Ciclo de clases en bioinformática: Principios de R

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

Instituto de Biotecnología (IBT) de la UNAM y Winter Genomics (WG)

Octubre - Noviembre, 2009

Graficando con R II

Ejercicios I

- ¿Qué hace `lty`?
  Con este argumento especificamos el tipo de línea.
- Si quiero cuatro colores con la función `rainbow`, ¿cómo los obtengo?
  Usando:

  ```
  > rainbow(4)
  ```

  ```
  [1] "#FF0000FF" "#80FF00FF" "#00FFFFFF"
  [4] "#8000FFFF"
  ```
- Si una línea la quiero 3 veces más gruesa, ¿cómo lo hago?
  Usando el argumento `lwd` e igualandolo a 3.
- ¿Qué argumento uso para reducir el tamaño de un punto?
  Chequen la ayuda de `par`.
  Es el argumento cex.

Ejercicios II

- Creen los siguientes vectores:

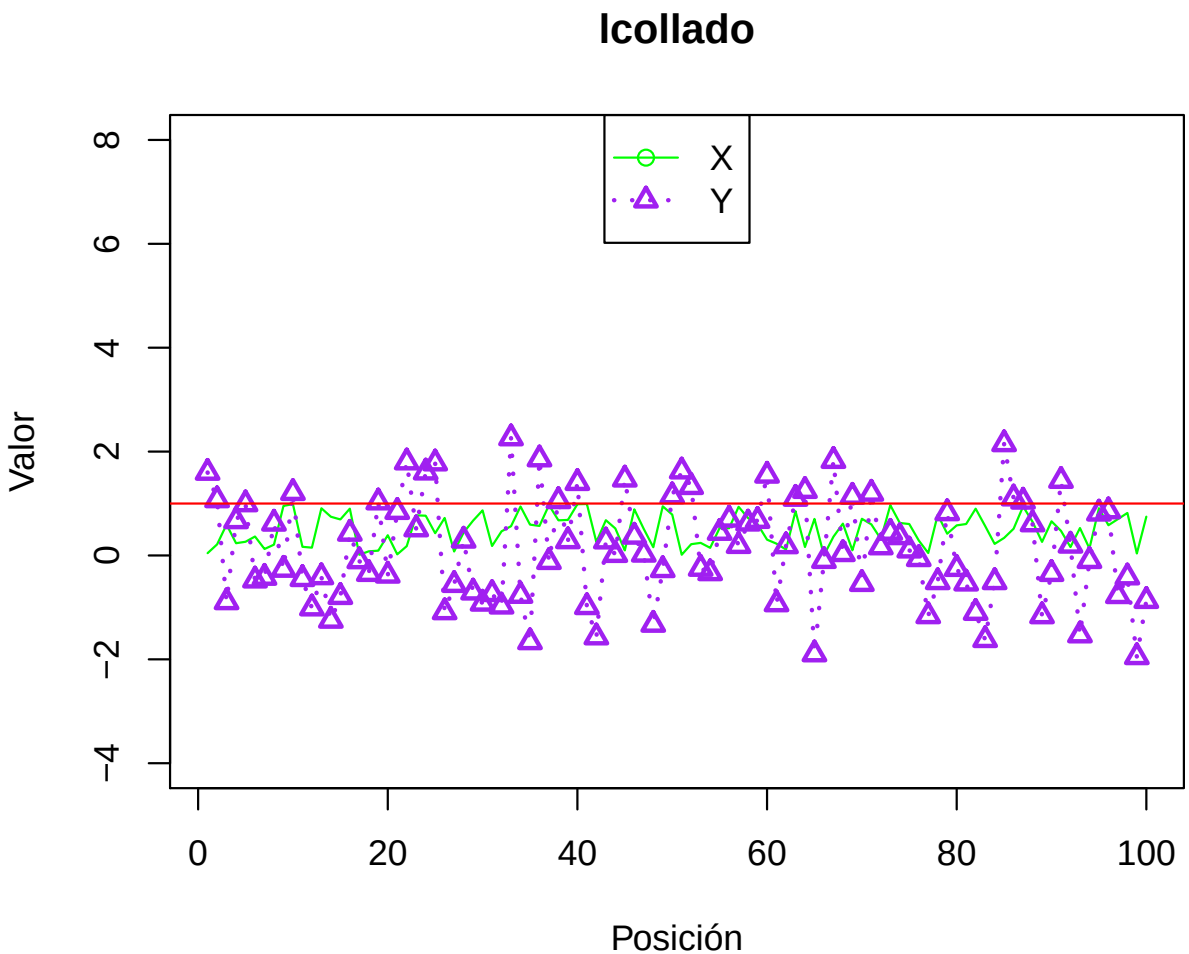
```
> x <- runif(100)
> y <- rnorm(100)
```

- Hagan una gráfica donde veamos esos puntos como líneas. Usen diferentes colores, diferentes puntos, diferente ancho de línea y diferente tipo de línea.
- Nombren el eje Y: *Valor*.
- Nombren el eje X: *Posición*.
- Pongan su usuario del dotProject como título.
- Agreguen una línea roja en $Y = 1$.
- Agreguen la *legend*. Para que no se sobrelape con los puntos, usen el argumento `ylim` en la función `plot`.

Solución

```
> plot(x, col = "green", ylab = "Valor",
+     xlab = "Posición", main = "lcollado",
+     ylim = c(-4, 8), type = "l")
> lines(y, col = "purple", lwd = 2,
+     lty = 3, pch = 2, type = "o")
> abline(a = 1, b = 0, col = "red")
> legend("top", c("X", "Y"), col = c("green",
+     "purple"), lwd = c(1, 2), lty = c(1,
+     3), pch = c(1, 2))
```

Solución



Ejercicios III

- Creen el siguiente vector:

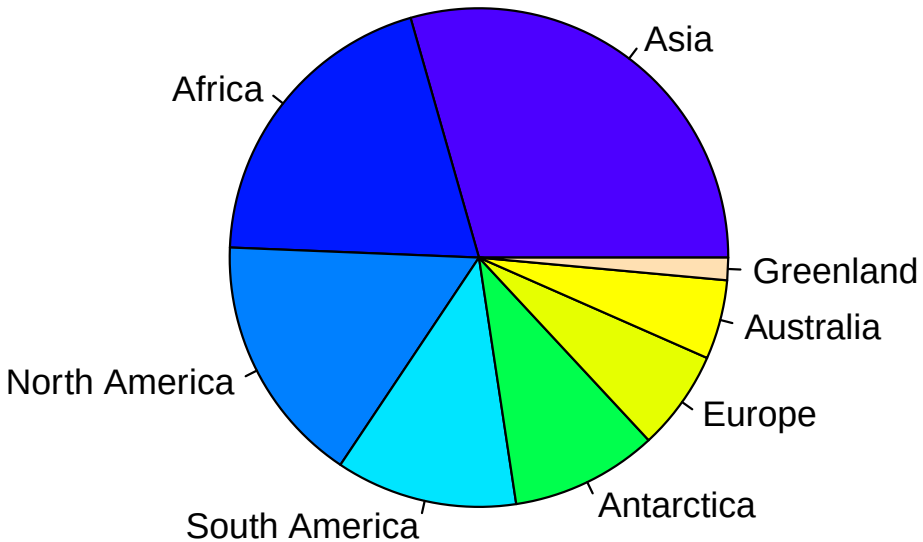
  ```
  > islas <- sort(islands, decreasing = TRUE)[1:8]
  ```

- Hagan una gráfica de pie usando los colores de la función `topo.colors`.
- Hagan una gráfica de barras usando los colores de la función `rainbow`. Pónganle un título.
- Los nombres no salen completos, así que usen $las = 2$ y el argumento cex. . . para que se vean los nombres.
- Agreguen la *legend* en la parte superior del lado derecho. Los colores tienen que ser los mismos.

Solución parte I

```
> pie(islas, col = topo.colors(8))
```

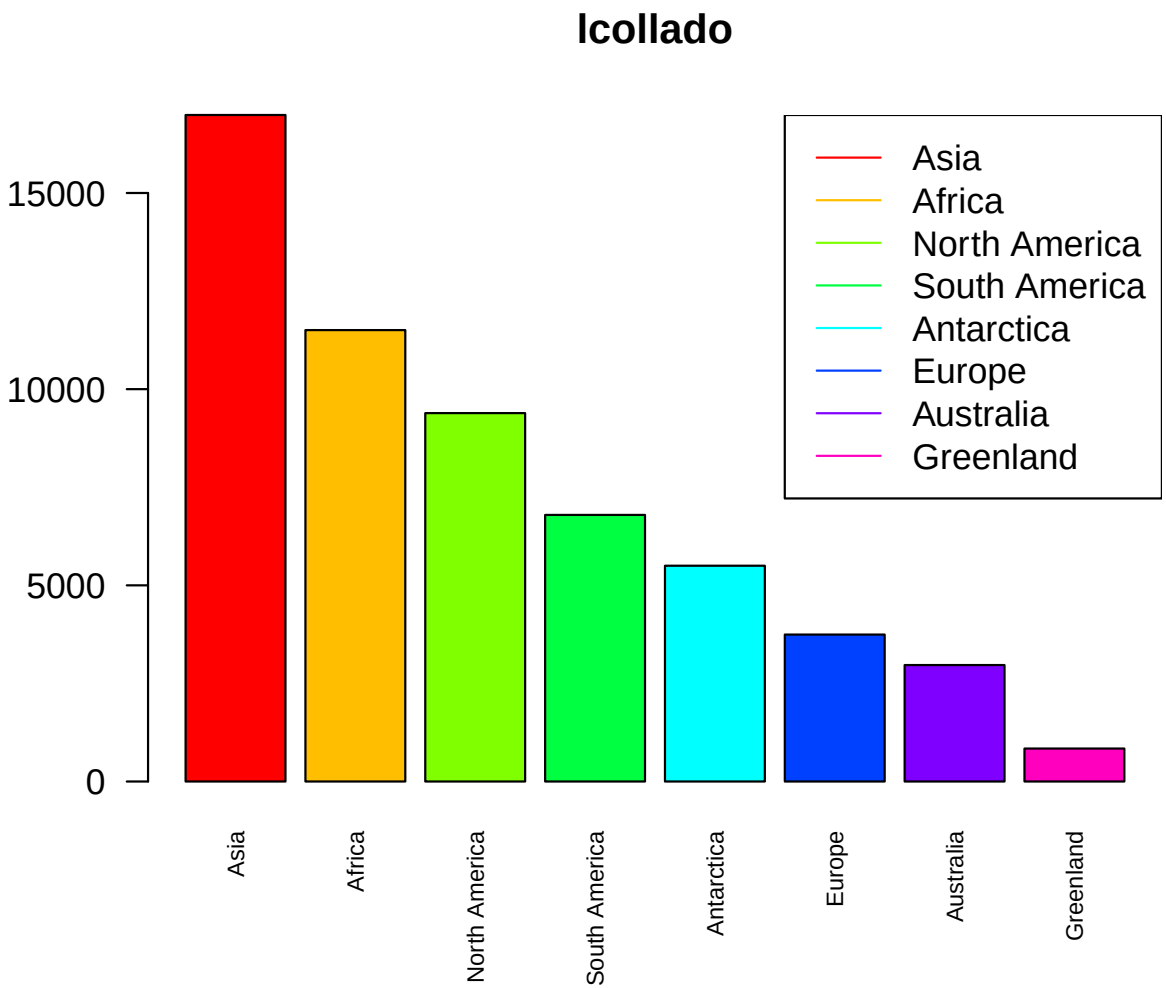
Solución parte I



Solución parte II

```
> barplot(islas, col = rainbow(8),
+     main = "lcollado", las = 2,
+     cex.names = 0.7)
> legend("topright", names(islas),
+     col = rainbow(8), lty = 1)
```

Solución parte II



Dimensiones en vectores

- Hasta ahorita hemos creado vectores, por ejemplo:

```
> v <- rnorm(9)
> v
[1] -1.1103050  3.4416022 -0.5028081
[4]  0.1045119  0.2574095  0.2529190
[7]  1.1236075 -0.1259004  1.0169894
```

- ¿Pero cúantas dimensiones tiene?

Función dim

- Aunque uno se lo imagina con 1 dimensión, en realidad no tiene:

```
> dim(v)

NULL
```

- Tenemos 9 elementos. Si queremos ponerlos en una tabla de 3x3 podemos usar de nuevo la función dim:

```
> dim(v) <- c(3, 3)
> dim(v)

[1] 3 3

> v
```

Función dim

```
            [,1]       [,2]       [,3]
[1,] -1.1103050  0.1045119  1.1236075
[2,]  3.4416022  0.2574095 -0.1259004
[3,] -0.5028081  0.2529190  1.0169894
```

- ¿Cómo asigna los valores?

Matrix

- ¿Nuestro objeto *v* sigue siendo un vector?

```
> class(v)
```

```
[1] "matrix"
```

- No, es una matriz.
  Si bien podemos crear matrices de esta forma, mejor usemos la función . . . matrix:

```
> args(matrix)
```

```
function (data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames =
NULL
```

Una matriz

- Creemos el objeto x tipo matriz con 6 números al azar usando `runif`, con 2 líneas y 3 columnas:

```
> x <- matrix(data = runif(6), nrow = 2,
+     ncol = 3)
```

- Ahora chequen su objeto x:

```
> x
```

```
          [,1]      [,2]      [,3]
[1,] 0.5244717 0.8290218 0.7597640
[2,] 0.0469955 0.3928759 0.5284191
```

- ¿Cómo diferencían el nombre de la columna 1 del nombre de la línea 1?

Accesando una matriz

- Al igual que con los vectores, para accesar una matrix podemos usar:
  1. vectores de enteros positivos,
  2. enteros negativos,
  3. lógicos
  4. los nombres (si tiene)

```
> x[1, ]
```

```
[1] 0.5244717 0.8290218 0.7597640
```

```
> x[, -2]
```

```
           [,1]      [,2]
[1,] 0.5244717 0.7597640
[2,] 0.0469955 0.5284191
```

```
> x[c(FALSE, TRUE), ]
```

```
[1] 0.0469955 0.3928759 0.5284191
```

Accesando una matriz

- ¿Cómo accesarían el valor en la línea 2 y columna 3?

Nombres

- Recuerden que va primero la línea y luego la columna, así que sería con la siguiente expresión:

```
> x[2, 3]

[1] 0.5284191
```

- Para agregar nombres es muy similar a los vectores, pero con las funciones colnames y rownames:

```
> colnames(x) <- letters[1:3]
> rownames(x) <- c("Linea1", "Linea2")
> x

              a         b         c
Linea1 0.5244717 0.8290218 0.7597640
Linea2 0.0469955 0.3928759 0.5284191

> x[, "a"]
```

Nombres

```
   Linea1    Linea2
0.5244717 0.0469955
```

- O si quieren usen la función dimnames, aunque usa un tipo de objeto que no hemos visto: list.

Dos tips

- Como hemos visto, las matrices se llenan por columnas. Sin embargo, muchas veces queremos llenar los datos por línea.
  Para eso usamos el argumento byrow:

```
> y <- runif(6)
> z <- matrix(y, 2, 3, byrow = T)
> y
[1] 0.5086079 0.6405159 0.7187039
[4] 0.6943053 0.5018389 0.1053259

> z
          [,1]      [,2]      [,3]
[1,] 0.5086079 0.6405159 0.7187039
[2,] 0.6943053 0.5018389 0.1053259
```

Dos tips

- Otros probablemente les va a gustar esta función, edit. Pruébenla:

  ```
  > edit(x)
  ```

Data frames

- Como han visto, una matriz de 2 dimensiones es una tabla. Se pueden hacer matrices de 3 dimensiones usando dim.
- Un data.frame es muy similar a una matriz, pero donde cada columna puede tener diferentes tipos de dato. Por ejemplo, nombres en la columna 1 y valores numéricos en la columna 2.
- Estos surgieron con la idea de guardar las mediciones de diferentes variables para experimentos replicados. Son prácticamente como las hojas de cálculo de Excel.
- Creemos uno:

```
> df <- data.frame(x = c("Juan",
+     "Pedro", "Maria"), y = runif(3),
+     z = c("IBT", "CCG", "LCG"))
> class(df)
```

Data frames

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

```
> df
```

```
      x         y   z
1  Juan 0.5999036 IBT
2 Pedro 0.2602036 CCG
3 Maria 0.8126240 LCG
```

Una forma especial de accesar a los datos

- En los data frames podemos usar el símbolo de pesos para accesar a las diferentes variables (columnas):

```
> df$x
```

```
[1] Juan  Pedro Maria
Levels: Juan Maria Pedro
```

- Obtengan los últimos 2 valores de la variable y de nuestro data frame.

Para escoger

Tenían dos opciones:

- Usarlo como matriz:

```
> df[2:3, 2]
```

```
[1] 0.2602036 0.8126240
```

- O usar el símbolo de pesos:

```
> df$y[2:3]
```

```
[1] 0.2602036 0.8126240
```

- Generalmente es más fácil entender un código si usan la segunda opción. Eso si, escogan nombres cortos pero significativos para sus variables (columnas).

Función t

- En R hay una gama de funciones para álgebra matricial, aunque la que usarán más frecuentemente es la función transpuesta, `t`:

```
> t(x)
```

```
     Linea1    Linea2
a 0.5244717 0.0469955
b 0.8290218 0.3928759
c 0.7597640 0.5284191
```

- ¿Qué pasa si la usan con un data frame?

Con data frames

- Se convierte en una matriz:

```
> class(t(df))

[1] "matrix"

> t(df)

  [,1]       [,2]       [,3]
x "Juan"     "Pedro"    "Maria"
y "0.5999036" "0.2602036" "0.8126240"
z "IBT"      "CCG"      "LCG"
```

- Como ven, convirtió todos nuestros datos en tipo `character`.

Para leer una tabla

- Muchas veces van a tener datos tabulares. Es decir, separados por `tab`
- Por ejemplo, este archivo.
- La función principal para leer este tipo de archivos en R es read.table:

```
> big <- read.table("http://www.lcg.unam.mx/~lcollado/R/data/10bigge
+     header = TRUE)
```

- Usé el argumento header igual a `TRUE` porque el archivo tiene los nombres de las columnas en la primera línea.

Tips

- Como vieron, podemos leer tablas desde la red usando esta función.
- Si tienen el archivo en su compu, fíjense en que directorio están usando R, porque de eso depende la ruta para un archivo.
- Funciones como getwd y setwd les pueden ayudar bastante (wd es *working directory*):

```
> getwd()
```

```
[1] "C:/Documents and Settings/Leonardo/My Documer
```

- Algo que puede serles util es la función file.choose:

```
> tabla <- read.table(file = file.choose())
```

CSV

- Otro formato de archivo que van a encontrar frecuentemente es el CSV por *comma separated values*.
- Para leer este tipo de archivos hay que usar la función read.csv:

```
> mirnas <- read.csv("http://www.lcg.unam.mx/~lcollado/E/data/mirnas
+     header = T)
```

Read delim

- Una tercera función, y que uso frecuentemente, es la función read.delim
- Es útil cuando la tabla tiene símbolos que normalmente provocarían que R muera, como el símbolo de gato o una apóstrofe.

```
> phage <- read.delim(file.path("ftp://ftp.ebi.ac.uk/pub/databases/g
+     header = F)
```

Head y Tail

- Muchas veces, estas tablas son grandes:

```
> dim(big)
[1] 10  4
> dim(mirnas)
[1] 396   8
> dim(phage)
[1] 602   6
```

- Las funciones head y tail van a ser muy útiles:

```
> head(big)
```

Head y Tail

```
                      Name GenomeSize
1 Pseudomonas phage phiKZ     280334
2       Cyanophage P-SSM2     252401
3     Bacteriophage KVP40     244834
4    Aeromonas phage Aeh1     233234
5    Pseudomonas phage EL     211215
6  Cyanophage phage S-PM2     196280
         EMBL  Taxid
1 AF399011_GR 169683
2 AY939844_GR 268746
3 AY283928_GR  75320
4 AY266303_GR 227470
5 AJ697969_GR 273133
6 AJ630128_GR 238854
```

```
> tail(phage)
```

Head y Tail

```
              V1     V2
597 AP008986_GR 331627
598 AY299121_GR 232237
599 AY986977_GR 322855
600 DQ777876_GR 470314
601 AY247822_GR 227720
602 EU734170_GR 532078
                                  V3
597          Xanthomonas phage OP2
598         Xanthomonas phage Xp10
599         Xanthomonas phage Xp15
600     Xanthomonas phage phiXo411
601 Yersinia pestis phage phiA1122
602           Yersinia phage Yepe2
            V4    V5    V6
597 Chromosome 27828 9CAUD
598 Chromosome 28213 9CAUD
599 Chromosome 27782 9CAUD
600 Chromosome 29301 9CAUD
601 Chromosome 28205 9CAUD
602 Chromosome 31357 9CAUD
```

Head y Tail

- Con esas funciones podemos ver las primeras 6 primeras o últimas líneas de nuestros data frames.
- ¿Cómo verían las últimas 10 líneas del objeto `mirnas`?

Otras dos opciones

- Una es que se fijen en las dimensiones y usen un vector de enteros positivos:

```
> dim(mirnas)

[1] 396   8

> mirnas[387:396, ]

       miRNAs X5..end.nucleotide Total
387    miR867                  U     4
388   miR867*                  C     0
389    miR868                  C     0
390   miR868*                  A     0
391  miR869.1                  A     0
392 miR869.1*                  A     0
393  miR869.2                  U    28
394 miR869.2*                  A     0
395    miR870                  U     4
396   miR870*                  U     0
    AGO1 AGO2 AGO4 AGO5
387   12    0    6   61
388    0    0    0    3
389    4    0    0    0
```

Otras dos opciones

```
390    0    0    0    0
391    0   44    1    0
392    0    0    0    0
393   47    0    3    0
394    0    0    0    0
395    4    0    0    0
396    0    0    0    0
    Free.energy..kcal.mol.a
387                  -9.100
388                  -6.550
389                  -4.225
390                  -6.850
391                  -8.950
392                  -7.000
393                 -10.950
394                  -3.875
395                  -7.000
396                 -10.800
```

- La otra es que usen la función tail con el argumento n:

```
> tail(mirnas, n = 10)
```

Otras dos opciones

```
      miRNAs X5..end.nucleotide Total
387    miR867                 U     4
388   miR867*                 C     0
389    miR868                 C     0
390   miR868*                 A     0
391  miR869.1                 A     0
392 miR869.1*                 A     0
393  miR869.2                 U    28
394 miR869.2*                 A     0
395    miR870                 U     4
396   miR870*                 U     0
    AGO1 AGO2 AGO4 AGO5
387   12    0    6   61
388    0    0    0    3
389    4    0    0    0
390    0    0    0    0
391    0   44    1    0
392    0    0    0    0
393   47    0    3    0
394    0    0    0    0
395    4    0    0    0
396    0    0    0    0
```

Otras dos opciones

	Free.energy..kcal.mol.a
387	-9.100
388	-6.550
389	-4.225
390	-6.850
391	-8.950
392	-7.000
393	-10.950
394	-3.875
395	-7.000
396	-10.800

Tablas complicadas

- Si no logran leer sus datos usando las funciones anteriores, y eso que tienen muchos argumentos, siempre pueden recurrir a la función scan.
- Aunque les recomiendo que exploren las otras funciones antes de recurrir a esta función.

```
> `?`(scan)
```

Las bases

- Hasta ahorita hemos explorado las gráficas usando R.
- Una vez que ya tienen una gráfica bonita que quieren guardar, hay una gama de funciones que nos permiten hacerlo.
- Por ejemplo, en formato JPG:

```
> jpeg(file = "imagen1.jpg")
> pie(islas, col = topo.colors(8))
> dev.off()

pdf
  2
```

- Todas funcionan igual:
  1. Llamas la función y especificas el nombre del archivo de salida.
  2. Usas la función (o funciones) gráfica de R.

Las bases

3. Finalmente usas la función dev.off para que se cree el archivo.

PNG

- En el caso de la función `jpeg`, nuestra imagen resultante es de 480 por 480 pixeles. Pueden hacer su imagen más grande usando los argumentos de la función.
- Si prefieren guardar la imagen en formato PNG solo usen la función del mismo nombre:

```
> png(file = "imagen2.png")
> pie(islas, col = topo.colors(8))
> dev.off()

pdf
  2
```

- La imagen en formato PNG sale del mismo tamaño.

El mejor: PDF

- Sin embargo, la mejor opción es guardar las imágenes en formato PDF.
- Si recuerdan, las gráficas en R están vectorizadas, y el formato PDF lo soporta.

```
> pdf(file = "imagen3.pdf")
> pie(islas, col = topo.colors(8))
> dev.off()

pdf
  2
```

- Además, puedes guardar más de una imagen usando el argumento `onefile`:

El mejor: PDF

```
> pdf(file = "imagenes.pdf", onefile = T)
> pie(islas, col = topo.colors(8))
> barplot(islas, col = topo.colors(8))
> dev.off()

pdf
  2
```

EDA

- Una gran rama de la estadística es la EDA por *exploratory data analysis*
- Es muy importante explorar los datos que uno tiene para poder entenderlos mejor.
- R tiene muchas funciones que nos generan gráficas interesantes, por ahora veremos las básicas

El histograma

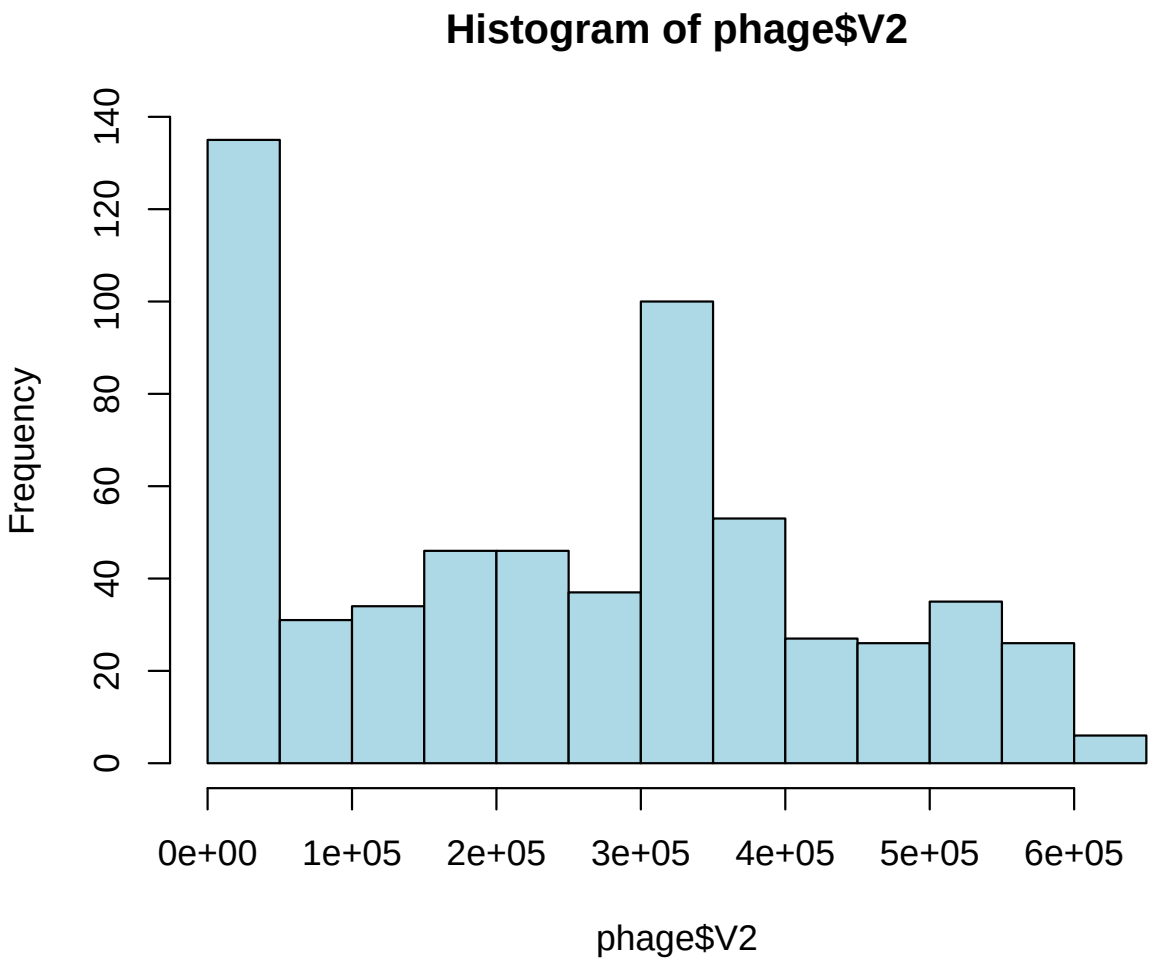
Usemos la columna dos del objeto `phage`. Son los tamaños de los genomas.
Creemos un histograma de los datos usando hist:

```
> length(phage$V2)
```

```
[1] 602
```

```
> hist(phage$V2, col = "light blue")
```

El histograma

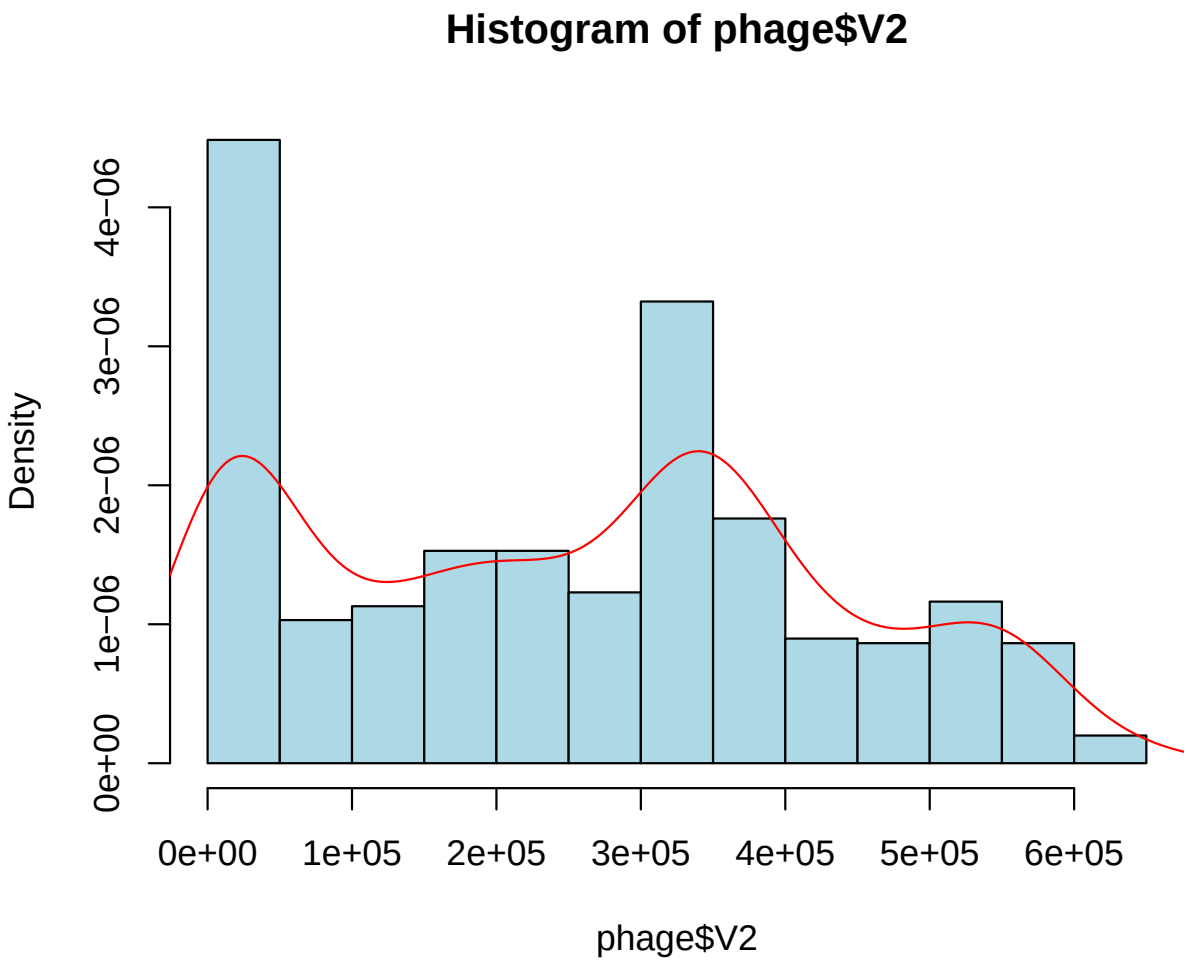


Con la densidad

Automáticamente nos define el tamaño de las cajas. Si quieren pueden especificarlos usando el argumento `breaks`.
Además, podemos graficar la densidad usando density.
¿Qué notan de diferente en la gráfica?

```
> hist(phage$V2, col = "light blue",
+      prob = T)
> lines(density(phage$V2), col = "red")
```

Con la densidad

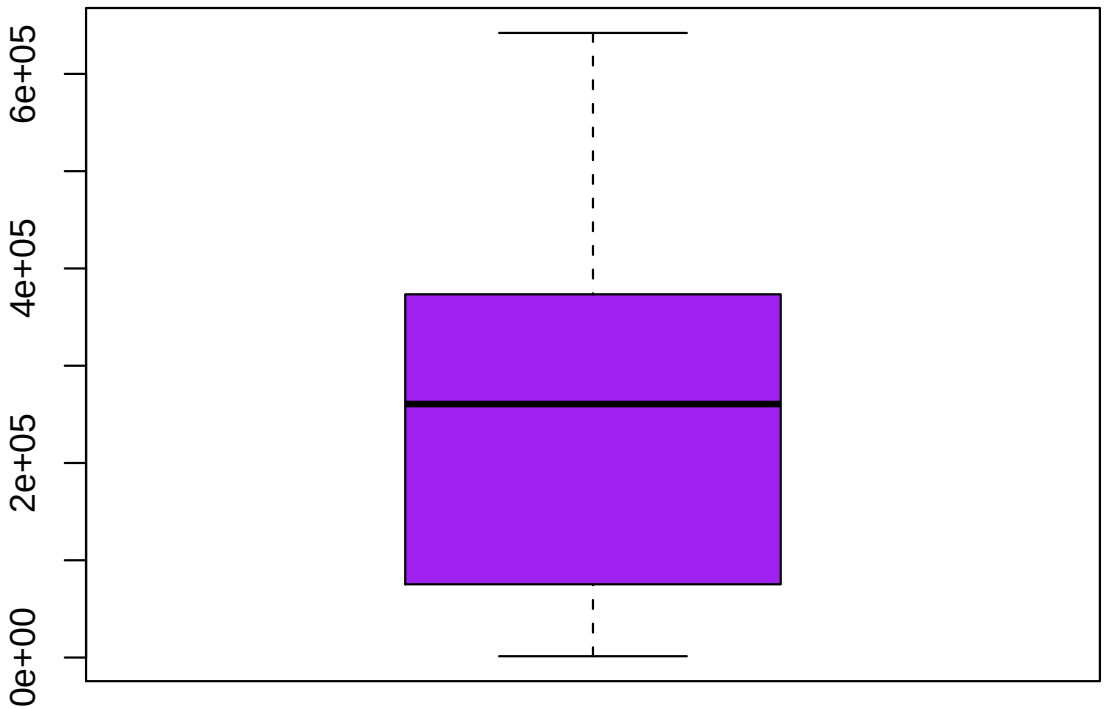


Boxplot

Otra imagen muy útil es un diagrama de caja y brazos.
La creamos con la función boxplot:

```
> boxplot(phage$V2, col = "purple")
```

Boxplot



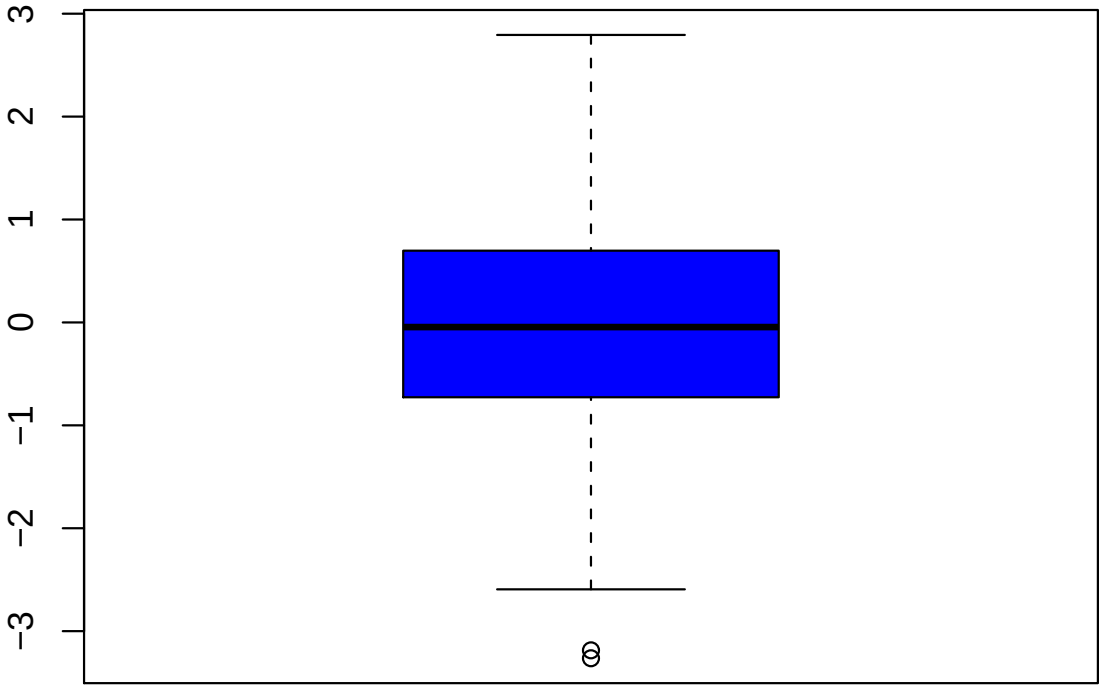
Entendiendo el boxplot

- ¿Dónde esta la mediana?
- ¿Dónde esta el primer cuartil?
- ¿Dónde esta el tercer cuartil?
- ¿Cúal es el rango intercuantílico?
- ¿Hay puntos extremos?

Otro boxplot

```
> boxplot(rnorm(1000), col = "blue")
```

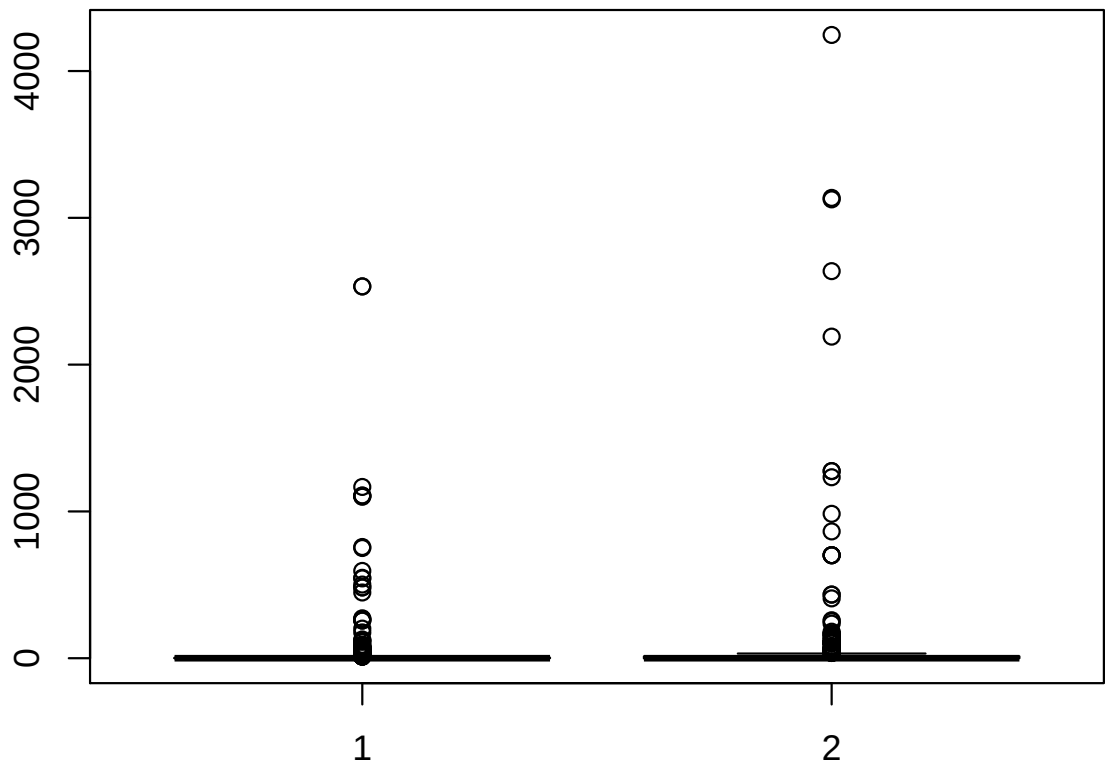
Otro boxplot



Con dos variables o más

```
> boxplot(mirnas$AGO4, mirnas$AGO5,
+         col = c("blue", "green"))
```

Con dos variables o más

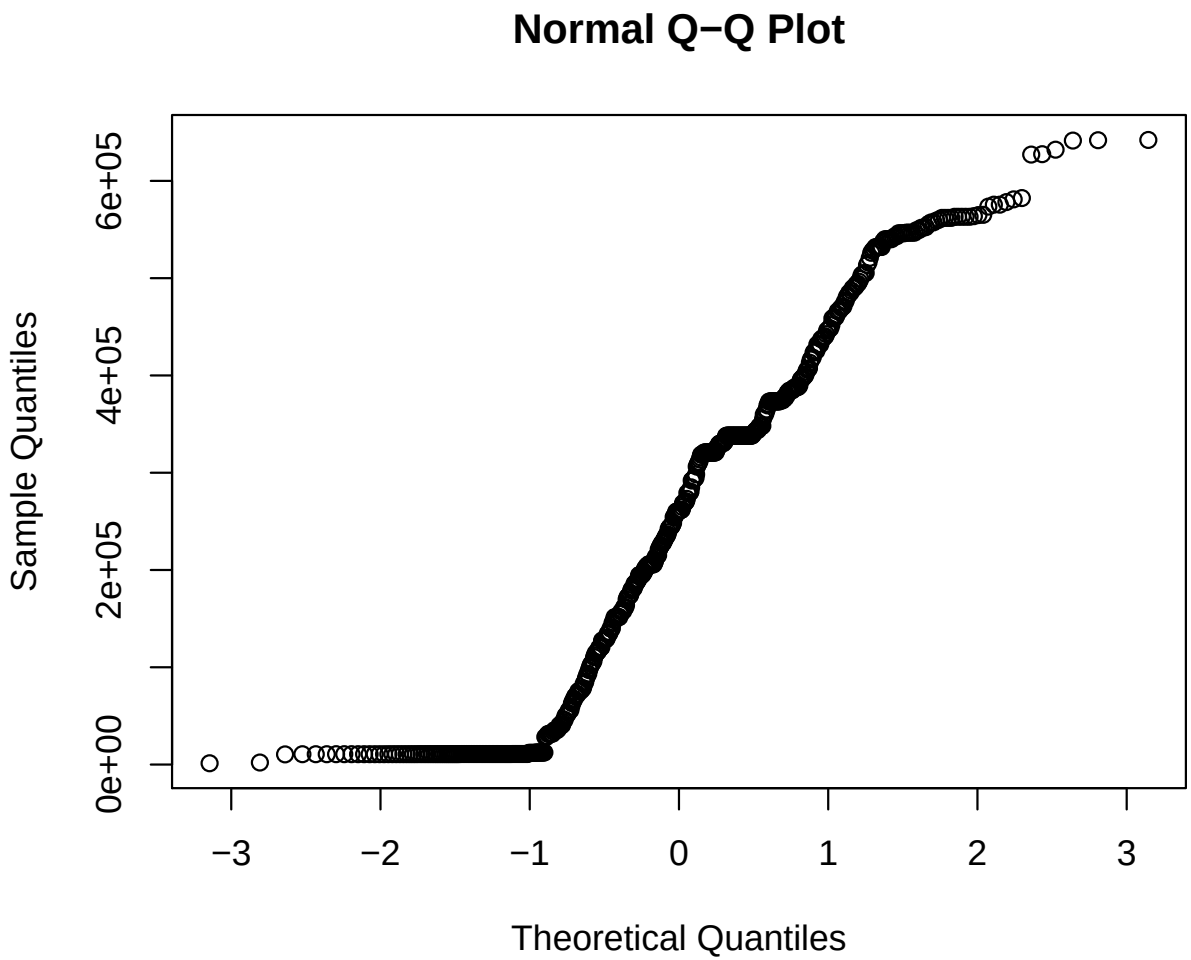


QQnorm

Otra función bastante útil es la qqnorm.
Con esta comparamos los cuantiles de la distribución de nuestros datos vs los cuantiles de la distribución normal teórica.
Si obtenemos una diagonal, vamos de gane :)

```
> qqnorm(phage$V2)
```

QQnorm

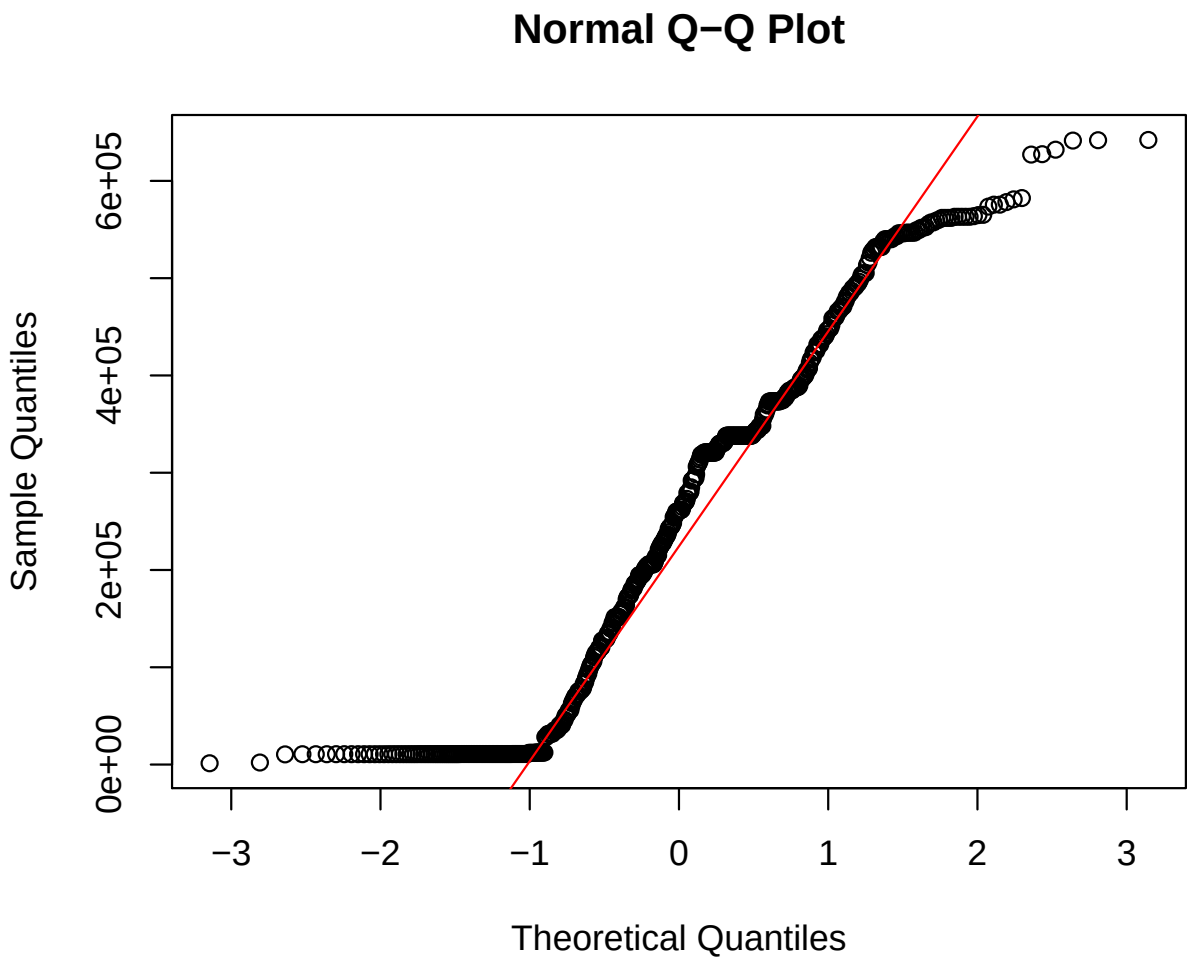

Normal Q−Q Plot
Sample Quantiles
0e+00
2e+05
4e+05
6e+05
−3
−2
−1
0
1
2
3
Theoretical Quantiles

title

La función qqline es bastante útil. Esta nos grafica una línea entre el primer y tercer cuartil:

```
> qqnorm(phage$V2)
> qqline(phage$V2, col = "red")
```

title

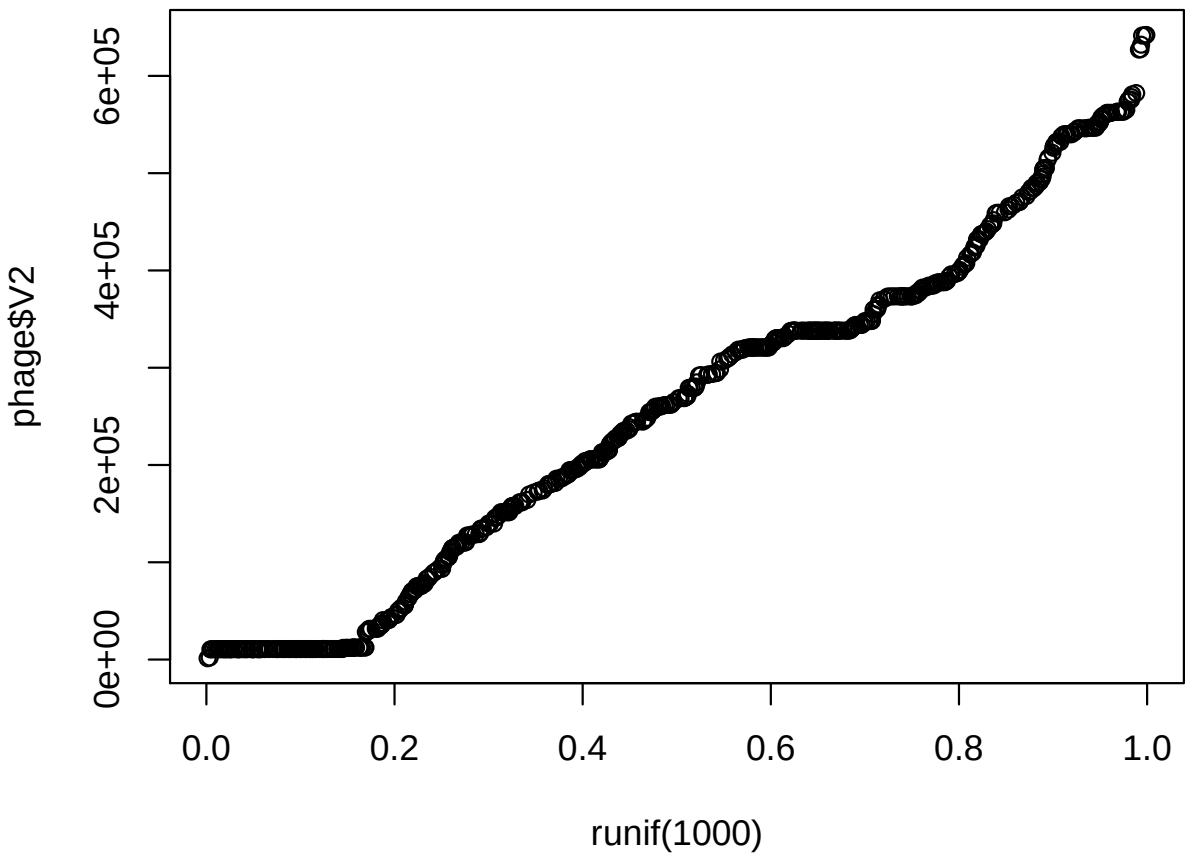

Normal Q−Q Plot
Sample Quantiles
0e+00
2e+05
4e+05
6e+05
−3
−2
−1
0
1
2
3
Theoretical Quantiles

QQplot

Si queremos comparar los cuantiles entre dos muestras o entre una muestra y una distribución teórica que no sea la normal, usamos qqplot:

```
> qqplot(runif(1000), phage$V2)
```

QQplot

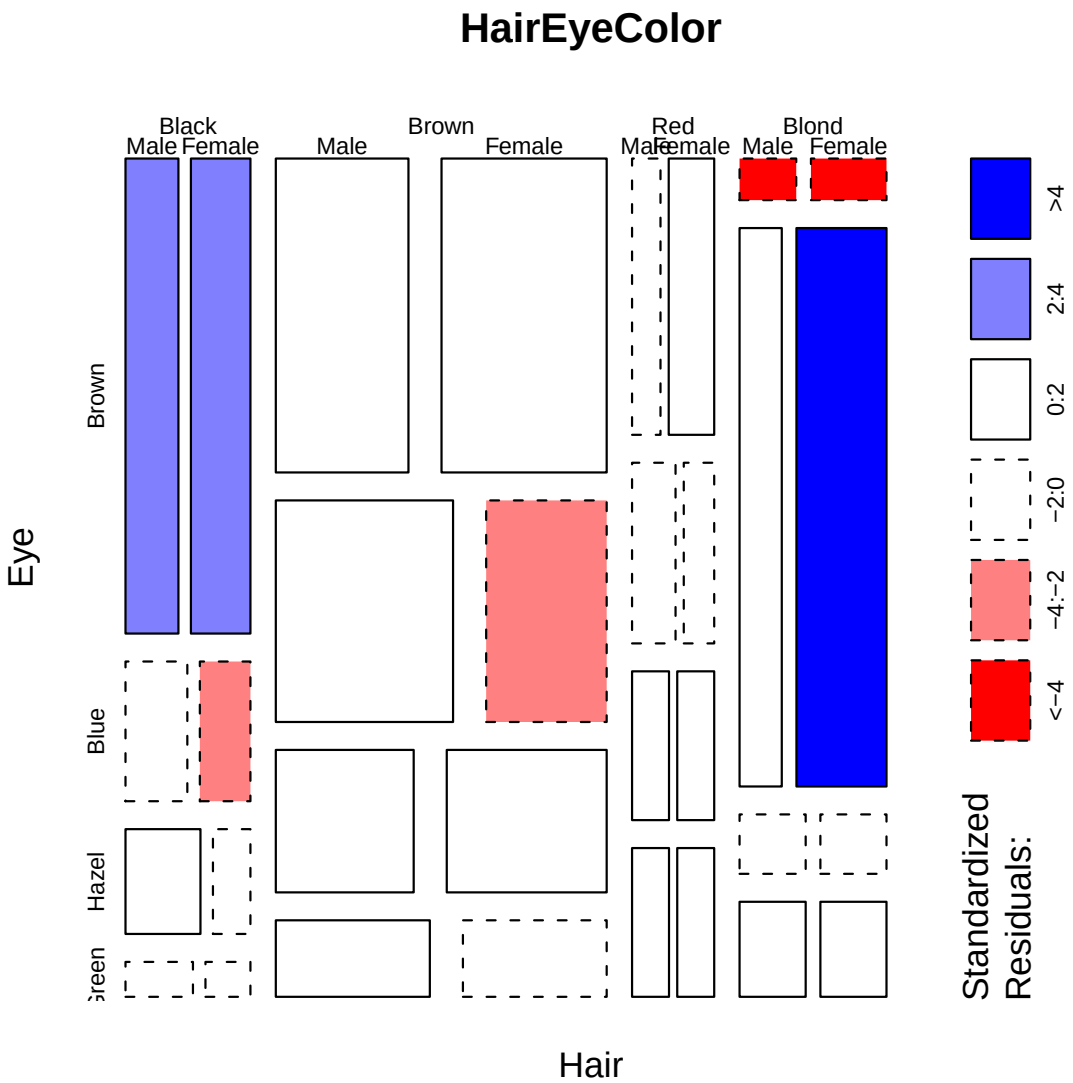


Mosaicplot

Una de las gráficas más interesantes es mosaicplot.
Veamos un ejemplo:

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

Mosaicplot


HairEyeColor
Black
Male Female
Brown
Male
Female
Red
Male Female
Blond
Male Female
Brown
Blue
Hazel
Green
Eye
Hair
>4
2:4
0:2
−2:0
−4:−2
<−4
Standardized Residuals:

Ejercicios

- Usando el objeto `mirnas`, guarden en un objeto[1] los valores de la columna Total de las líneas impares.
- Usando ese objeto, hagan un histograma con la línea de densidad. Póngale un título y nombres a los ejes.
- Con el mismo objeto, hagan un boxplot. Ponganle un título y nombres a los ejes.
  ¿Qué pueden concluir?
- Exploren gráficamente si esos datos se distribuyen como normal usando . . .

[1] Si no quieren está bien, es solo para evitar escribir más código.

Pronto veremos

- Más gráficas con 3 o más variables
- Fin!

sessionInfo

Información de mi sesión:

```
> sessionInfo()

R version 2.10.0 (2009-10-26)
i386-pc-mingw32

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats     graphics  grDevices
[4] utils     datasets  methods
[7] base
```