lightblue"))
```


y
0
2
4
6
8
A.HF
B.HF
A.M1
B.M1
A.M2
B.M2
Drug : Diet


y
0
2
4
6
8
HF.A
M1.A
M2.A
HF.B
M1.B
M2.B
Diet : Drug

Dados estos resultados (la fuerte interacción, que puede incluso revertir los efectos de un factor), no tiene mucho sentido informar de ningún efecto principal global y rara vez nos interesaría interpretar la significación (o no) del término Dieta o Fármaco. En general, **en presencia de interacciones, a menudo nos abstenemos de interpretar los efectos principales; esta es una consecuencia de lo que a menudo se conoce como el "principio de marginalidad"** [2]

XVIII.4.6. ANOVA sin interacciones

Se puede ajustar un modelo sin interacciones:

```
amodelnoint <- (lm(y ~ Diet + Drug, data=dcholest))
Anova(amodelnoint)

## Anova Table (Type II tests)
##
## Response: y
##           Sum Sq Df F value    Pr(>F)
## Diet      75.453  2  14.793 2.046e-05 ***
## Drug      32.261  1  12.650  0.001074 **
## Residuals 91.809 36
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Sin embargo, hay buenas razones para empezar ajustando primero un modelo **con** interacciones y, **solo** si no hay interacciones, ajustar un modelo aditivo más simple.

XVIII.4.7. El orden de los factores

1. Supongamos que realizamos un ANOVA de dos vías en el que la variable dependiente (Y) es «despierto en clase» y sus variables predictoras son el sexo (mujer u hombre) y el café por la mañana (sí o no).

2. Supongamos que en la muestra hay 10 mujeres que beben café y 10 hombres que no beben café. ¿Se puede decir algo sobre el efecto del sexo que no esté diciendo algo (o incluso todo) sobre el efecto del café? No, porque no se pueden estimar los factores de forma independiente.

3. Ahora supongamos que el diseño está perfectamente equilibrado: 5 mujeres que beben café, 5 mujeres que no beben café, 5 hombres que beben café, 5 hombres que no beben café. Si se dijera "he medido a una mujer", ¿se podría saber si también es bebedora de café o no? En este caso sí podemos estimar independientemente el efecto de los factores.

[2] Formulado correctamente, que también implica generalmente el uso de otros tipos de contrastes —como contr.sum, en lenguaje R— pruebas marginales en presencia de interacciones, lo que se llama Tipo III, puede tener sentido, pero no siempre son de interés.

4. Se puede sustituir la Y por "enfermedad cardiovascular" y las variables predictoras por "tabaco" (sí y no) y "ejercicio" (sí y no). ¿Se puede decir algo sobre los efectos del ejercicio si todas las personas de la muestra que hacen ejercicio son también no fumadores? (Se puede repetir esto con otros pares de variables, como dieta y ejercicio, expresión génica y edad, expresión génica y sexo, etc).

Para realizar el ANOVA sin interacción, creamos los datos excluyendo la dieta HF y calculamos ANOVA de interacción.

```
dcholest2 <- subset(dcholest, subset = Diet != "HF")
cholest2anova <- (lm(y ~ Diet * Drug, data = dcholest2))
Anova(cholest2anova)

## Anova Table (Type II tests)
##
## Response: y
##           Sum Sq Df F value    Pr(>F)
## Diet      17.879  1 11.7483  0.001967 **
## Drug      68.809  1 45.2150 3.216e-07 ***
## Diet:Drug  0.507  1  0.3335  0.568417
## Residuals 41.089 27
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

El resultado indica que no hay interacción (p-valor $>$ 0,05), por lo que podemos utilizar el modelo sin interacción.

```
lm1 <- lm(y ~ Diet + Drug, data = dcholest2)
anova(lm1)

## Analysis of Variance Table
##
## Response: y
##           Df Sum Sq Mean Sq F value    Pr(>F)
## Diet       1 15.897  15.897  10.701  0.002842 **
## Drug       1 68.809  68.809  46.318 2.156e-07 ***
## Residuals 28 41.597   1.486
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

lm2 <- lm(y ~ Drug + Diet, data = dcholest2)
anova(lm2)

## Analysis of Variance Table
##
## Response: y
```

```
##           Df Sum Sq Mean Sq F value    Pr(>F)
## Drug       1 66.827  66.827  44.983 2.793e-07 ***
## Diet       1 17.879  17.879  12.035  0.001708 **
## Residuals 28 41.597   1.486
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Los valores en ambas tablas son diferentes. Esta tabla de ANOVA se conoce como secuencial. En `lm2`, la primera fila testa la hipótesis nula de si hay alguna diferencia en la media de y relacionada con la variación del término Drug. La segunda fila mira si hay una diferencia en la media de y después de haber ajustado de lo que viene antes. Así, al mirar el efecto de Dieta, antes se ha ajustado el efecto de Drug, es decir, intenta explicar lo que no ha quedado explicado por el término anterior. En `lm1` ocurre lo mismo, pero en orden invertido. Por ello, los valores no son iguales (el diseño no es balanceado u ortogonal [3]). Se podría expresar también mediante:

```
m_a <- lm(y ~ Drug)
m_b <- lm(y ~ Drug + Diet)

m_a vs m_b
```

A continuación comparamos la salida de `Anova`.

```
Anova(lm1)

## Anova Table (Type II tests)
##
## Response: y
##           Sum Sq Df F value    Pr(>F)
## Diet      17.879  1  12.035  0.001708 **
## Drug      68.809  1  46.318 2.156e-07 ***
## Residuals 41.597 28
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Anova(lm2)

## Anova Table (Type II tests)
##
## Response: y
##           Sum Sq Df F value    Pr(>F)
## Drug      68.809  1  46.318 2.156e-07 ***
## Diet      17.879  1  12.035  0.001708 **
## Residuals 41.597 28
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

[3] Ortogonal hace referencia a un diseño experimental con un producto escalar que sea 0. Si los tamaños de muestra son iguales, se considera suficiente para poder decir que se trata de un diseño ortogonal, pudiendo así estimar cada efecto principal e interacción de forma independiente.

En este caso, las entradas son idénticas, y el orden no afecta. Las sumas de cuadrados de tipo II no son secuenciales. Se mira cada término como si hubiera sido introducido el último, habiendo ajustado por todo lo demás que no incluya a ese término.

El valor F (y el valor p) de la tabla ANOVA de Sumas de Cuadrados de Tipo II son los mismos que los del término que entra en último lugar en el Tipo I (los producidos mediante `anova`, sin la "A" mayúscula).

Se trata de un **fenómeno extremadamente común** cuando el diseño no está perfectamente equilibrado (con variables independientes categóricas) o existen correlaciones (con covariables continuas, como en la regresión). En resumen:

- Las sumas de cuadrados de tipo II (similares a los estadísticos t de un modelo lineal) muestran lo que aporta ese término, **dado que todos los demás** ya están en el modelo ("dado todos los demás": todos los demás que no incluyen este término, por lo que no hay interacciones con este término). En otras palabras, dado todos los demás términos (que no incluyen este término) ya se han tenido en cuenta. En realidad, éste es el resultado que obtendríamos al comparar dos modelos, uno con todos los términos y otro con todos los términos excepto el término en cuestión. (Siempre suponiendo que las interacciones con el término en cuestión son cero). Esto es lo que se obtiene con `Anova`.
- Las sumas de cuadrados de tipo I (o secuenciales) son secuenciales, en el orden mostrado en la salida. R, por defecto, da esto a través de `anova`.

XVIII.4.7.1. Tipo I vs Tipo II

Dejemos que R represente la suma residual de cuadrados para un modelo, así por ejemplo R(A, B, AB) es la suma residual de cuadrados que ajusta todo el modelo, R(A) es la suma residual de cuadrados que ajusta sólo el efecto principal de A, y R(1) es la suma residual de cuadrados que ajusta sólo la media.

Un residuo es la desviación entre lo observado y lo predicho por el modelo.

Type I

Las sumas de cuadrados de tipo I son dependientes del orden en que se introducen los factores en el modelo. R las genera por defecto con `anova`. En este enfoque, cada término se ajusta secuencialmente:

1. El primer término se ajusta comparando un modelo con solo la media ($R(1)$) contra uno que incluye ese término ($R(A)$).
2. El segundo término se ajusta después de incluir el primero, comparando $R(A)$ contra $R(A, B)$.
3. La interacción se ajusta al final, comparando $R(A, B)$ contra $R(A, B, AB)$.

```
A:  SS(A) = R(1) - R(A)
B:  SS(B|A) = R(A) - R(A, B)
AB: SS(AB|A, B) = R(A, B) - R(A, B, AB)
```

Type II

Las sumas de cuadrados tipo II son independientes del orden de los términos y miden el efecto de un factor asumiendo que no hay interacción significativa. Se considera cada término principal después de ajustar por los demás. Asume que no hay interacción, por lo que es más apropiado cuando las interacciones no son significativas.

```
A:  SS(A|B) = R(B) - R(A, B)
B:  SS(B|A) = R(A) - R(A, B)
AB: SS(AB|A, B) = R(A, B) - R(A, B, AB)
```

XVIII.4.7.2. Importancia del orden

Cuando el diseño está equilibrado, el orden no importa. Sin embargo, salvo que sepamos que los datos cumplen con unas propiedades concretas, deberíamos esperar que el orden sí importe.

```r
set.seed(1)
sex <- factor(rep(c("Male", "Female"), c(20, 20)))
drug <- factor(rep(rep(c("A", "B"), c(10, 10)), 2))
y <- rep(c(10, 13, 12, 16), rep(10, 4))
y <- y + rnorm(length(y), sd = 1.5)
y.data <- data.frame(y, sex, drug)

#Notice the perfect balance:
with(y.data, tapply(y, list(sex, drug), function(x) sum(!is.na(x))))
```

```
##         A  B
## Female 10 10
## Male   10 10
```

```r
with(y.data, tapply(y, list(sex, drug), mean))
```

```
##                A        B
## Female 11.79949 16.18110
## Male   10.19830 13.37327
```

Sólo a ojo, parece que la diferencia entre sexos es de unos 2, y la diferencia entre fármacos de unos 4. Y no, no hay interacción:

```r
summary(lm(y ~ sex * drug, data = y.data))
```

```
##
## Call:
## lm(formula = y ~ sex * drug, data = y.data)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6953 -0.6854  0.1639  0.9228  2.1946
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    11.7995     0.4322  27.304  < 2e-16 ***
## sexMale        -1.6012     0.6112  -2.620   0.0128 *
## drugB           4.3816     0.6112   7.169 1.97e-08 ***
## sexMale:drugB  -1.2066     0.8643  -1.396   0.1712
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.367 on 36 degrees of freedom
## Multiple R-squared:  0.7436, Adjusted R-squared:  0.7222
## F-statistic:  34.8 on 3 and 36 DF,  p-value: 9.746e-11
```

Ajustamos dos modelos, asumiendo que no hay interacción y cambiando el orden de los factores.

```
m1 <- lm(y ~ sex + drug, data = y.data)
m2 <- lm(y ~ drug + sex, data = y.data)

#La salida de coeficientes es la misma
summary(m1)

##
## Call:
## lm(formula = y ~ sex + drug, data = y.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9970 -0.7100  0.0357  0.8676  2.4963
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  12.1012     0.3790  31.927  < 2e-16 ***
## sexMale      -2.2045     0.4377  -5.037 1.26e-05 ***
## drugB         3.7783     0.4377   8.633 2.15e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.384 on 37 degrees of freedom
## Multiple R-squared:  0.7297, Adjusted R-squared:  0.7151
## F-statistic: 49.95 on 2 and 37 DF,  p-value: 3.079e-11

summary(m2)
```

```
##
## Call:
## lm(formula = y ~ drug + sex, data = y.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9970 -0.7100  0.0357  0.8676  2.4963
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  12.1012     0.3790  31.927  < 2e-16 ***
## drugB         3.7783     0.4377   8.633 2.15e-10 ***
## sexMale      -2.2045     0.4377  -5.037 1.26e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.384 on 37 degrees of freedom
## Multiple R-squared:  0.7297, Adjusted R-squared:  0.7151
## F-statistic: 49.95 on 2 and 37 DF,  p-value: 3.079e-11
```

También ajustamos dos modelos más pequeños, uno solo con sexo y otro solo con fármaco.

```
msex <- lm(y ~ sex, data = y.data)
mdrug <- lm(y ~ drug, data = y.data)

summary(msex)

##
## Call:
## lm(formula = y ~ sex, data = y.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.9743 -1.9275 -0.3339  1.9228  4.0477
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  13.9903     0.5302   26.39  < 2e-16 ***
## sexMale      -2.2045     0.7498   -2.94  0.00556 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.371 on 38 degrees of freedom
## Multiple R-squared:  0.1853, Adjusted R-squared:  0.1639
## F-statistic: 8.645 on 1 and 38 DF,  p-value: 0.005556

summary(mdrug)
```

```
##
## Call:
## lm(formula = y ~ drug, data = y.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.0992 -1.1956  0.0094  1.1359  3.2608
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  10.9989     0.3965  27.741  < 2e-16 ***
## drugB         3.7783     0.5607   6.738 5.57e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.773 on 38 degrees of freedom
## Multiple R-squared:  0.5444, Adjusted R-squared:  0.5324
## F-statistic: 45.41 on 1 and 38 DF,  p-value: 5.569e-08
```

En ambos casos, la estimación es la misma a partir del modelo con los dos factores o con un solo factor. Por ejemplo, las diferencias entre sexos son de aproximadamente 2,2 (el coeficiente que dice «sexMale») y las diferencias entre drogas de aproximadamente 3,8 (el coeficiente que dice «drugB»). Sin embargo, el error típico y, por tanto, el valor t y el valor p cambian.

La clave está en comprender que, aunque el coeficiente no cambie si se incluye o no el otro factor en el modelo (y no cambia porque aquí hay un equilibrio completo), el estadístico t y el valor p sí cambian porque el otro factor explica una gran parte de la varianza y, por tanto, hace que el error típico residual sea mucho menor si lo incluimos en el modelo.

```
anova(m1)

## Analysis of Variance Table
##
## Response: y
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## sex        1  48.599  48.599  25.371 1.258e-05 ***
## drug       1 142.754 142.754  74.527 2.149e-10 ***
## Residuals 37  70.873   1.915
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anova(m2)

## Analysis of Variance Table
##
```

```
## Response: y
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## drug       1 142.754 142.754  74.527 2.149e-10 ***
## sex        1  48.599  48.599  25.371 1.258e-05 ***
## Residuals 37  70.873   1.915
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

En cuanto a las tablas ANOVA, el orden no cambia nada. Esto se debe a que las contribuciones de cada factor no dependen en absoluto del otro (es decir, los cuadrados medios de cada factor no dependen del otro). Y como la F es el cociente de los cuadrados medios del factor sobre los cuadrados medios de los residuos (y esto es lo que queda después de haber ajustado todo), el orden no afecta al estadístico F ni al valor p.

```
anova(msex)

## Analysis of Variance Table
##
## Response: y
##           Df  Sum Sq Mean Sq F value   Pr(>F)
## sex        1  48.599  48.599  8.6447 0.005556 **
## Residuals 38 213.627   5.622
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anova(mdrug)

## Analysis of Variance Table
##
## Response: y
##           Df Sum Sq Mean Sq F value    Pr(>F)
## drug       1 142.75 142.754  45.406 5.569e-08 ***
## Residuals 38 119.47   3.144
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Observa que el cuadrado medio de cada factor es el mismo que en las tablas anteriores. Así que los cuadrados medios de Sexo no dependen de si el fármaco está o no en el modelo. Pero el estadístico F (y el valor p) sí cambian mucho, porque lo que cambia mucho son los cuadrados medios de los residuos. Esto se debe a que el otro factor, el que no hemos incluido, sí explica mucha variabilidad, pero en estas dos últimas tablas, como el otro factor no está en el modelo, esa variabilidad está incluida ahora en el término de error.

Así que, resumiendo: cuando hay equilibrio, el orden no cambia nada si incluimos ambos factores en el modelo. Sin embargo, tener o no el otro factor en el modelo

puede suponer una diferencia para los errores estándar, los errores estándar residuales y, por tanto, los p-valores.

XVIII.4.8. Una observación por celda

No se deben ajustar modelos con una única observación por celda.

XVIII.4.9. Breve ejemplo de dos vías

```
library(ISwR)

##
## Adjuntando el paquete: 'ISwR'
## The following object is masked from 'package:survival':
##
##     lung

ck1 <- lm(time ~ width * temp, data = coking)
Anova(ck1)

## Anova Table (Type II tests)
##
## Response: time
##             Sum Sq Df F value    Pr(>F)
## width      123.143  2 222.102 3.312e-10 ***
## temp        17.209  1  62.076 4.394e-06 ***
## width:temp   5.701  2  10.283  0.002504 **
## Residuals    3.327 12
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anova(ck1)

## Analysis of Variance Table
##
## Response: time
##            Df  Sum Sq Mean Sq F value    Pr(>F)
## width       2 123.143  61.572 222.102 3.312e-10 ***
## temp        1  17.209  17.209  62.076 4.394e-06 ***
## width:temp  2   5.701   2.851  10.283  0.002504 **
## Residuals  12   3.327   0.277
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Tan pronto como vemos una interacción entre los dos factores, no vamos a tener en cuenta el efecto de cada factor por individual.

XVIII.4.10. Análisis y consideraciones en modelos de ANOVA con tres factores y comparaciones múltiples

Cuando se trabaja con un modelo de ANOVA de tres factores (U, V, W), las interacciones pueden complicar la interpretación de los efectos principales. La tabla ANOVA típica incluiría las siguientes filas:

- Efectos principales: U, V, W
- Interacciones de dos vías: $U : V, U : W, V : W$
- Interacción de tres vías: $U : V : W$

Supongamos las siguientes observaciones:

- No hay evidencia de una interacción a tres vías.
- No hay evidencia de una interacción U:V.
- No hay evidencia de una interacción U:W.
- Solo la interacción V:W es significativa.

Así, solo podemos examinar el efecto de U. Los efectos de V y W no deben interpretarse de forma independiente debido a la interacción significativa V:W.

XVIII.4.11. Comparaciones múltiples de medias en ANOVA de dos vías

Cuando trabajamos con modelos de ANOVA de dos vías (A y B):

- Sin interacciones: las comparaciones múltiples son relativamente sencillas, ya que los efectos principales de A y B pueden interpretarse independientemente.
- Con interacciones: La interpretación de los efectos principales cambia, ya que el efecto de un factor depende del nivel del otro. Las comparaciones deben hacerse considerando los niveles específicos de los factores involucrados.

Supongamos que tenemos el modelo `Y ~ A * B` En el que A y B tienen dos niveles (A: a1, a2; B: b1, b2).

Level of A	Level of B	Mean
a1	b1	3
a1	b2	5
a2	b1	8
a2	b2	2

Podemos dibujar dos gráficos de interacción: uno con a1 y a2 en el eje X y otro con b1 y b2 en el eje X. En el primer gráfico, las líneas de b1 y b2 se cruzarían, mientras que en el segundo gráfico no.

Así, la diferencia entre a1 y a2 depende del nivel de B:
Diferencia para b1: $8 - 3 = 5$
Diferencia para b2: $2 - 5 = -3$

Para calcular intervalos de confianza (IC), es necesario especificar el nivel de B al que se desea evaluar la diferencia entre a1 y a2, y viceversa.

Es posible calcular IC a diferentes niveles de los factores involucrados. Por ejemplo, nivel promedio (media marginal) o nivel específico (por ejemplo, B = b1). Calcular diferencias promediadas (media marginal) es una opción, pero su utilidad depende de la pregunta de investigación. Si las interacciones son fuertes (como líneas que se cruzan), los promedios marginales pueden ser poco informativos.

Es fundamental describir claramente cómo se calcularon las comparaciones y en qué niveles de los factores se basaron. Realizar comparaciones extensivas y reportar selectivamente es una mala práctica científica.

En conclusión, en ANOVA con interacciones significativas, la interpretación de los efectos principales se complica. Es necesario considerar los niveles específicos de los factores al analizar las diferencias. El contexto de la pregunta de investigación debe guiar si los efectos promediados son útiles o si se deben reportar efectos a niveles específicos.

XVIII.5. Regresión lineal

La regresión lineal es una técnica de modelado estadístico que busca describir la relación entre una variable dependiente Y y una o más variables independientes X. En su forma más simple, el modelo lineal se expresa como:

$$Y = \alpha + \beta X + \varepsilon$$

donde Y es la variable dependiente, X la independiente, β la pendiente (cambio en Y por unidad de cambio en X) y α el intercepto (valor esperado de Y cuando X=0). El ajuste del modelo implica estimar los parámetros α y β para minimizar la suma de los errores cuadrados.

XVIII.5.1. Transformación logarítmica

En casos donde la relación entre las variables no es lineal (como en el ejemplo de la tasa metabólica y la masa corporal), aplicar una transformación logarítmica puede simplificar la relación:

$$log(Y) = \alpha + \beta log(X) + \varepsilon$$

Esto permite modelar relaciones de tipo potencia ($Y = kX^b$) como una línea recta en escala log-log.

Queremos tomar el logaritmo de todas las variables continuas relevantes [4]

```
anage_a_r$logMetabolicRate <- log(anage_a_r$Metabolic.rate..W.)
anage_a_r$logBodyMass <- log(anage_a_r$Body.mass..g.)
anage_a_r$logLongevity <- log(anage_a_r$Maximum.longevity..yrs.)
```

Por ahora, solo nos vamos a centrar en las Aves.

Queremos modelar la tasa metabólica como una función de la masa corporal (ten en cuenta que este conjunto de datos es bastante agradable, porque los nombres de las columnas están bien etiquetados e incluyen información sobre las unidades). **Cuidado:** lo que vamos a hacer no es correcto, ya que los datos no son independientes (las especies comparten antepasados comunes, y están relacionados en diversos grados, como cualquier árbol filogenético mostraría, y como se puede inferir al mirar los nombres de algunas especies). Así que estamos violando el supuesto de independencia. Lo que estamos haciendo aquí es sólo por el bien del ejemplo, y porque este es un buen conjunto de datos[5]

```
metab <- lm(Metabolic.rate..W. ~ Body.mass..g., data = anage_a)
summary(metab)
```

```
##
## Call:
## lm(formula = Metabolic.rate..W. ~ Body.mass..g., data = anage_a)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2390 -0.3386 -0.2095  0.1578  3.8380
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.5300123  0.0694005   7.637 1.74e-12 ***
## Body.mass..g. 0.0025673  0.0001133  22.663  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7975 on 164 degrees of freedom
##   (1020 observations deleted due to missingness)
## Multiple R-squared:  0.758, Adjusted R-squared:  0.7565
## F-statistic: 513.6 on 1 and 164 DF,  p-value: < 2.2e-16
```

La fila de la salida que dice "(Intercepto)" da la estimación del intercepto. El estadístico t (bajo "valor t") está probando que el intercepto es cero. Y no lo es. Pero

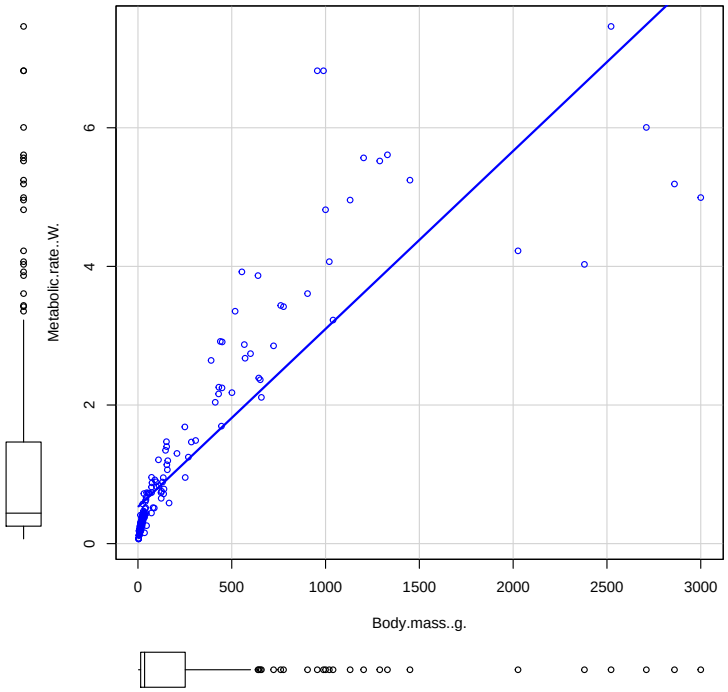
[4] Crear estas nuevas variables no es realmente necesario en general para ajustar modelos. Pero algunas funciones del paquete `HH` dan problemas si no lo hacemos. problemas si no lo hacemos.

[5] Esto se puede hacer correctamente, la incorporación de información filogenética en la regresión modelo de regresión, pero esto está fuera del alcance de esta clase. Es realmente fascinante. A menudo se habla de utilizar el método comparativo en biología evolutiva.

las pruebas sobre el intercepto rara vez son interesantes (excepto para los casos con un 0 natural y significativo). La segunda línea es más interesante: es la pendiente, cuánto aumenta la tasa metabólica por unidad de aumento de la masa corporal (por supuesto, para interpretar esto necesitamos conocer las unidades). Y el estadístico t comprueba si la pendiente es 0. Desde luego, hay pruebas sólidas de que la tasa metabólica aumenta con la masa corporal.

Tendríamos que haber mostrado los datos al comienzo. Realizamos un scatterplot para mostrar la dispersión de los datos.

```
scatterplot(Metabolic.rate..W. ~ Body.mass..g.,
            smooth = FALSE,
            data = anage_a)
```

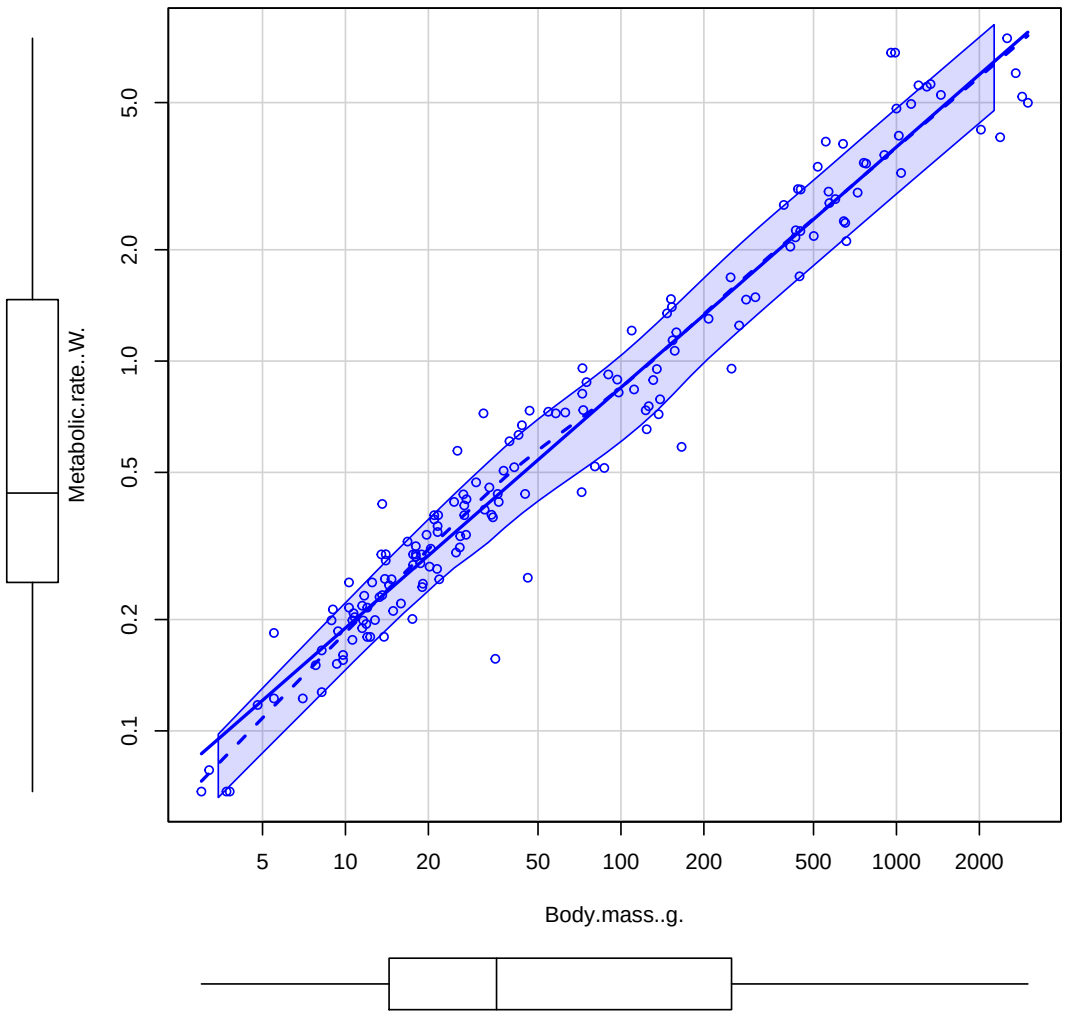


Los datos no se ajustan a una recta, por lo que es necesario reajustar el modelo, transformando las variables dependientes e independientes mediante logaritmo.

```
metablog <- lm(logMetabolicRate ~ logBodyMass, data = anage_a)
summary(metablog)

##
## Call:
## lm(formula = logMetabolicRate ~ logBodyMass, data = anage_a)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.00686 -0.14349  0.01545  0.16584  0.61638
##
```

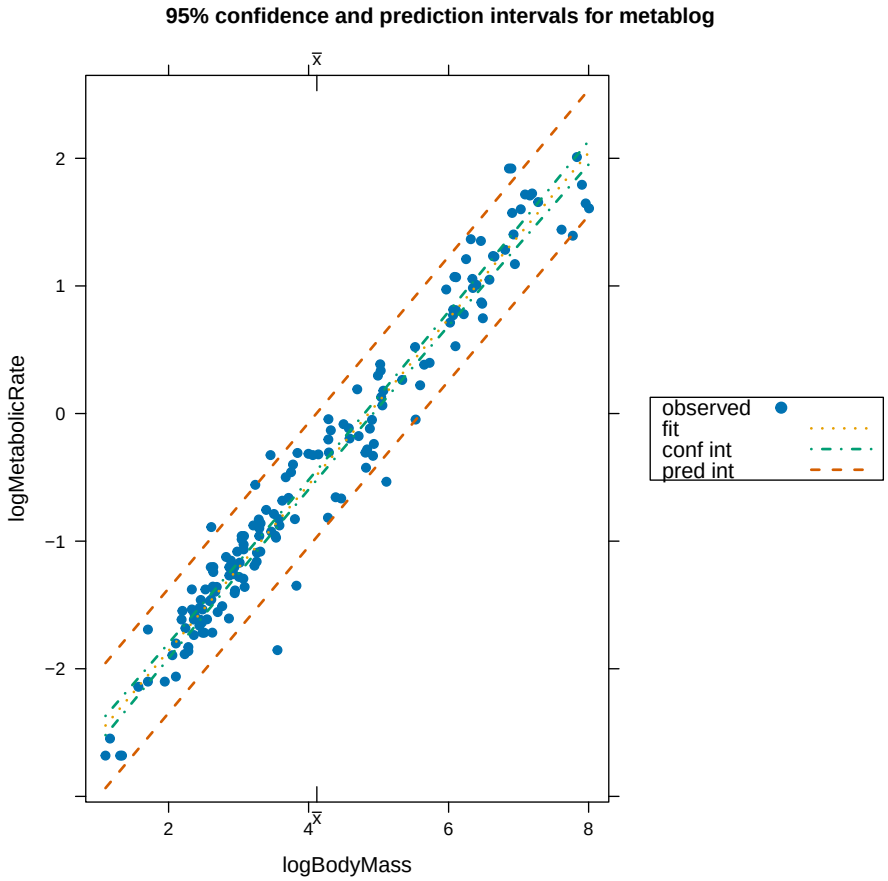
```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.15949    0.04895  -64.55   <2e-16 ***
## logBodyMass  0.65037    0.01095   59.38   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2452 on 164 degrees of freedom
##   (1020 observations deleted due to missingness)
## Multiple R-squared:  0.9556, Adjusted R-squared:  0.9553
## F-statistic:  3527 on 1 and 164 DF,  p-value: < 2.2e-16
```

```
scatterplot(Metabolic.rate..W. ~ Body.mass..g., log = "xy",
            smooth = TRUE, boxplots = 'xy',
            data = anage_a)
```



XVIII.5.2. Intervalos de confianza e intervalos de predicción

```
library(HH)

## Cargando paquete requerido: lattice
## Cargando paquete requerido: grid
## Cargando paquete requerido: latticeExtra
##
## Adjuntando el paquete: 'latticeExtra'
## The following object is masked from 'package:ggplot2':
##
##     layer
## Cargando paquete requerido: gridExtra
##
## Adjuntando el paquete: 'HH'
## The following objects are masked from 'package:car':
##
##     logit, vif
## The following object is masked from 'package:base':
##
##     is.R

ci.plot(metablog)
```



Las líneas rojas delimitan por dónde se debe esperar que caigan la mayor parte de los puntos. Las líneas rojas dependen de la variabilidad de las y, mientras que el IC permite ajustar los parámetros del modelo. Si el tamaño de muestra es muy grande, se puede estimar la línea con mucha precisión. AUnque se sepa realmente el valor esperado de y dado x, en torno a la media los valores pueden tener mucha variabilidad.

- Las bandas de intervalo de confianza son para la propia línea de regresión, que es lo mismo que decir que son para el valor esperado de la variable de respuesta. (Estamos modelando $E[y] = \alpha + \beta x$). Cuanta más incertidumbre tengamos sobre la tendencia general, sobre la línea, más amplias serán las bandas del intervalo de confianza.

- Las bandas del intervalo de predicción son para las observaciones; así, además de la incertidumbre en torno a la recta de regresión, tenemos el σ, la varianza de las observaciones en torno a su valor medio, en torno a $E[y]$.

- Para aclarar lo anterior, piensa en lo siguiente: supongamos que tomamos una muestra enorme, digamos de 10 millones de personas y hace una regresión de la masa corporal sobre la altura corporal. Estimarás la recta de regresión con muy poca incertidumbre. En otras palabras, estarás muy, muy seguro de dónde está $E[masa_corporal]$ dado $altura_corporal$. Pero, independientemente del tamaño de la muestra, hay bastante variación en la masa corporal para una altura corporal dada. Así que las bandas de predicción serán relativamente anchas, para acomodar este hecho.

Esta es, por cierto, la razón por la que se puede estar muy seguro de una tendencia general (el valor esperado) y, sin embargo, ser incapaz de predecir realmente el valor real de un individuo específico. Es fundamental comprender la diferencia entre predecir la media y predecir el valor de un individuo concreto.

Por el contrario, si σ es muy, muy, muy pequeño, entonces las bandas de predicción serán muy, muy cerca de las bandas de intervalo de confianza.

En resumen, los intervalos de confianza delimitan la incertidumbre en la estimación de la línea de regresión (valores esperados de Y). Cuando mayor sea la muestra, más estrecho será el intervalo. Los intervalos de predicción capturan la variabilidad de las observaciones individuales en torno a la línea de regresión. Incluyen la incertidumbre de la estimación más la varianza de las observaciones.

XVIII.5.3. Intervalos de confianza para los parámetros

```
confint(metablog)
```

```
##                  2.5 %    97.5 %
## (Intercept) -3.256139 -3.062837
## logBodyMass  0.628743  0.671992
```

Se trata de intervalos de confianza para los propios parámetros. Sin embargo, las estimaciones de los parámetros están correlacionadas. Esto significa que la comprobación de cada parámetro por separado puede dar lugar a respuestas diferentes de las que se obtienen comprobando ambos a la vez. Mostramos una elipse de confianza conjunta.

```
## Correlation of estimated coefficients
round(cov2cor(vcov(metablog)), 3)
```

```
##             (Intercept) logBodyMass
## (Intercept)       1.000      -0.921
## logBodyMass      -0.921       1.000
```

```
## Plot of joint and each-at-time CIs
library(ellipse)
```

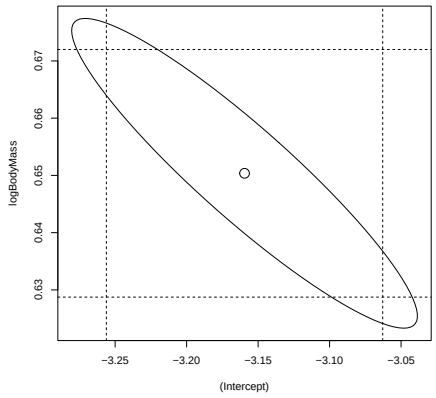
```
## Warning: package 'ellipse' was built under R version 4.4.2
```

```
##
## Adjuntando el paquete: 'ellipse'
```

```
## The following object is masked from 'package:car':
##
##     ellipse
```

```
## The following object is masked from 'package:graphics':
##
##     pairs
```

```r
plot(ellipse(metablog), type = "l")
points(coef(metablog)[1], coef(metablog)[2], pch = 1, cex = 2)
abline(v = confint(metablog)[1, ], lty = 2)
abline(h = confint(metablog)[2, ], lty = 2)
```



XVIII.6. Regresión múltiple

XVIII.6.1. Introducción a la regresión múltiple

En la regresión múltiple, el modelo se extiende a múltiples predictores:

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \varepsilon$$

Cada coeficiente describe el efecto de su predictor sobre Y, manteniendo constantes los demás predictores. Correlaciones altas entre predictores pueden inflar errores estándar, dificultando la identificación de efectos individuales. R^2 mide la proporción de varianza explicada por el modelo, mientras que el valor ajustado penaliza la inclusión de predictores irrelevantes.

El significado de las variables es:

```
'age' a numeric vector, age in years.
'sex' a numeric vector code, 0: male, 1:female.
'height' a numeric vector, height (cm).
'weight' a numeric vector, weight (kg).
'pemax' a numeric vector, maximum expiratory pressure.
```

Para la regresión múltiple ajustamos

$$pemax = \alpha + \beta_1 age + \beta_2 height + \beta_3 weight + \varepsilon$$

```
mcyst <- lm(pemax ~ age + height + weight, data=cystfibr2)
summary(mcyst)
```

```
##
## Call:
## lm(formula = pemax ~ age + height + weight, data = cystfibr2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -43.675 -21.566   3.229  16.274  48.068
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 64.65555   82.40935   0.785    0.441
## age          1.56755    3.14363   0.499    0.623
## height      -0.07608    0.80278  -0.095    0.925
## weight       0.86949    0.85922   1.012    0.323
##
## Residual standard error: 27.41 on 21 degrees of freedom
## Multiple R-squared:  0.4118, Adjusted R-squared:  0.3278
## F-statistic: 4.901 on 3 and 21 DF,  p-value: 0.009776
```

La tabla tiene 4 líneas: un intercepto y tres pendientes. Ya no hay una línea, si no tres planos. La línea del intercepto la vamos a ignorar por ahora. La línea con "age" muestra el coeficiente una vez visto todo lo demás (incluyendo interacciones si las hubiera, que aquí no las hay; como en la suma de cuadrados de tipo II). En la línea de edad se muestra cómo cambia pemax en base a edad una vez ajustado por altura y peso. El p-valor es grande, dando la impresión de que pemax no cambia con edad una vez que se ha ajustado por estatura y peso. La misma interpretación se da para estatura y peso; ninguna de las tres filas tiene un p-valor que se acerque a lo significativo. El estadístico de la F dicef que con 3 (edad, altura y peso) y 21 grados de libertad, el estadístico de la F es de 4,9, y el p-valor es de 0,009. Esta F mide un modelo de una media frente a uno en el que pemax cambia por edad, altura y peso. Esa comparación dice que nuestro modelo es mejor que solo una media; en otras palabras, hay evidencias significativas frente a la hipótesis nula de que el modelo con solo el intercepto es suficiente. Hay evidencia significativa frente a la hipótesis nula de que no hay interacción entre pemax, estatura, edad y peso.

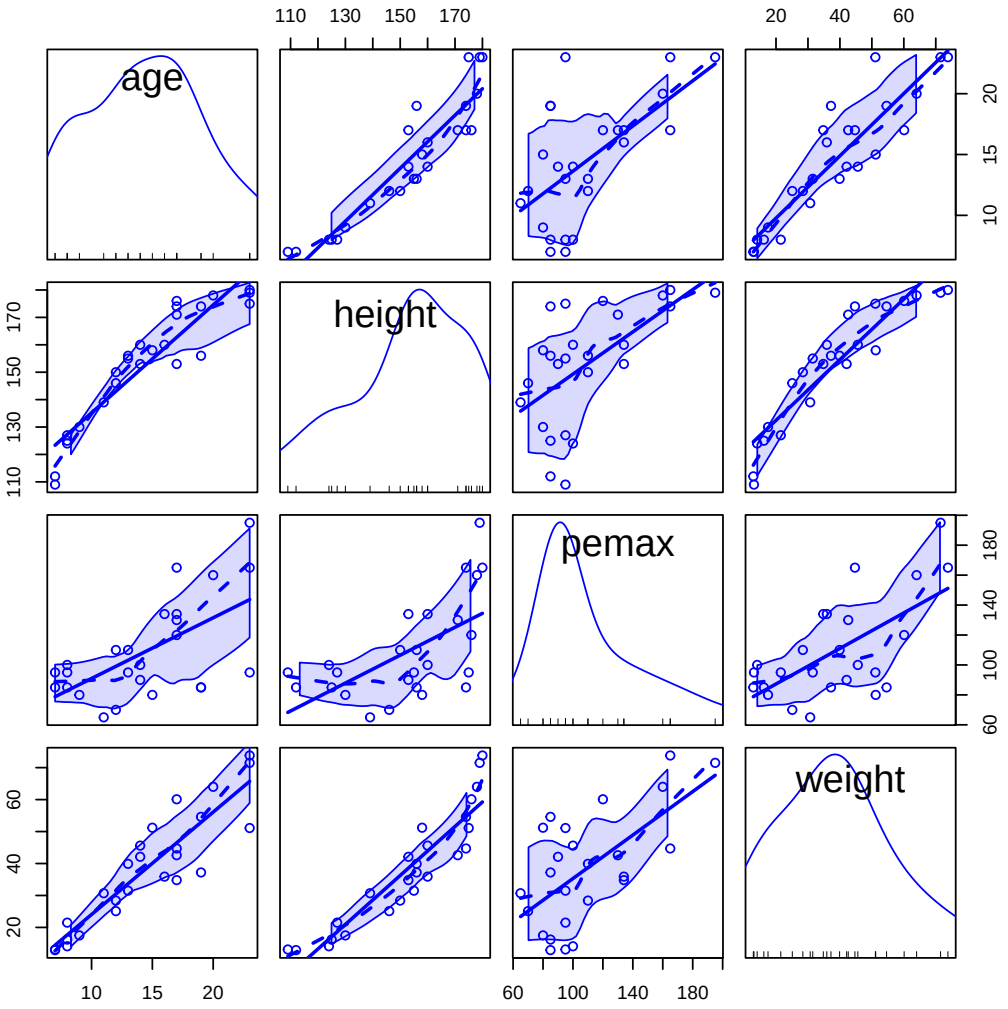
Para los R cuadrados, aparecen uno múltiple y otro ajustado. Ambos hacen referencia a la misma idea: cuánta variabilidad en la variable dependiente (pemax) se puede explicar o capturar con el modelo. Esto hay dos formas de medirlo. El R cuadrado múltiple coge las observaciones de pemax, coge las predicciones de pemax de acuerdo al modelo, calcula la correlación entre lo observado y predicho y lo eleva al cuadrado. El problema del número es que solo puede subir a medida que se introducen términos. El R cuadrado ajustado utiliza el error residual, comparando cómo cambia teniendo en cuenta el número de términos que se van introduciendo. Esta línea indica que en torno al 40 % de la variabilidad de la variable dependiente se puede explicar por el modelo.

Como se ha ajustado el modelo con lm, se pueden mirar las tablas de ANOVA:

```
anova(mcyst)

## Analysis of Variance Table
##
## Response: pemax
##           Df  Sum Sq Mean Sq F value   Pr(>F)
## age        1 10098.5 10098.5 13.4371 0.001441 **
## height     1   182.3   182.3  0.2426 0.627427
## weight     1   769.6   769.6  1.0240 0.323082
## Residuals 21 15782.2   751.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Anova(mcyst)

## Anova Table (Type II tests)
##
## Response: pemax
##            Sum Sq Df F value Pr(>F)
## age         186.9  1  0.2486 0.6232
## height        6.8  1  0.0090 0.9254
## weight      769.6  1  1.0240 0.3231
## Residuals 15782.2 21
```

Al mirar edad tras calcular anova sin ajustar por estatura o peso (de forma secuencial), resulta ser muy significativa. Como es significativo, una vez que se introduce edad, estatura no dice nada, ni peso. Al mirar el Anova, en este modelo se muestra lo mismo que la tabla anterior en la que nada es significativo.

```
scatterplotMatrix( ~ age+height+pemax+weight,
                   data = cystfibr2)
```



Si cambiamos el orden de los factores, el primer factor es significativo, mientras que los demás dejan de serlo.

```
anova(lm(pemax ~ height + weight + age, data = cystfibr2))

## Analysis of Variance Table
##
## Response: pemax
##           Df  Sum Sq Mean Sq F value   Pr(>F)
## height     1  9634.6  9634.6 12.8200 0.001763 **
## weight     1  1228.9  1228.9  1.6352 0.214935
## age        1   186.9   186.9  0.2486 0.623214
## Residuals 21 15782.2   751.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anova(lm(pemax ~ weight + height + age, data = cystfibr2))
```

```
## Analysis of Variance Table
##
## Response: pemax
##           Df  Sum Sq Mean Sq F value   Pr(>F)
## weight     1 10827.2 10827.2 14.4067 0.001058 **
## height     1    36.4    36.4  0.0484 0.827949
## age        1   186.9   186.9  0.2486 0.623214
## Residuals 21 15782.2   751.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

En este caso, estamos ajustando pemax a un modelo en función de tres variables que están muy correlacionadas entre ellas.

Así que, para resumir, cuando las variables predictoras están correlacionadas:

- Incluso si cada uno de los predictores (o subconjuntos de ellos) parece estar fuertemente asociado al resultado, sus valores p podrían ser elevados y el signo del coeficiente podría invertirse al ajustar todos los predictores correlacionados.
- El modelo global (dado por el estadístico F global o el R^2) podría indicar que el modelo está haciendo un trabajo decente (ciertamente mucho mejor que ajustarse a una sola media) y, sin embargo, los valores p individuales podrían sugerir que ningún predictor individual es relevante.

XVIII.6.2. R^2 y R^2 ajustado

- R^2 (R-cuadrado) es la proporción de variabilidad de la variable dependiente explicada por el modelo. También es el cuadrado de la correlación entre los valores observados y predichos (predichos según el modelo) de la variable dependiente.
- Pero añadir variables predictoras que en realidad no explican nada nunca disminuirá R^2. El R^2 ajustado tiene esto en cuenta (añadir un predictor sólo aumentará el R^2 ajustado si el predictor contribuye de algún modo a mejorar el valor predictivo). Tenga en cuenta que el R^2 (sin ajustar) es el cambio relativo en las sumas de cuadrados residuales, mientras que el R^2 ajustado es el cambio relativo en la varianza residual. En general, el R^2 ajustado es un mejor número para mirar (y tenga en cuenta que puede, en los modelos que no explican nada, llegar a ser negativo)[6]

XVIII.6.3. Interacciones entre variables continuas

[6] Hay cuestiones adicionales que pueden necesitar ser considerado. Por ejemplo, la R^2 por defecto que R proporciona, tanto ajustada como sin ajustar, no debe utilizarse en modelos sin intercepción (es decir, regresión a través del origen). Y el R^2 no ajustado tiene la virtud de que se puede calcular para muchos otros modelos, ya que es sólo el cuadrado de la correlación entre lo predicho y lo observado.