Seminario III: R/Bioconductor

Leonardo Collado Torres
lcollado@lcg.unam.mx
Licenciado en Ciencias Genómicas
www.lcg.unam.mx/~lcollado/

Agosto - Diciembre, 2009

Cómo usar R

Bienvenida

Introducción básica a R

Buscando ayuda

Objetos y estructuras en R

Leyendo archivos en R

Gráficas básicas en R

Cómo usar R

Control de flujo

Ejercicios

Y así comienza

- Primer curso exclusivo de Bioconductor de 32 horas en la LCG
- Inspirado en el BioC2009
- Todo el material esta disponible en inglés y español
- Clases en Inglés: Bioc y OCW
- Asistentes Alejandro, José y Víctor
- Página oficial del curso:
  `http://www.lcg.unam.mx/~lcollado/B/`
- Recuerden utilizar el foro para aclarar sus dudas

Programa

- Objetivos
- Proyecto: Buscar artículos de Bioconductor
- Una clase de muestra
- Evaluación
- Calendario de clases *tentativo*

Información del curso

- El curso es proporciona una visión general del curso.
- Algunos expertos en Bioconductor hicieron comentarios acerca del curso!
- El calendario guarda una estrecha relación con *Bioinformática y Estadística I*. Como es el caso de `Biostrings`.
- Buscamos que un experto nos visite :)

Grabación de videos

- Este es el curso piloto de la LCG para el OpenCourseWare
- Todas las clases serán videograbadas!
- Así que Inglés todo el tiempo
- Hay un retraso de una semana

Antecedentes de R

- R es una implementación open-source del lenguaje S: Becker, Wilks y Chambers. S-PLUS es una de uso privado.
- Creado por Ross Ihaka y Robert Gentleman[1]
- Es un lenguaje interpretado y *vive* en el momento de interpretación.
- Es muy útil como ambiente de programación: gráficas, estadísticas, paquetes biológicos (genómicos) como los de Bioconductor.
- Ciclo de lanzamientos de seis meses. Versiones estables y de desarrollo.
- R es un lenguaje multi plataforma: Windows, Linux/Unix y Mac.

Antecedentes de R

- R Core y la Comprehensive R Archive Network (CRAN) `http://cran.r-project.org`

[1] El también creó el proyecto de Bioconductor

Instalando R

- Para Windows y Mac, descarga el binario de CRAN, doble click y a seguir las instrucciones.
  - Lanzamientos estables en Windows y Estables en Mac.
- Para Linux/Unix, dependerá del sabor de tu preferencia. Suponiendo que tienes Ubuntu, entonces sigue estas instrucciones para obtener la versión estable más reciente, pues `sudo apt-get install r-base` usualmente no está actualizado para la versión más reciente.
- Para este curso necesitarán la versión de desarrollo de R develllamada 2.10.0devel y Bioconductor versión 2.5.
  - Ya está instalado en Montealban (Windows) y pronto estará en los servidores Solaris.

Una sesión básica de R

- Se recomienda el uso de Emacs o XEmacs para trabajar con R. En el último de los casos usa un editor de textos, copia y pega tus comandos [2].
- Teclea R en tu terminal o haz doble click en el ícono de R. Se muestra alguna información básica.
- Se puede usar R simplemente como una calculadora, así que escribe algunos comandos :)

```
> 2 + 3 * 5

[1] 17

> 2^3

[1] 8
```

Una sesión básica de R

```
> 6/3
[1] 2
> sqrt(pi)
[1] 1.772454
> exp(log(5))
[1] 5
```

- Se pueden insertar comentarios en el código con el símbolo de #.
- Sal usando las funciones q o quit.

```
> q("no")
```

[2] En Windows pueden usar la interfaz gráfica de R y correr sus comandos con CTRL + R.

Espacio de trabajo e historial

En algunas ocasiones es necesario interrumpir tu trabajo así que guardar tus objetos de R, historial o sesión es muy útil.

- Es posible salvar y cargar objetos especificando los objetos, su ruta y el nombre de archivo en un archivo .Rda.

```
> save(object1, object2, file = file.path("folder",
+     "file.Rda"))
> load(file = file.path("folder",
+     "file.Rda"))
```

- Para ver tus comandos recientes usa la función history. Puedes salvar y guardar tu historial usando los comandos savehistory y loadhistory.

Espacio de trabajo e historial

```
> history()
> savehistory(file = file.path("folder",
+     "file.Rhistory"))
> loadhistory(file = file.path("folder",
+     "file.Rhistory"))
```

- Puedes guardar tu sesión en un archivo `.Rdata` mediante la especificación de ello al momento de salir o con la función save.image y usando load para cargarla de nuevo.

```
> q(save = "yes")
> save.image(file = file.path("folder",
+     "file.Rdata"))
> load(file = file.path("folder",
+     "file.Rdata"))
```

Espacio de trabajo e historial

- Mientras trabajas, a veces será necesario cambiar el directorio en donde te encuentras o ver lo que estáhí contenido. Funciones como getwd, setwd, list.files() y dir() serán útiles.

Ayuda en R

Hay muchas formas de obtener ayuda en R. Se mencionan algunas a continuación.

- La función más básica para buscar ayuda es, help. Esta es una forma más corta: ?

  ```
  > help(quit)
  > `?`(q)
  ```

- Una buena herramienta es el buscador de ayuda que se acciona mediante la función help.start.Durante la sesión, las páginas de ayuda se abrirán en tu navegador.

  ```
  > help.start()
  ```

Ayuda en R

- Otras funciones que se usan frecuentemente son apropos y args. La primera enlista todas las funciones cuyo nombre incluye tu consulta y la segunda enlista los argumentos de una función determinada.

```
> apropos("history")
```

```
[1] "history"     "loadhistory"
[3] "savehistory"
```

```
> args(savehistory)
```

```
function (file = ".Rhistory")
NULL
```

- Si se desea buscar en el sitio web de R, se puede usar RSiteSearch. Por ejemplo:

Ayuda en R

```
> RSiteSearch("help")
```

- Para un paquete en específico, se puede ver alguna información básica usando la siguiente sintaxis. Intenta para el paquete `stats`.

  ```
  > library(help = packagename)
  ```

- Otra herramienta excelente es la R mailing list `https://stat.ethz.ch/mailman/listinfo/r-help`
- Lee la *posting guide*. Usar la función sessionInfo es muy importante.

  ```
  > sessionInfo()
  ```

Ayuda en R

```
R version 2.10.0 Under development (unstable) (2009-07-25 r48998)
i686-pc-linux-gnu

locale:
 [1] LC_CTYPE=en_US.UTF-8
 [2] LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=C
 [6] LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8
 [8] LC_NAME=C
 [9] LC_ADDRESS=C
[10] LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8
[12] LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices
```

Ayuda en R

```
[4] utils     datasets  methods
[7] base
```

Objetos

- Todo en R y se le puede llamar con números, letras, punto y guión bajo.[3].
- Para asignar un valor a una variable[4], se hace con el operador <- o con =. Sin embargo, es una buena práctica usar = sólo dentro de funciones o en la definición de argumentos.
- Todo objeto tiene una *clase* como `integer` y puede tener *atributos* los cuales se pueden anadir y manipular usando la función attr. Para verlos se puede usar la función attributes.

```
> x <- 1:10
> names(x) <- letters[1:10]
> attributes(x)
```

Objetos

```
$names
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i"
[10] "j"
```

- Así mismo las funciones tienen *métodos* y R soporta dos sistemas de programación orientada a objetos POO (S3 y S4) pero no hablaremos de ellos.

[3] No puede empezar con los últimos dos

[4] Lo cual crea un objeto

Vectores

- Es la estructura básica de R. Se puede crear uno, usando la función más usada en R . . . c

```
> x <- c("hola", seq(0, 25, by = 5),
+     TRUE)
> x
```

```
[1] "hola" "0"    "5"    "10"   "15"
[6] "20"   "25"   "TRUE"
```

- ¿Cuál es la clase del objeto x?
- Los *vectores atómicos* contienen valores del mismo tipo como enteros, valores lógicos o cadenas de caracteres.

Vectores

```
> y <- c(NA, sample(rep(c(TRUE, FALSE),
+     10), 4))
> y
[1]    NA FALSE FALSE FALSE  TRUE
```

- ¿Es y un vector atómico?

Un paréntesis curioso

- Teclea[5] el siguiente código:

```
> a <- sqrt(2)
> a * a == 2
> a * a - 2
```

- ¿De qué te percatas?

[5] El código de R está disponible en el sitio oficial del curso

Factores

- Son muy útiles cuando los datos se pueden categorizar. Por ejemplo, niños, adultos y personas mayores.

```
> f <- sample(c("kid", "adult", "elderly"),
+     10, replace = T)
> f <- factor(f)
> f
```

```
 [1] adult   adult   adult   elderly
 [5] adult   kid     adult   kid
 [9] elderly adult
Levels: adult elderly kid
```

- También se pueden crear factores ordenados usando la función ordered.

Listas

- Es un objeto parecido a un vector pero puede contener elementos de distintas clases, incluso otros objetos R.

```
> x <- list(name = "Leonardo", age = 22,
+       x = c(TRUE, FALSE, NA))
> x

$name
[1] "Leonardo"

$age
[1] 22

$x
[1]  TRUE FALSE    NA
```

Listas

```
> names(x)
[1] "name" "age"  "x"
> x$age
[1] 22
> x[[3]]
[1]  TRUE FALSE    NA
> y <- "name"
> x[[y]]
[1] "Leonardo"
```

Data frames y matrices

- Puedes definir una *matriz* usando la función matrix o cambiando las dimensiones de un vector con dim. Todos los valores deben ser del mismo tipo.

```
> x <- 1:4
> dim(x) <- c(2, 2)
> x[, 2]
```

```
[1] 3 4
```

- Los *data frames* son rectangulares como las matrices pero cada columna (variable) puede tener distintos tipos de datos.

```
> students <- data.frame(age = 18:21,
+     height = 170:173, passed = c(TRUE,
+         FALSE, TRUE, TRUE))
> students
```

Data frames y matrices

```
  age height passed
1  18    170   TRUE
2  19    171  FALSE
3  20    172   TRUE
4  21    173   TRUE
```

Bases

- Las dos funciones básicas para leer archivos en R son scan y read.table. Por ejemplo, read.csv es análogo a `read.table`.Checa su ayuda para más referencias.
- Leamos el archivo `stats.txt` que contiene información de varios contigs.

```
> contigs <- read.table(file = file.path("../../data",
+     "stats.txt"), header = T)
```

- La línea de arriba funciona bien para mí, pero mi estructura de archivos es diferente de la tuya.[6] Podemos resolver esto simplemente leyendo el archivo del Internet :)

```
> contigs <- read.table(file = file.path("http://www.lcg.unam.mx/~lcollado/
+     "stats.txt"), header = T)
```

[6] Usamos la función file.path para no depender de la plataforma

Explorando tu objeto

- Ahora que ya leímos un objeto, hay una serie de funciones que nos ayudarán a explorarlo.
- Pruébalas :)

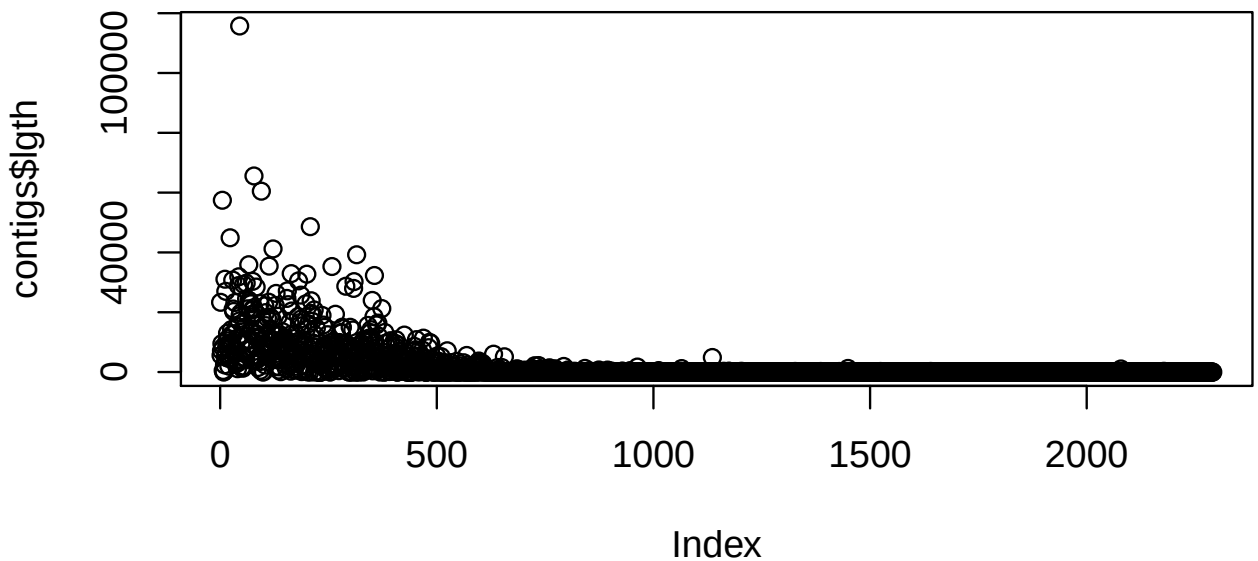
```
> class(contigs)
> object.size(contigs)
> names(contigs)
> head(contigs)
> tail(contigs)
> dim(contigs)
> summary(contigs$lgth)
```

Bases

- R es muy potente para graficar datos rápidamente.
- Algunas funciones para graficar, generan gráficas de novo mientras que otras grafican sobre una gráfica previa.
- La mayor parte de los parámetros se pasan . . . Pueden aprender más acerca de los parámetros graficos con `?par`
- `http://www.harding.edu/fmccown/R/` es muy útil para tener tips de principiante.
- Las gráficas son una parte *crucial* del Análisis exploratorio de los datos

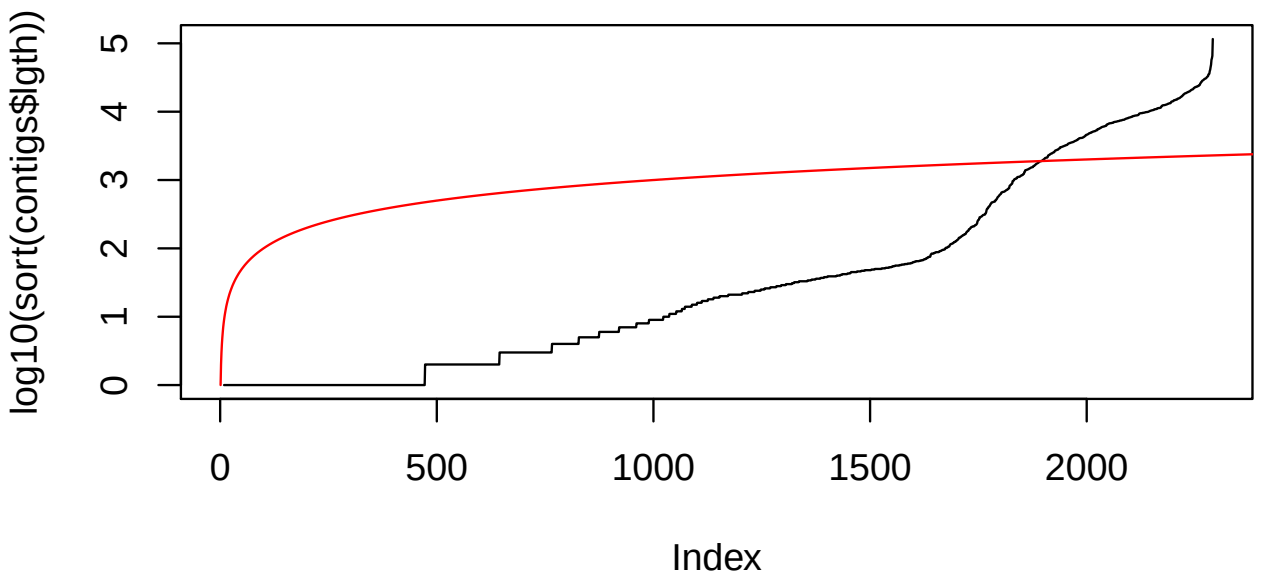
Plot

```
> plot(contigs$lgth)
```



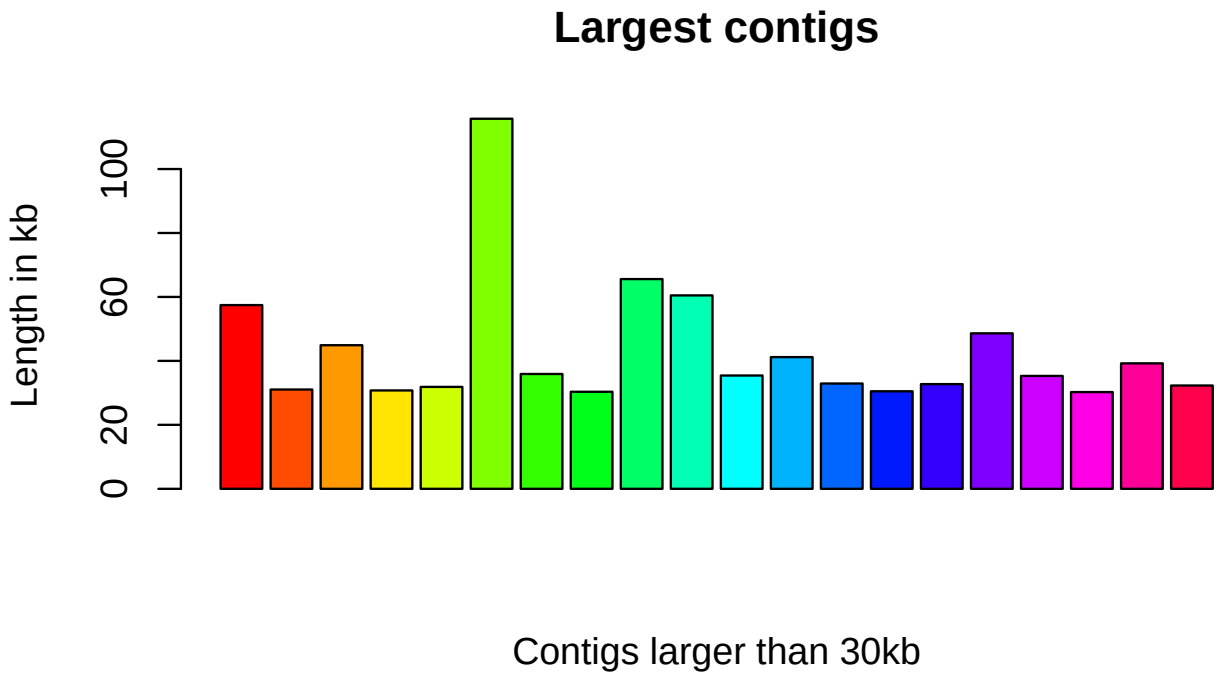
Lines

```
> plot(log10(sort(contigs$lgth)),
+      type = "l")
> lines(log10(1:length(contigs$lgth)^2),
+      col = "red")
```



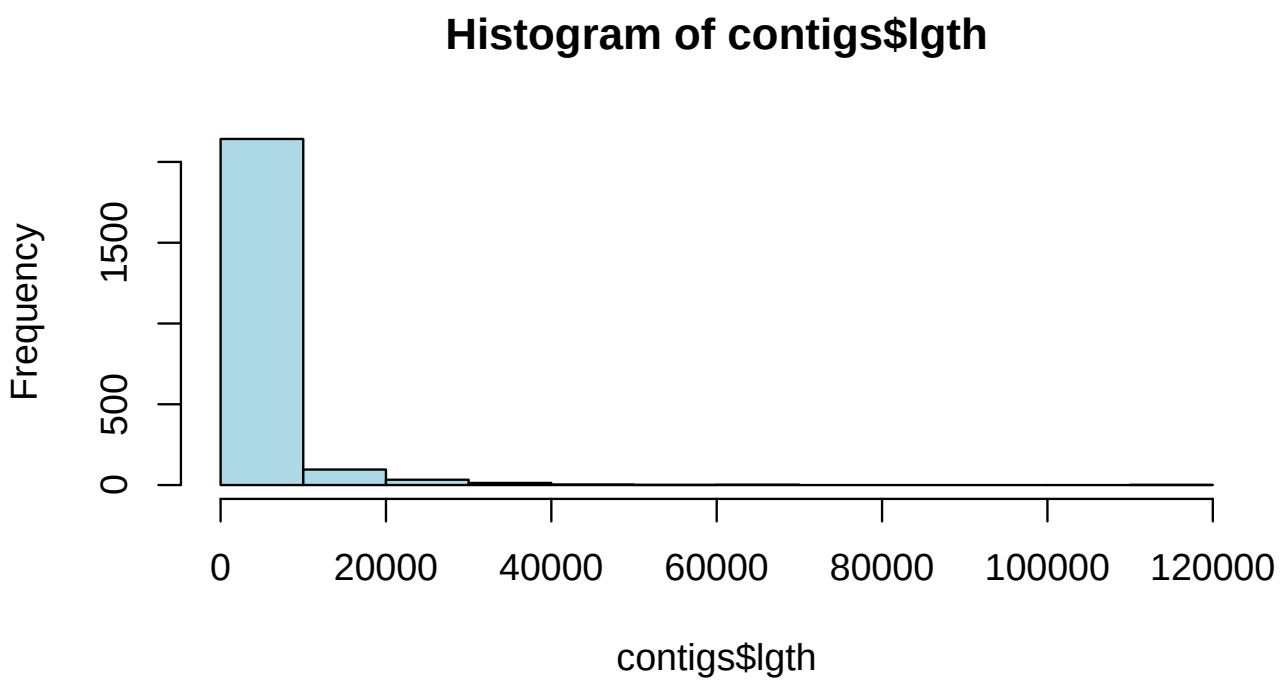
Barplot

```
> barplot(contigs$lgth[contigs$lgth >
+     30000]/1000, col = rainbow(length(contigs$lgth[contigs$lgth >
+     30000])), xlab = "Contigs larger than 30kb",
+     ylab = "Length in kb", main = "Largest contigs")
```



Histograma básico

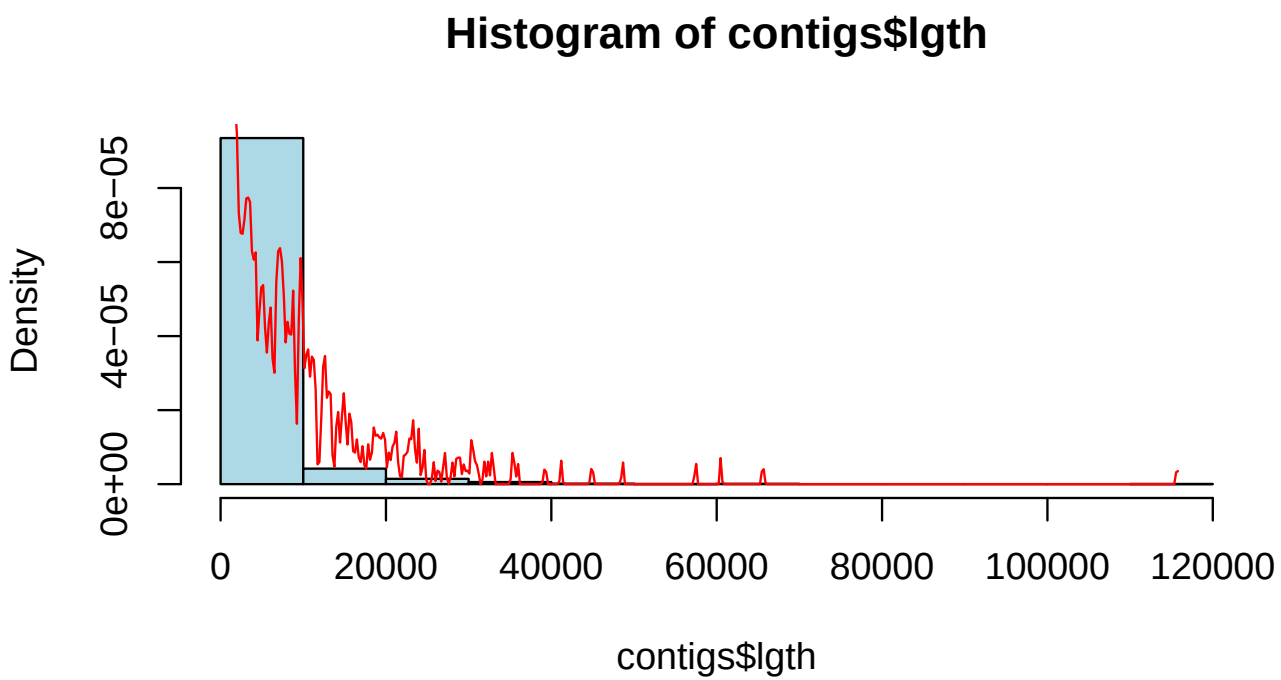
```
> hist(contigs$lgth, col = "lightblue")
```



Graficando la densidad

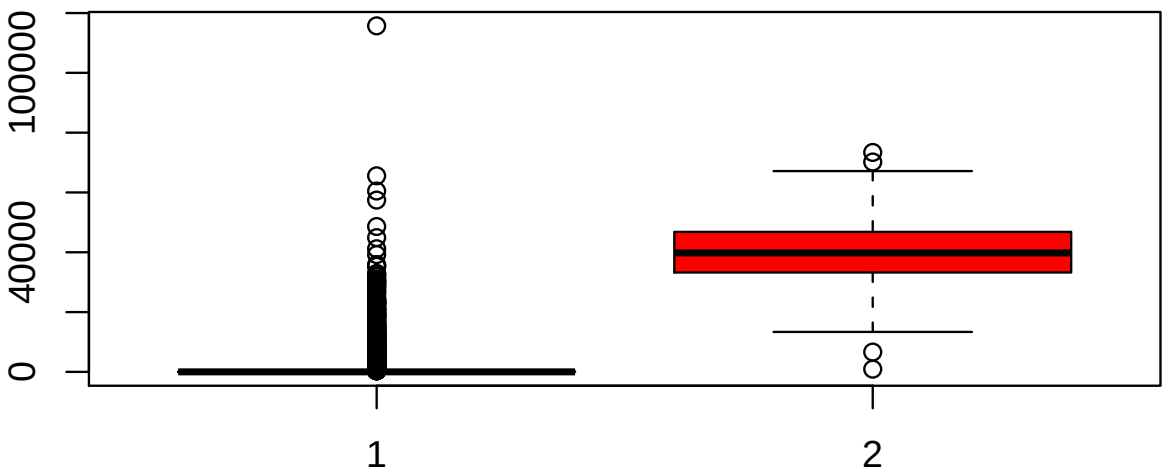
```
> hist(contigs$lgth, col = "lightblue",
+      prob = T)
> lines(density(contigs$lgth), col = "red")
```

Graficando la densidad



Resumen gráfico

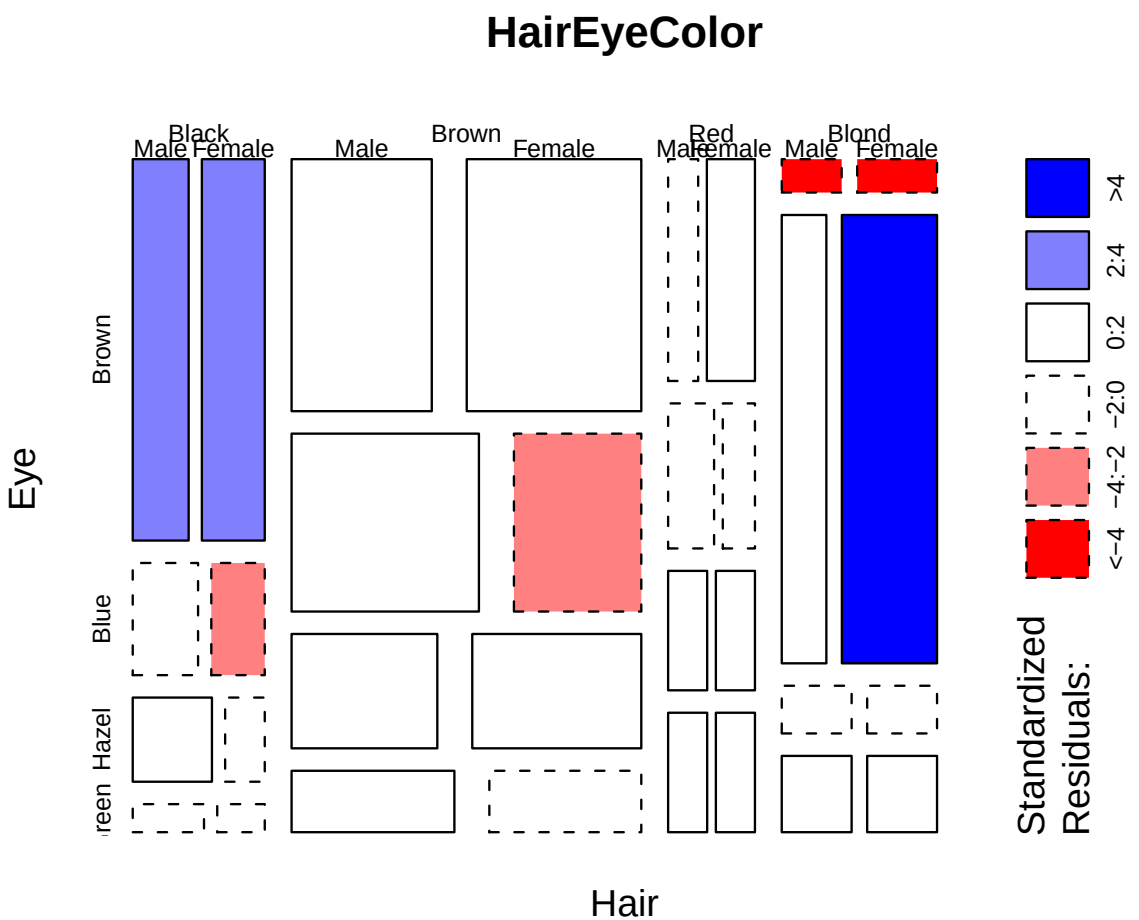
```
> boxplot(contigs$lgth, rnorm(1000,
+     40000, 10000), col = c("lightblue",
+     "red"))
```



Excelente para tablas con 3 dims

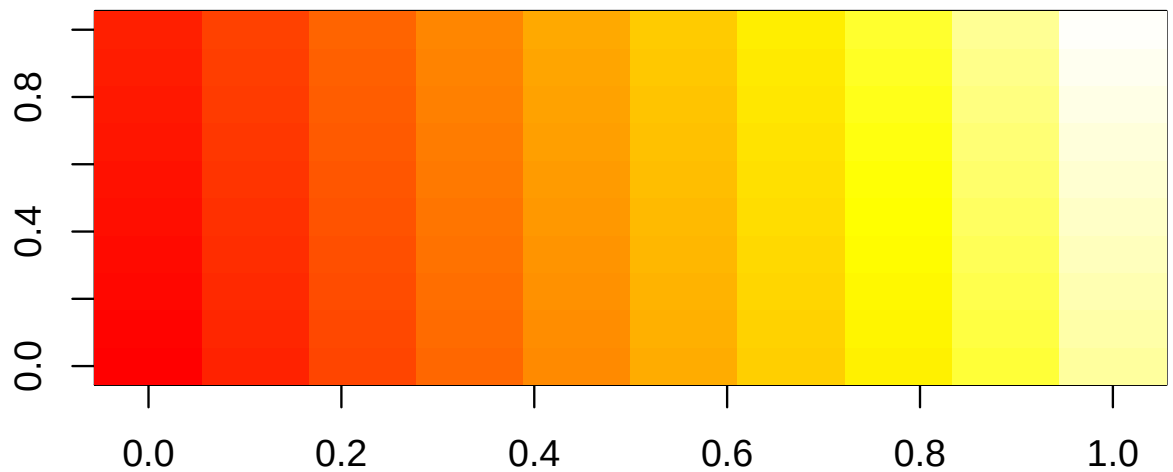
```
> mosaicplot(HairEyeColor, shade = TRUE)
```

Excelente para tablas con 3 dims



Te ayuda a visualizar tu matriz

```
> x <- matrix(1:100, 10, 10, byrow = T)
> image(x, col = heat.colors(100))
```



Exportando imágenes

- Siempre puedes exportar tus imágenes a archivos PDF o PNG.

```
> pdf(file = "file.pdf", onefile = T)
> plot("some data")
> dev.off()
> png(file = "image.png")
> plot("some data")
> dev.off()
```

Dos opciones

- While es muy fácil de usar: `while (cond) expr`

```
> x <- NULL
> while (length(x) < 10) {
+     x <- c(x, runif(1))
+ }
```

- ¿Cuál es la longitud del objeto `x`? Ahora usemos repeat con break.
- Con `while` y `repeat` se cuidadoso y evita ciclos infinitos!

Dos opciones

```
> x <- 1
> repeat {
+     x <- x + 2
+     print(x)
+     if (x > 10)
+         break
+ }
```

```
[1] 3
[1] 5
[1] 7
[1] 9
[1] 11
```

Una alternativa

- La forma más común de iteración es un ciclo for: `for (var in seq) expr`

```
> for (i in seq_len(3)) print(i)

[1] 1
[1] 2
[1] 3

> for (i in letters[4:6]) print(i)

[1] "d"
[1] "e"
[1] "f"
```

- Usar `seq_len` es lo más recomendado en vez de usar `1:length(object)`

Una alternativa

- Como querrás usar condiciones if, ifelse y switch serán de tu interés.

Bases

- Es muy fácil escribir tus propias funciones de R usando function.
- Así como puede aceptar muchos argumentos de entrada, sólo regresa un objeto que puede ser un vector.
- El objeto que se regresa es o el último en ser evaluado o alguno especificado con la función return.
- Si se usa un objeto x dentro de la función, éste no estará relacionado con la variable x fuera de la función .[7]

```
> x <- 5
> y <- function(x) rnorm(x)
> y(2)
```

```
[1]  0.3796838 -0.8461351
```

Bases

```
> x
[1] 5
```

[7] Los usuarios más curiosos pueden buscar guías acerca de los ambientes

Una familia ejemplar

- Su mayor utilidad es *aplicar* una función a todos los elementos de un objeto. Por ejemplo, todos los elementos de una matriz.
- En muchos de los casos, es simplificado o en algunos otros es un argumento.
- Es más facil para una persona entender un script con funciones apply que con ciclos `for`.

```
> mat <- matrix(rnorm(100), 10, 10)
> apply(mat, 1, sum)
```

```
 [1]  2.241366 -3.799676  2.924169
 [4]  0.905126  7.618892  2.001233
 [7] -6.468566 -3.374802 -3.162069
[10] -3.786084
```

Una familia ejemplar

- Recuerda que algunas funciones de R son más rápidas usando apply, como rowMeans.

```
> apply(mat, 1, sum) == rowSums(mat)
```

```
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [8] TRUE TRUE TRUE
```

- Algunos paquetes implementan nuevas funciones apply pero las más comunes son:
  - apply útil para matrices y data frames
  - lapply La versión para listas
  - sapply La más sencilla (listas y vectores).

```
> x <- list(rnorm(100), runif(100),
+     rlnorm(100))
> sapply(x, quantile)
```

Una familia ejemplar

```
             [,1]       [,2]       [,3]
0%   -2.42221515 0.03082297  0.1807016
25%  -0.58079951 0.26005973  0.5831613
50%  -0.01462831 0.47147562  1.1230483
75%   0.77904047 0.70181975  2.2492986
100%  2.59119996 0.98880743 12.9508204
```

- `tapply` Usa un vector y un factor, muy útil para datos agrupados.

```
> x <- data.frame(info = rnorm(10),
+     group = as.factor(sample(1:3,
+         10, replace = T)))
> tapply(x$info, x$group, mean)
```

```
         1          2          3
 0.2763497 -0.8849198  0.5146212
```

Una familia ejemplar

- `eapply` Para ambientes y los curiosos.
- `mapply` Versión multivariada de `sapply`

```
> mapply(rep, 1:4, 4:1)
```

```
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

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

- `rapply` Versión recursiva de `lapply`

Una familia ejemplar

- Este sitio puede parecerte útil: advanced_function_r.htm

o tarea :P

- Ve al sitio oficial del curso y realiza el primer ejercicio.
- Las especificaciones de la tarea están en el `syllabus`.
  - Para esta tarea sólo entrega un script portable `.R` con comentarios. La siguiente semana aprenderemos acerca de los archivos `Sweave` y *vignette*.