Principios de Estadística

Leonardo Collado Torres y María Gutiérrez Arcelus
Licenciatura en Ciencias Genómicas, UNAM
www.lcg.unam.mx/~lcollado/index.php
www.lcg.unam.mx/~mgutierr/index.php

Cuernavaca, México
Febrero - Junio, 2009

Introducción y R básico

De donde viene R

- Para muchos R es un dialecto porque es un derivado del lenguaje S creado por John Chambers y co en los Bell Labs. En sí, R lo escribieron a mitad de los 90s Ross Ihaka y Robert Gentleman.
- Desde 1997, R ha sido manejado por el *R Development Core Team* y se ha mantenido como open-source.
- Una ventaja de R es que se puede usar en varias plataformas: UNIX, Windows, Mac.
- R en sí es un lenguaje de computación creado para facilitar la manipulación de datos, hacer cálculos y gráficas de alto nivel. Es por esto que R es fuerte en estadística.

Propiedades de R

`R` es un ambiente para trabajar en estadística computacional y al mismo tiempo es un lenguaje de programación. Hay usuarios que solo van a usar las funciones básicas de `R` (como una calculadora) mientras otros incluso harán paquetes que liguen `R` con `C`. En fin, `R`:

- es efectivo en el manejo de datos y su almacenamiento.
- tiene muchos operadores para hacer cálculos en arreglos (vectores) y matrices.
- tiene una gama de herramientas para el análisis de datos. Hay muchos paquetes disponibles, como la familia de Bioconductor.
- tiene un sistema de gráficas muy útil para el análisis de datos. Excel es cosa del pasado ;)

Propiedades de R

- ya viene con modelos estadísticos.
- hay muchos manuales y un sistema de ayuda bastante bueno. Además hay una comunidad internacional que te extiende la mano :).

Abrir y cerrar R

- Para abrir `R` simplemente tienen que escribir el comando R en UNIX. Lo primero que verán es una pequeña descripción de `R` incluyendo la versión que tienen instalada.
- Al abrir `R`, este busca en el directorio donde están información de alguna sesión previa. Esto luego sera útil con los *workspace*.
- Para cerrar `R` simplemente escriban q(). Les va a pedir si quieren guardar una imagen del workspace – por ahora digan que no.

Workspace

- Muchas veces tienes que interrumpir tu trabajo. R tiene toda una funcionalidad llamada workspace que te ayuda a retomar tu trabajo de sesiones previas.
- Cuando guardas el workspace se crean dos archivos: .RData y .Rhistory en el directorio donde estes trabajando. Estos almacenan todos los objetos que haya definido el usuario (vectores, matrices, listas, funciones). La próxima vez que abras R en ese directorio, carga todo lo que creaste antes automáticamente.
- Hay una serie de funciones que les pueden ayudar para organizar su trabajo en R. getwd te da tu directorio de trabajo actual, setwd lo cambia y history te muestra los últimos 25 comandos que usaste.

R como Calculadora

- R es un *expression language*. Aka[1], una R no es igual a una r. Los nombres de variables tienen que empezar por un punto[2] o caracteres alfanuméricos.
- Hagan los siguientes comandos:

```
> 2 + 2
> 2^2
> r <- c(1:3, 4.5, 109)
> pi * r^2
> sqrt(36)
> sin(2 * pi)
> exp(1)
> log(10)
> log(10, base = 10)
```

[1] also known as

[2] Una letra le tiene que seguir al punto para que sea un nombre válido

Asignación de valores

- En R, hay 3 formas de asignar valores, aunque en general se usan solo dos: = y <-
- Preferencialmente usen <- simplemente para evitar confusiones. Es que el signo = se usa para el paso de valores en las funciones.

```
> A <- c(a = 1, b = 2)["b"]
> A = c(a = 1, b = 2)["b"]
> A

b
2
```

- Aquí queda más clara la asignación en la primera línea, aunque las dos hacen lo mismo.

Vectores

- R es un lenguaje vectorizado, así que puedes ver todas tus variables como vectores. Hay varios *modos*: `numeric`, `character`, `logical`.
- Tal vez la función más usada en R es `c()`. Con esta función puedes generar vectores de datos.

```
> v1 <- c(1:10)
> v2 <- runif(10)
> v3 <- sample(c("A", "C", "G", "T"),
+      size = 10, replace = TRUE)
> v4 <- v3 %in% c("A", "G")
> v5 <- c("foo", 2, TRUE)
> v6 <- c(2, "3")
```

Vectores

- Puedes usar la función mode para encontrar que tipo de vector tienes. Además, con as puedes cambiar el modo de un vector. Intenten cambiar al modo *numeric* los vectores `v5 y v6`:

```
> as.numeric(v5)
> as.numeric(v6)
```

- La familia de la función `as` es muy extensa, aunque los principales son: `as.character`, `as.data.frame`, `as.matrix` y `as.factor`.

Un ejemplo sencillo

Example (Tamaño Fagos)

Conocemos el tamaño del genoma de 10 bacteríofagos y queremos explorar esta información. Sus tamaños en mbs son: `233.2 180.5 280.3 244.8 252.4 178.2 211.2 196.2 176.8 185.7` Almacenen esta informaciónel en el vector `fagos` y encuentren:

1. La suma de los tamaño de los genomas
2. La longitud del vector fagos
3. El tamaño promedio de los genomas

Así se resuelve:

Un ejemplo sencillo


```
> fagos <- c(233.2, 180.5, 280.3,
+      244.8, 252.4, 178.2, 211.2,
+      196.2, 176.8, 185.7)
> sum(fagos)
[1] 2139.3
> length(fagos)
[1] 10
> sum(fagos)/length(fagos)
[1] 213.93
> mean(fagos)
[1] 213.93
```

Además, pueden usar `sort()`, `min()`, `max()`, `range()`, `diff()`, `cumsum()` y summary().

Reciclaje de vectores

- En `R` la mayoría de las funciones están vectorizadas[3]. Por ejemplo cuando hacemos `x = 2; y = 3; x + y` en realidad estamos haciendo $x[i] + y[i], i \in 1, \ldots \max\{|x|, |y|\}$
- Si la longitud de los dos vectores no es la misma, `R` recicla el más chico con tal de llegar a la longitud del grande.
- Prueben con `c(2,3) + c(3,4,5)` y compárenlo con `c(2,3) + c(3,4,5,8)`
- Siempre tengan cuidado con los warnings que salen. En el caso del reciclaje, estos solo salen si `(length(x) %% length(y)) != 0`
- Con esto en mente ahora podemos encontrar la suma de los cuadrados de un vector.

Reciclaje de vectores

Example (Suma de cuadrados)

Normalmente sacamos el cuadrado de cada valor y luego los sumamos.

```
> x <- c(2, 7, 19)
> x[1]^2 + x[2]^2 + x[3]^2
```

```
[1] 414
```

Pero ahora con el reciclaje, simplemente aplicamos la función para sacar el cuadrado al vector entero. Por reciclaje, la función se va a aplicar a cada elemento del vector. Además, usamos la función `sum` para sumar los valores resultantes.

```
> sum(x^2)
```

```
[1] 414
```

[3] En las que no, es porque no tendría sentido vectorizarlas

Buscando ayuda

- Tal vez lo más importante en cualquier lenguaje de programación es saber donde buscar ayuda. `R` tiene un sistema de ayuda bastante completo, aunque a veces si hay que meterse a google.
- La función madre para buscar ayuda es `help()` Digamos que no saben que hace la función `names`, así que pueden buscar ayuda al respecto con `help("names")` o alguno de los atajos: `?names` o `?"names"`.
- Si no saben que es lo que buscan pueden usar `help.start()` que abre una página html. Aquí siempre encontrarán ejemplos que los pueden ayudar a entender. Estos los pueden copiar y pegar en R para correlos :)

Buscando ayuda

- Para hacer una búsqueda más profunda usen `help.search()` ya que esta función busca en más secciones de los manuales de ayuda. Por ejemplo, `help.search("names")`
- Si están buscando nombres de funciones, usen `apropos()` Por ejemplo, `apropos("names")`. Otras funciones útiles son `RSiteSearch()`, `args()` y `example()`.

Un ejercicio simple

Aprendiendo a hacer secuencias y repeticiones

Como un ejercicio simple queremos que aprendan a usar las funciones seq, rev, rep, paste y el operador `colon :`.
Almacenen en diferentes vectores los siguientes datos sin usar `c()` a menos de que no haya otra opción.

- Las fracciones 1/1 hasta 1/10 *usando enteros*.
- Los años desde 1964 hasta el 2008.
- Los múltiplos de 25 desde 1000 hasta 0 *en ese orden*.
- La serie "A" "A" "T" "T" "T" "T" "C" "G" y luego conviertanla a "AATTTTCG".
- Los números de Fibonnacci del 1 al 34

Usar los índices

- En `R` al igual que en otros lenguajes es importante aprender como accesar a la información que tienes en tu variable; vectores en este caso.
- Muchas veces van a tener su información almacenada en un vector de datos; hay cuatro índices principales que puedes usar para seleccionar subconjuntos de tu vector: vectores lógicos, un vector de enteros positivos, otro de negativos y un string de caracteres.

Vectores Lógicos

- Cuando accesas a un vector por medio de un vector lógico, estas filtrando a los que te dan como TRUE[4] en alguna comparación.

```
> x <- c(-2:2)
> y <- x/x
> z <- y[!is.na(y)]
> z2 <- y[!is.na(y) & x > 0]
```

- En la primera z, estamos eliminando a los valores NaN[5] como 0/0. En z2 además queremos solo los de x>0.
- Tengan en mente que la longitud de los vectores z y z2 son diferentes a la longitud de x.

Vectores Lógicos

Operadores Lógicos

En R hay diversos operadores lógicos que funcionan como en otros lenguajes con alguna pequeña diferencia. Corran los siguientes comandos para aprender como funcionan :). Para un aprendizaje más detallado lean la ayuda con `?">"` y/o `?all.equal`

```
> x <- c(1:5)
> x < 5
> x > 1
> x > 1 & x < 5
> x > 1 && x < 5
> x > 1 | x < 5
> x > 1 || x < 5
```

Vectores Lógicos


```
> x == 3
> x != 3
> !x == 3
> x == c(2, 4)
> x %in% c(2, 4)
```

[4] Muchas veces puedes usar `T` en vez de `TRUE`; eviten llamar una variable como T

[5] NaN significa Not a Number

Vector de enteros positivos

- Tal vez la forma más comun de accesar a un vector de datos es por posición. Aquí simplemente las posiciones van desde 1[6] hasta n donde n es la longitud del vector de datos.
- Si `x` tiene 100 elementos, puedes entrar a los primeros 10 usando `x[1:10]` o a los elementos 1, 5 y 8 usando `x[c(1,5,8)]`.
- Otra forma de usar esta tipo de índice sería:

```
> c("A", "T", "C", "G")[rep(c(1,
+     2, 2, 4, 3), times = 2)]
```

[6] Es diferente de Perl!

Vector de enteros negativos

- En realidad esto es muy sencillo. Simplemente son las posiciones que queremos excluir.
- En el siguiente ejemplo simplemente nos quedamos con las posiciones 1, 7, 8 y 10.

```
> x <- c("inicio", rep(c("A", "T",
+     "C", "G"), times = 2), "fin")
> y <- x[-c(2:6, 9)]
> y
```

```
[1] "inicio" "T"      "C"      "fin"
```

Por vector de caracteres

- En realidad esta forma es muy parecida a las anteriores. Simplemente tienen que poner entre comillas dobles las palabras que identifican a las posiciones.

```
> fagos <- c(233.2, 180.5, 280.3)
> names(fagos) <- c("Aeromonas phage Aeh1",
+     "Enterobacteria phage RB43",
+     "Pseudomonas phage phiKZ")
> fagos["Aeromonas phage Aeh1"]
> fagos[grep("Aeh1", names(fagos))]
```

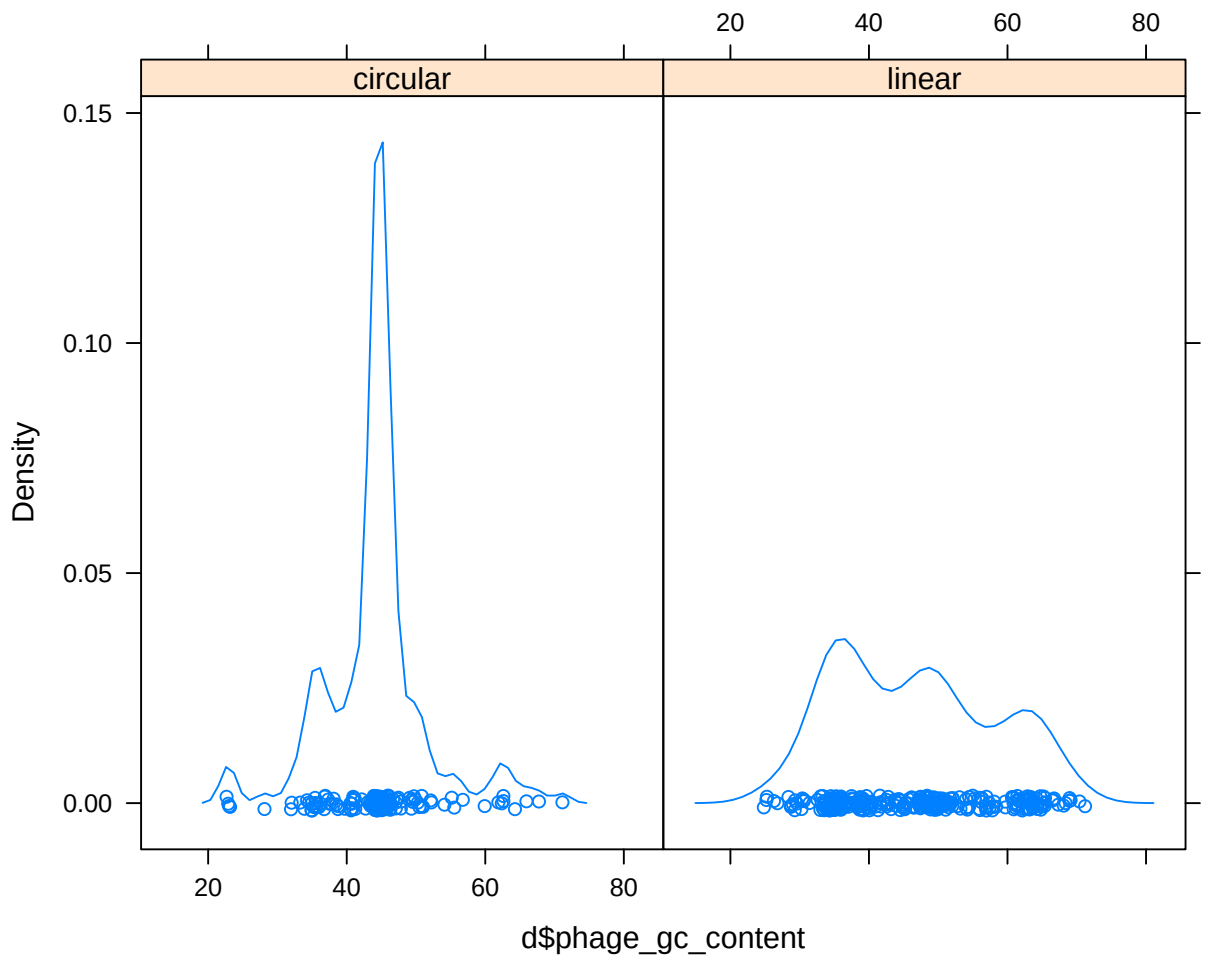
Chequen la función grep! which, match y subset también son bastante útiles!

- Para ahora ya se deben haber dado cuenta... las funciones siempre usan (...) y los vectores de datos usan [...]

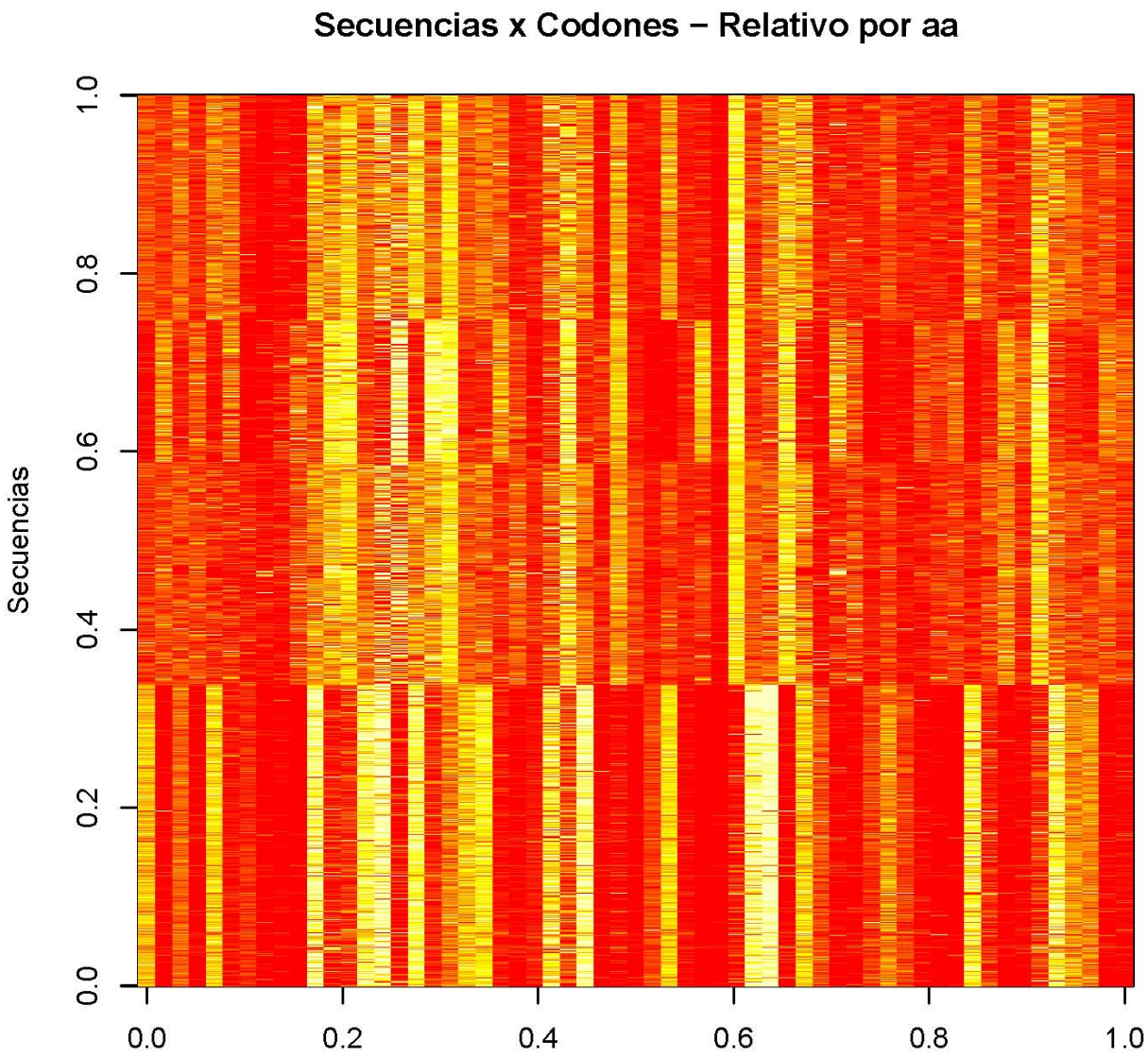
:)

- Las siguientes imágenes son unas que creamos nosotros para mostrarles el poder de R. Mas adelante podrán hacer cosas parecidas :D

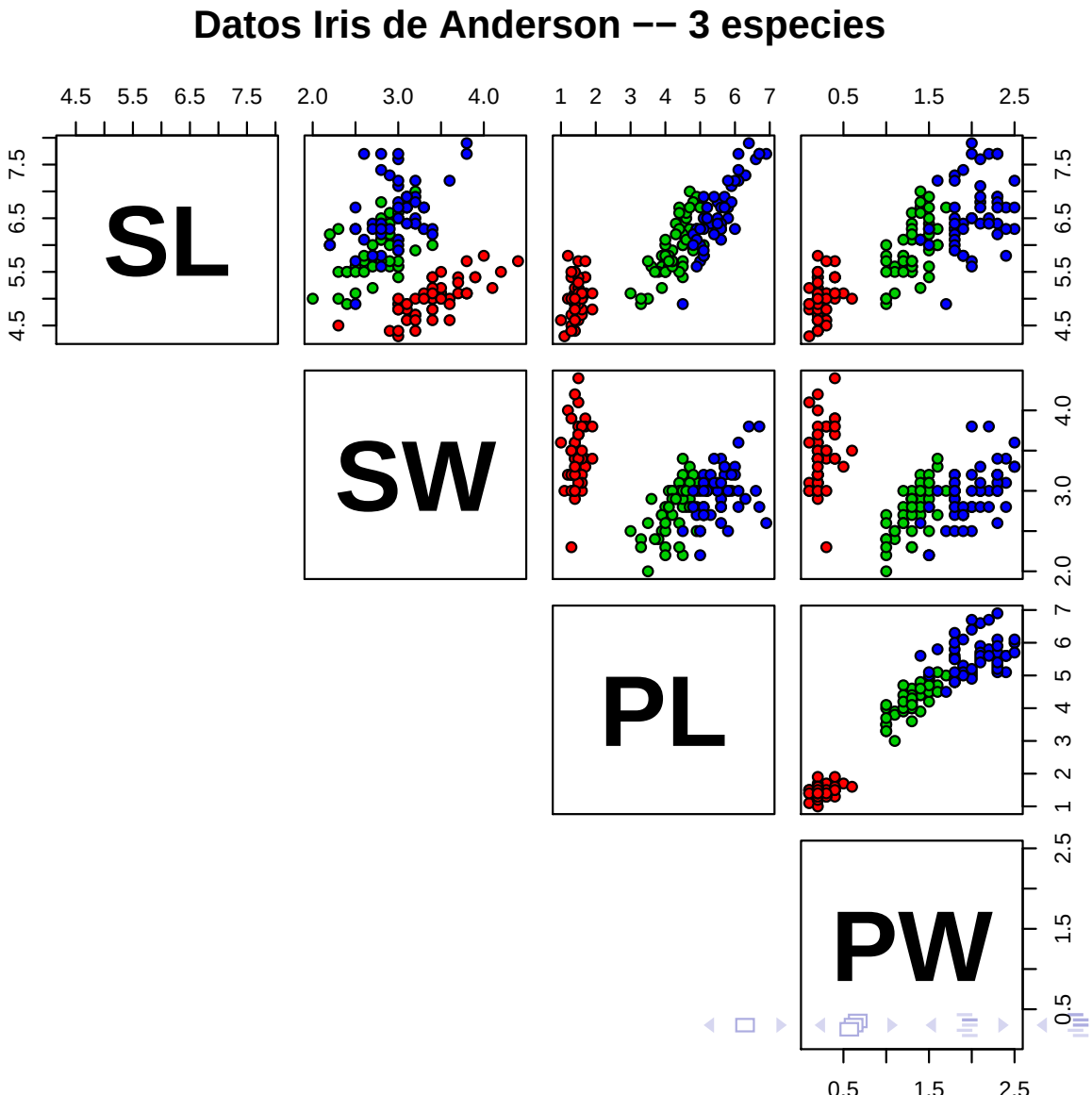
Una gráfica con lattice



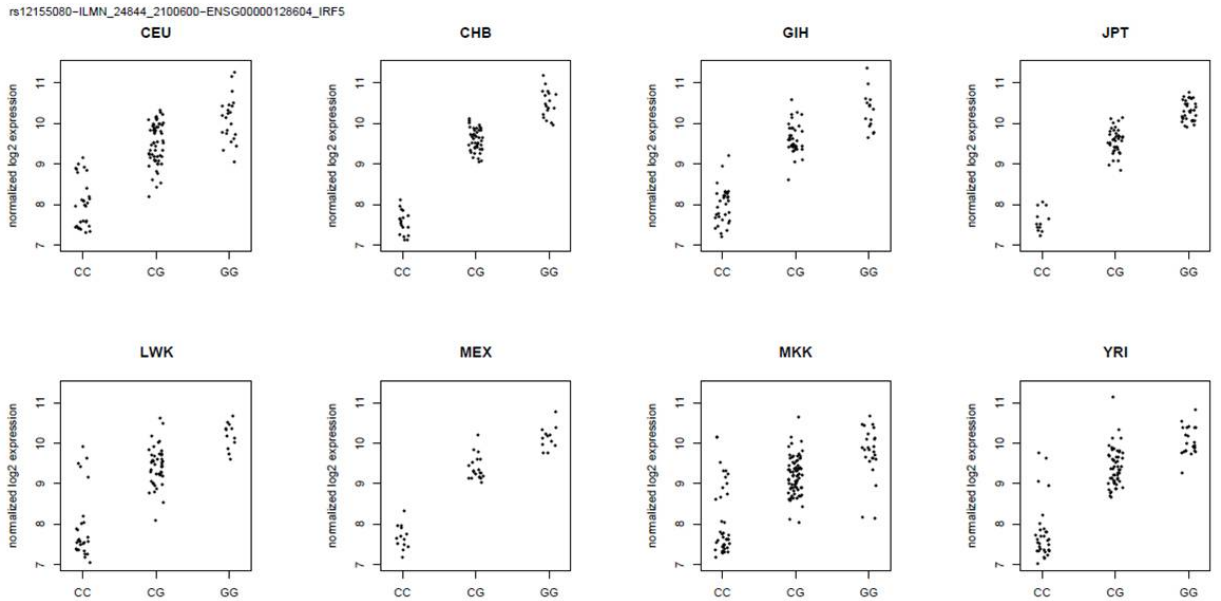
Una matriz de datos gigante



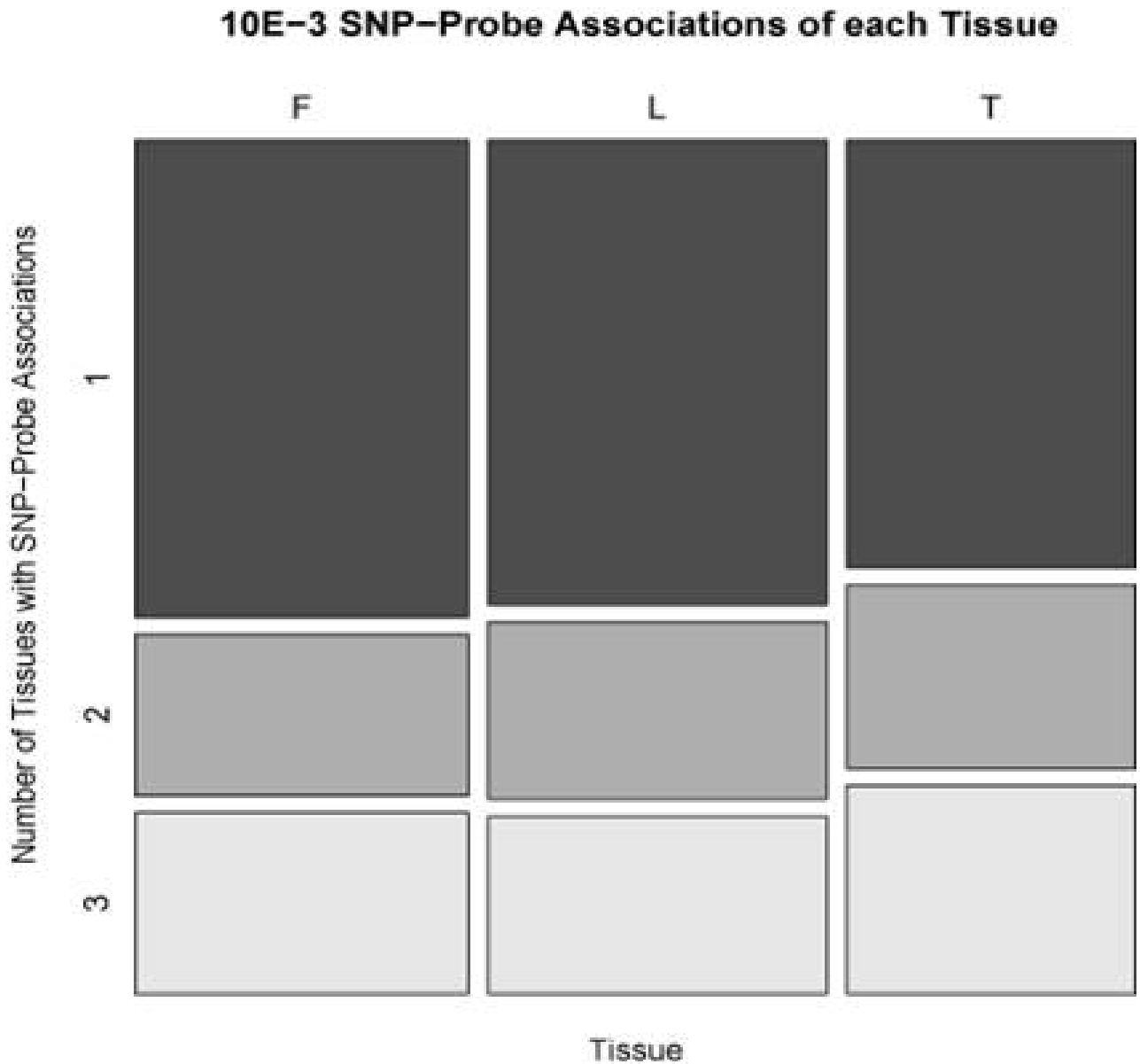
Un scatterplot de los datos Iris



Asociación de un SNP con la expresión de un gene



Una gráfica de mosaicos



Emacs

- Emacs es un editor de texto muy extenso y que nos va a facilitar el trabajo en `R`.
- Ya está instalado en su servidor pero para que corra bien creen el archivo *.emacs* en su HOME con la siguiente línea: `(require 'ess-site)`
- Ya después pueden abrirlo con el comando *emacs* y correr `R` usando M-x-R[7]

[7] M == ESC

En sus laps

- Si tiene una laptop con Windows, el archivo *.emacs* debe tener una línea extra al principio con el lugar de donde tienen instalado el ESS. Por ejemplo:
- `(load "C:/Documents and Settings/Leonardo/Desktop/Curso R/ess-5.3.6/lisp/ess-site")`
- Les recomendamos que lean los siguientes links para saber más del Emacs y de como instalarlo en Windows.