Freddy Hernández Barajas
Juan Carlos Correa Morales

Gráficos en R usando la solo la base

Gracias a Dios por todo lo que me ha dado!

Índice general

Índice de cuadros

Índice de figuras

Prefacio

Este libro fue creado con la intención de ayudar a los estudiantes de pregrado, especialización, maestría e investigadores a crear gráficos estadísticos con las herramientas básicas de R. Como complemento, recomendamos a los lectores el libro Hernández (2018) que muestra cómo usar R para realizar diversos procedimiento estadísticos.

Estructura del libro

En el capítulo 1 se presenta una introducción breve de R, sus orígenes, la instalación, tipos de objetos y una guía de estilo para escribir en R.

Software information and conventions

Para realizar este libro usamos los paquetes **knitr** (Xie, 2015) y **bookdown** (Xie, 2016) que permiten unir la ventajas de LaTeXy R en un mismo archivo.

En todo el libro se presentarán códigos que el lector puede copiar y pegar en su consola de R para obtener los mismos resultados aquí presentados. Los códigos se destacan en una caja de color beis (o beige) similar a la mostrada a continuación.

```
4 + 6
a <- c(1, 5, 6)
5 * a
1:10
```

Los resultados o salidas obtenidos de cualquier código se destacan con dos símbolos de númeral (**##**) al inicio de cada línea o renglón, esto quiere decir

que todo lo que inicie con `##` son resultados obtenidos y **NO** los debe copiar. Abajo se muestran los resultados obtenidos luego de correr el código anterior.

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

```
## [1]  5 25 30
```

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

Bloques informativos

En varias partes del libro usaremos bloques informativos para resaltar algún aspecto importante. Abajo se encuentra un ejemplo de los bloques y su significado.

Este bloque sirve para una nota aclaratoria.

Este bloque sirve para una sugerencia.

Este bloque sirve para una advertencia.

Agradecimientos

Agradecemos a nuestros estudiantes, profesores y colegas que han leído el manuscrito y se han tomado el trabajo de escribirnos dándonos sus sugerencias y comentarios para mejorar continuamente este material.

Freddy Hernández Barajas

Juan Carlos Correa Morales

Sobre los autores

Juan Carlos Correa Morales es profesor asociado de la Universidad Nacional de Colombia adscrito a la Escuela de Estadística de la Facultad de Ciencias.

jccorreamorales@gmail.com[1]

Freddy Hernández Barajas es profesor asistente de la Universidad Nacional de Colombia adscrito a la Escuela de Estadística de la Facultad de Ciencias.

fhernanb@unal.edu.co[2]

[1] mailto:jccorreamorales@gmail.com
[2] mailto:fhernanb@unal.edu.co

1

Introducción

1.1. Orígenes

R es un lenguaje de programación usado para realizar procedimientos estadísticos y gráficos de alto nivel, este lenguaje fue creado en 1993 por los profesores e investigadores Robert Gentleman y Ross Ihaka. Inicialmente el lenguaje se usó para apoyar los cursos que tenían a su cargo los profesores, pero luego de ver la utilidad de la herramienta desarrollada, decidieron colocar copias de R en StatLib. A partir de 1995 el código fuente de R está disponible bajo licencia GNU GPL para sistemas operativos Windows, Macintosh y distribuciones Unix/Linux. La comunidad de usuarios de R en el mundo es muy grande y los usuarios cuentan con diferentes espacios para interactuar, a continuación una lista no exhaustiva de los sitios más populares relacionados con R:

- Rbloggers[1].
- Comunidad hispana de R[2].
- Nabble[3].
- Foro en portugués[4].
- Stackoverflow[5].
- Cross Validated[6].
- R-Help Mailing List[7].
- Revolutions[8].
- R-statistics blog[9].
- RDataMining[10].

[1] https://www.r-bloggers.com/
[2] http://r-es.org/
[3] http://r.789695.n4.nabble.com/
[4] http://r-br.2285057.n4.nabble.com/
[5] http://stackoverflow.com/questions/tagged/r
[6] http://stats.stackexchange.com/questions/tagged/r
[7] https://stat.ethz.ch/mailman/listinfo/r-help
[8] http://blog.revolutionanalytics.com/
[9] https://www.r-statistics.com/
[10] https://rdatamining.wordpress.com/

Figura 1.1: Robert Gentleman (izquierda) y Ross Ihaka (derecha) creadores de R.

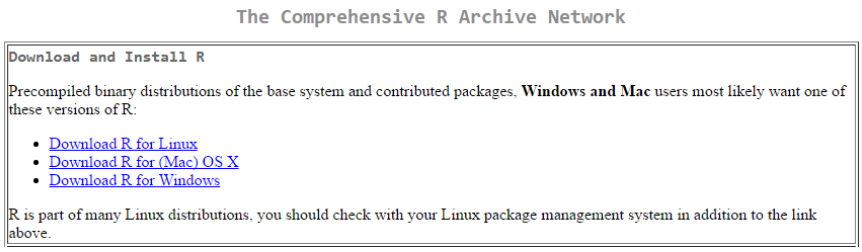

Figura 1.2: Página del Cran.

1.2. Descarga e instalación

Para realizar la instalación de R usted debe visitar la página del CRAN (*Comprehensive R Archive Network*) disponible en este enlace[11]. Una vez ingrese a la página encontrará un cuadro similar al mostrado en la Figura 1.2 donde aparecen los enlaces de la instalación para los sistemas operativos Linux, Mac y Windows.

Supongamos que se desea instalar R en Windows, para esto se debe dar clic sobre el hiperenlace Download R for Windows de la Figura 1.2. Una vez hecho esto se abrirá una página con el contenido mostrado en la Figura 1.3. Una vez ingrese a esa nueva página usted debe dar clic sobre el hiperenlace install R for the first time como es señalado por la flecha roja en la Figura 1.3.

Luego de esto se abrirá otra página con un encabezado similar al mostrado en la Figura 1.4, al momento de capturar la figura la versión actual de R era 3.2.5 pero seguramente en este momento usted tendrá disponible una versión actua-

[11] https://cran.r-project.org/

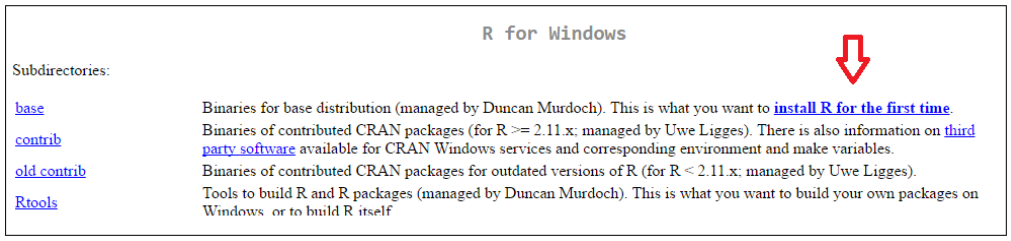

Figura 1.3: Página de instalación para la primera ocasión.

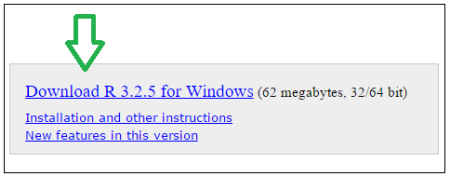

Figura 1.4: Página de descarga.

lizada. Una vez allí uste debe dar clic sobre Download R 3.2.5 for Windows como es señalado por la flecha verde. Luego de esto se descargará el instalador R en el computador el cual deberá ser instalado con las opciones que vienen por defecto.

Se recomienda observar el siguiente video didáctico de instalación de R disponible en este enlace[12] para facilitar la tarea de instalación.

1.3. Apariencia del programa

Una vez que esté instalado R en su computador, usted podrá acceder a él por la lista de programas o por medio del acceso directo que quedó en el escritorio, en la Figura 1.5 se muestra la apariencia del acceso directo para ingresar a R.

Al abrir R aparecerá en la pantalla de su computador algo similar a lo que está en la Figura 1.6. La ventana izquierda se llama consola y es donde se ingresan las instrucciones, una vez que se construye un gráfico se activa otra ventana llamada ventana gráfica. Cualquier usuario puede modificar la posición y tamaños de estas ventanas, puede cambiar el tipo y tamaño de las letras en la consola, para hacer esto se deben explorar las opciones de *editar* en la barra de herramientas.

[12] http://tinyurl.com/jd7b9ks

Figura 1.5: Apariencia del acceso directo para ingresar a R.

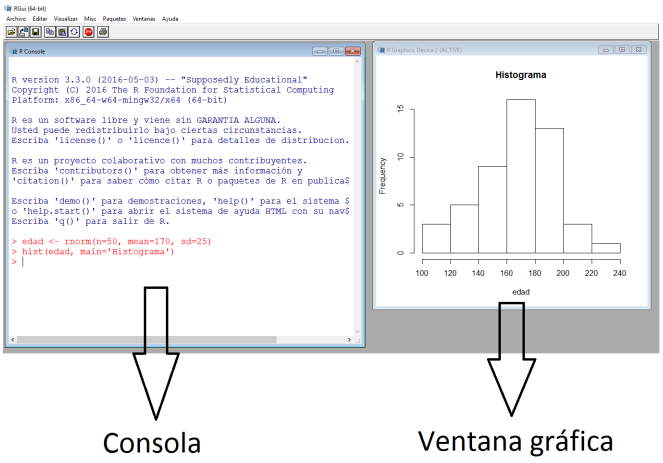

Figura 1.6: Apariencia de R.

1.4. Tipos de objetos

En R existen varios tipos de objectos que permiten que el usuario pueda almacenar la información para realizar procedimientos estadísticos y gráficos. Los principales objetos en R son vectores, matrices, arreglos, marcos de datos y listas. A continuación se presentan las características de estos objetos y la forma para crearlos.

1.4.1. Vectores

Los vectores son arreglos ordenados en los cuales se puede almacenar información de tipo numérico (variable cuantitativa), alfanumérico (variable cua-

litativa) o lógico (`TRUE` o `FALSE`), pero no mezclas de éstos. La función de R para crear un vector es `c()` y que significa concatenar; dentro de los paréntesis de esta función se ubica la información a almacenar. Una vez construído el vector se acostumbra a etiquetarlo con un nombre corto y representativo de la información que almacena, la asignación se hace por medio del operador `<-` entre el nombre y el vector.

A continuación se presenta un ejemplo de cómo crear tres vectores que contienen las respuestas de cinco personas a tres preguntas que se les realizaron.

```
edad <- c(15, 19, 13, NA, 20)
deporte <- c(TRUE, TRUE, NA, FALSE, TRUE)
comic.fav <- c(NA, 'Superman', 'Batman', NA, 'Batman')
```

El vector `edad` es un vector cuantitativo y contiene las edades de las 5 personas. En la cuarta posición del vector se colocó el símbolo `NA` que significa *Not Available* debido a que no se registró la edad para esa persona. Al hacer una asignación se acostumbra a dejar un espacio antes y después del operador `<-` de asignación. El segundo vector es llamado `deporte` y es un vector lógico que almacena las respuestas a la pregunta de si la persona practica deporte, nuevamente aquí hay un `NA` para la tercera persona. El último vector `comic.fav` contiene la información del cómic favorito de cada persona, como esta variable es cualitativa es necesario usar las comillas `' '` para encerrar las respuestas.

Cuando se usa `NA` para representar una información *Not Available* no se deben usar comillas.

Es posible usar comillas sencillas `'foo'` o comillas dobles `"foo"` para ingresar valores de una variable cualitativa.

Si se desea ver lo que está almacenado en cada uno de estos vectores, se debe escribir en la consola de R el nombre de uno de los objetos y luego se presiona la tecla *enter* o *intro*, al realizar esto lo que se obtiene se muestra a continuación.

```
edad
```

```
## [1] 15 19 13 NA 20
```

```
deporte
```

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

```
comic.fav
```

```
## [1] NA         "Superman" "Batman"   NA
## [5] "Batman"
```

1.4.2. Matrices

Las matrices son arreglos rectangulares de filas y columnas con información numérica, alfanumérica o lógica. Para construir una matriz se usa la función `matrix( )`. Por ejemplo, para crear una matriz de 4 filas y 5 columnas (de dimensión 4×5) con los primeros 20 números positivos se escribe el código siguiente en la consola.

```
mimatriz <- matrix(data=1:20, nrow=4, ncol=5, byrow=FALSE)
```

El argumento `data` de la función sirve para indicar los datos que se van a almacenar en la matriz, los argumentos `nrow` y `ncol` sirven para definir la dimensión de la matriz y por último el argumento `byrow` sirve para indicar si la información contenida en `data` se debe ingresar por filas o no. Para observar lo que quedó almacenado en el objeto `mimatriz` se escribe en la consola el nombre del objeto seguido de la tecla *enter* o *intro*.

```
mimatriz
```

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

1.4.3. Arreglos

Un arreglo es una matriz de varias dimensiones con información numérica, alfanumérica o lógica. Para construir una arreglo se usa la función `array( )`. Por ejemplo, para crear un arreglo de $3 \times 4 \times 2$ con las primeras 24 letras minúsculas del alfabeto se escribe el siguiente código.

```
miarray <- array(data=letters[1:24], dim=c(3, 4, 2))
```

El argumento `data` de la función sirve para indicar los datos que se van a almacenar en el arreglo y el argumento `dim` sirve para indicar las dimensiones

del arreglo. Para observar lo que quedó almacenado en el objeto `miarray` se escribe en la consola lo siguiente.

```
miarray
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,] "a"  "d"  "g"  "j"
## [2,] "b"  "e"  "h"  "k"
## [3,] "c"  "f"  "i"  "l"
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,] "m"  "p"  "s"  "v"
## [2,] "n"  "q"  "t"  "w"
## [3,] "o"  "r"  "u"  "x"
```

1.4.4. Marco de datos

El marco de datos o *data frame* es uno de los objetos más utilizados porque permite agrupar vectores con información de diferente tipo (numérica, alfanumérica o lógica) en un mismo objeto, la única restricción es que los vectores deben tener la misma longitud. Para crear un marco de datos se usa la función `data.frame( )`, como ejemplo vamos a crear un marco de datos con los vectores `edad`, `deporte` y `comic.fav` definidos anteriormente.

```
mimarco <- data.frame(edad, deporte, comic.fav)
```

Una vez creado el objeto `mimarco` podemos ver el objeto escribiendo su nombre en la consola, a continuación se muestra lo que se obtiene.

```
mimarco
```

```
##   edad deporte comic.fav
## 1   15    TRUE      <NA>
## 2   19    TRUE  Superman
## 3   13      NA    Batman
## 4   NA   FALSE      <NA>
## 5   20    TRUE    Batman
```

De la salida anterior vemos que el marco de datos tiene 3 variables (columnas)

cuyos nombres coinciden con los nombres de los vectores creados anteriormente, los números consecutivos al lado izquierdo son sólo de referencia y permiten identificar la información para cada persona en la base de datos.

1.4.5. Listas

Las listas son otro tipo de objeto muy usado para almacenar objetos de diferente tipo. La instrucción para crear una lista es `list( )`. A continuación vamos a crear una lista que contiene tres objetos: un vector con 5 números aleatorios llamado `mivector`, una matriz de dimensión 6×2 con los primeros doce números enteros positivos llamada `matriz2` y el tercer objeto será el marco de datos `mimarco` creado en el apartado anterior. Las instrucciones para crear la lista requerida se muestran a continuación.

```
set.seed(12345)
mivector <- runif(n=5)
matriz2 <- matrix(data=1:12, ncol=6)
milista <- list(E1=mivector, E2=matriz2, E3=mimarco)
```

La función `set.seed` de la línea número 1 sirve para fijar la semilla de tal manera que los números aleatorios generados en la segunda línea con la función `runif` sean siempre los mismos. En la última línea del código anterior se construye la lista, dentro de la función `list` se colocan los tres objetos `mivector`, `matriz2` y `mimarco`. Es posible colocarle un nombre especial a cada uno de los elementos de la lista, en este ejemplo se colocaron los nombres `E1`, `E2` y `E3` para cada uno de los tres elementos. Para observar lo que quedó almacenado en la lista se escribe `milista` en la consola y el resultado se muestra a continuación.

```
milista
```

```
## $E1
## [1] 0.7209 0.8758 0.7610 0.8861 0.4565
##
## $E2
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
##
## $E3
##   edad deporte comic.fav
## 1   15    TRUE      <NA>
## 2   19    TRUE  Superman
```

```
## 3   13      NA    Batman
## 4   NA   FALSE      <NA>
## 5   20    TRUE    Batman
```

1.5. Guía de estilo para la escritura en R

Así como en el español existen reglas ortográficas, la escritura de códigos en R también tiene unas reglas que se recomienda seguir para evitar confusiones. Tener una buena guía de estilo es importante para que el código creado por usted sea fácilmente entendido por sus lectores Wickham (2015). No existe una única y mejor guía de estilo para escritura en R, sin embargo aquí vamos a mostrar unas sugerencias basadas en la guía llamada *Google's R style guide*[13].

1.5.1. Nombres de los archivos

Se sugiere que el nombre usado para nombrar un archivo tenga sentido y que termine con extensión .R. A continuación dos ejemplos de como nombrar mal y bien un archivo.

- Mal: `hola.R`
- Bien: `analisis_icfes.R`

1.5.2. Nombres de los objetos

Se recomienda no usar los símbolos `_` y `-` dentro de los nombres de objetos. Para las variables es preferible usar letras minúsculas y separar las palabras con puntos (`peso.maiz`) o utilizar la notación camello iniciando en minúscula (`pesoMaiz`). Para las funciones se recomienda usar la notación camello iniciando todas la palabras en mayúscula (`PlotRes`). Para los nombres de las constantes se recomienda que inicien con la letra k (`kPrecioBus`). A continuación ejemplos de buenas y malas prácticas.

Para variables:

- Bien: `avg.clicks`
- Aceptable: `avgClicks`
- Mal: `avg_Clicks`

[13] https://google.github.io/styleguide/Rguide.xml

Para funciones:

- Bien: `CalculateAvgClicks`
- Mal: `calculate_avg_clicks` , `calculateAvgClicks`

1.5.3. Longitud de una línea de código

Se recomienda que cada línea tenga como máximo 80 caracteres. Si una línea es muy larga se debe cortar siempre por una coma.

1.5.4. Espacios

Use espacios alrededor de todos los operadores binarios (=, +, -, <-, etc.). Los espacios alrededor del símbolo "=" son opcionales cuando se usan para ingresar valores dentro de una función. Así como en español, nunca coloque espacio antes de una coma, pero siempre use espacio luego de una coma. A continuación ejemplos de buenas y malas prácticas.

```
tab <- table(df[df$days < 0, 2])  # Bien
tot <- sum(x[, 1])                # Bien
tot <- sum(x[1, ])                # Bien
tab <- table(df[df$days<0, 2])    # Faltan espacios alrededor '<'
tab <- table(df[df$days < 0,2])   # Falta espacio luego de coma
tab <- table(df[df$days < 0 , 2]) # Sobra espacio antes de coma
tab<- table(df[df$days < 0, 2])   # Falta espacio antes de '<-'
tab<-table(df[df$days < 0, 2])    # Falta espacio alrededor de '<-'
tot <- sum(x[,1])                 # Falta espacio luego de coma
tot <- sum(x[1,])                 # Falta espacio luego de coma
```

Otra buena práctica es colocar espacio antes de un paréntesis excepto cuando se llama una función.

```
if (debug)    # Correcto
if(debug)     # Funciona pero no se recomienda
colMeans (x)  # Funciona pero no se recomienda
```

Espacios extras pueden ser usados si con esto se mejora la apariencia del código, ver el ejemplo siguiente.

```
plot(x    = x.coord,
     y    = data.mat[, MakeColName(metric, ptiles[1], "roiOpt")],
```

```
     ylim = ylim,
     xlab = "dates",
     ylab = metric,
     main = (paste(metric, " for 3 samples ", sep = "")))
```

No coloque espacios alrededor del código que esté dentro de paréntesis `( )` o corchetes `[ ]`, la única excepción es luego de una coma, ver el ejemplo siguiente.

```
if (condicion)    # Correcto
x[1, ]            # Correcto
if ( condicion )  # Sobran espacios alrededor de condicion
x[1,]             # Se necesita espacio luego de coma
```

Los signos de agrupación llaves `{ }` se utilizan para agrupar bloques de código y se recomienda que nunca una llave abierta `{` esté sola en una línea; una llave cerrada `}` si debe ir sola en su propia línea. Se pueden omitir las llaves cuando el bloque de instrucciones esté formado por una sola línea pero esa línea de código NO debe ir en la misma línea de la condición. A continuación dos ejemplos de lo que se recomienda.

```
if (is.null(ylim)) {                    # Correcto
  ylim <- c(0, 0.06)
}

if (is.null(ylim))                      # Correcto
  ylim <- c(0, 0.06)

if (is.null(ylim)) ylim <- c(0, 0.06)   # Aceptable

if (is.null(ylim))                      # No se recomienda
{
  ylim <- c(0, 0.06)
}

if (is.null(ylim)) {ylim <- c(0, 0.06)}
# Frente a la llave { no debe ir nada
# la llave de cierre } debe ir sola
```

La sentencia else debe ir siempre entre llaves `} {`, ver el siguiente ejemplo.

```
if (condition) {
  one or more lines
} else {                  # Correcto
  one or more lines
}


if (condition) {
  one or more lines
}
else {                    # Incorrecto
  one or more lines
}


if (condition)
  one line
else                      # Incorrecto
  one line
```

1.5.5. Asignación

Para realizar asignaciones se recomienda usar el símbolo `<-`, el símbolo de igualdad `=` no se recomienda usarlo para asignaciones.

```
x <- 5  # Correcto
x = 5   # No recomendado
```

Para una explicación más detallada sobre el símbolo de asignación se recomienda visitar este enlace[14].

1.5.6. Punto y coma

No se recomienda colocar varias instrucciones separadas por `;` en la misma línea, aunque funciona dificulta la revisión del código.

```
n <- 100; y <- rnorm(n, mean=5); hist(y)  # No se recomienda
```

[14] http://www.win-vector.com/blog/2016/12/the-case-for-using-in-r/

```
n <- 100                                    # Correcto
y <- rnorm(n, mean=5)
hist(y)
```

A pesar de la anterior advertencia es posible que en este libro usemos el ; en algunas ocasiones, si lo hacemos es para ahorrar espacio en la presentación del código.

2

Gráficos para una variable cuantitativa

En este capítulo se presentan funciones para la creación de gráficos con una sola variable cuantitativa.

2.1. Función `stem`

Nos permite obtener el gráfico llamado de tallo y hoja debido a su apariencia. Este gráfico fue propuesto por Tukey (1977) y a pesar de no ser un gráfico para presentación definitiva se utiliza a la vez que el analista recoge la información para ver rápidamente la distribución de los datos.

¿Qué muestra este gráfico?

1. El centro de la distribución.
2. La forma general de la distribución:
   - Simétrica: Si las porciones a cada lado del centro son imágenes espejos de las otras.
   - Sesgada a la izquierda: Si la cola izquierda (los valores menores) es mucho más larga que los de la derecha (los valores mayores).
   - Sesgada a la derecha: Opuesto a la sesgada a la izquierda.
3. Desviaciones marcadas de la forma global de la distribución.
   - Outliers: Observaciones individuales que caen muy por fuera del patrón general de los datos.
   - Gaps: Huecos en la distribución

Ventajas del gráfico:

1. Muy fácil de realizar y puede hacerse a mano.
2. Fácil de entender.

Desventajas del gráfico:

1. El gráfico es tosco y no sirve para presentaciones definitivas.

2. Funciona cuando el número de observaciones no es muy grande.
3. No permite comparar claramente diferentes poblaciones

Ejemplo

Como ilustración vamos a crear el gráfico de tallo y hoja para la variable altura (cm) de un grupo de estudiantes de la universidad. Primero se leerán los datos disponibles en github y luego se usará la función `stem` para obtener el gráfico. A continuación el código usado.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
datos <- read.table(file=url, header=T)
stem(datos$altura)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   14 | 7
##   15 | 3
##   15 | 679
##   16 | 0001
##   16 | 68888
##   17 | 001334
##   17 | 5678899
##   18 | 000033
##   18 | 88
##   19 | 1
```

De este gráfico sencillo se puede ver que la variable presenta una mayor frecuencia para alturas ente 170 y 179 cm y que no tiene una distribución simétrica.

2.2. Función `boxplot`

La función `boxplot` sirve para crear un diagrama de cajas y bigote para una variable cuantitativa. La estructura de la función `boxplot` con los argumentos más comunes de uso se muestran a continuación.

```
boxplot(x, formula, data, subset, na.action,
        range, width, varwidth, notch, names,
        plot, col, log, horizontal, add, ...)
```

Los argumentos de la función `boxplot` son:

- `x`: vector numérico con los datos para crear el boxplot.
- `formula`: fórmula con la estructura `x ~ g` para indicar que las observaciones en el vector `x` van a ser agrupadas de acuerdo a los niveles del factor `g`.
- `data`: marco de datos con las variables.
- `subset`: un vector opcional para especificar un subconjunto de observaciones a ser usadas en el proceso de ajuste.
- `na.action`: una función la cual indica lo que debería pasar cuando los datos contienen "NA's".
- `range`: valor numérico que indica la extensión de los bigotes. Si es positivo, los bigotes se extenderán hasta el punto más extremo de tal manera que el bigote no supere `range` veces el rango intercuatílico (IQ). Un valor de cero hace que los bigotes se extiendan hasta los datos extremos.
- `width`: un vector con los anchos relativos de las cajas.
- `varwidth`: Si es `TRUE`, las cajas son dibujadas con anchos proporcionales a las raíces cuadradas del número de observaciones en los grupos.
- `notch`: si es `TRUE`, una cuña es dibujada a cada lado de las cajas. Cuando las cuñas de dos gráficos de caja no se traslapan, entonces las medianas son significativamente diferentes a un nivel del 5 %.
- `names`: un con las etiquetas a ser impresas debajo de cada boxplot.
- `plot`: si es `TRUE` (por defecto) entonces se produce el gráfico, de lo contrario, se producen los resúmenes de los boxplots.
- `col`: vector con los colores a usar en el cuerpo de las cajas.
- `log`: para indicar si las coordenadas `x` o `y` o serán graficadas en escala logarítmica.
- `...`: otros parámetros gráficos que pueden ser pasados como argumentos para el boxplot.

Ejemplo

Como ilustración vamos a crear tres boxplot para la variable altura (cm) de un grupo de estudiantes de la universidad, el primer boxplot será sobre la variable altura, el segundo será un boxplot para altura diferenciando por sexo y el tercer boxplot será igual que el primero pero modificando los nombres a imprimir en el eje horizontal. Primero se leerán los datos disponibles en github y luego se usará la función `boxplot` para obtener ambos gráfico. A continuación el código usado.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
datos <- read.table(file=url, header=T)

par(mfrow=c(1, 3))
boxplot(x=datos$altura, ylab='Altura (cm)')

boxplot(altura ~ sexo, data=datos,
        xlab='Sexo', ylab='Altura (cm)')

boxplot(altura ~ sexo, data=datos, horizontal=TRUE,
        ylab='Género', xlab='Altura (cm)',
        names=c('Masculino', 'Femenino'))
```

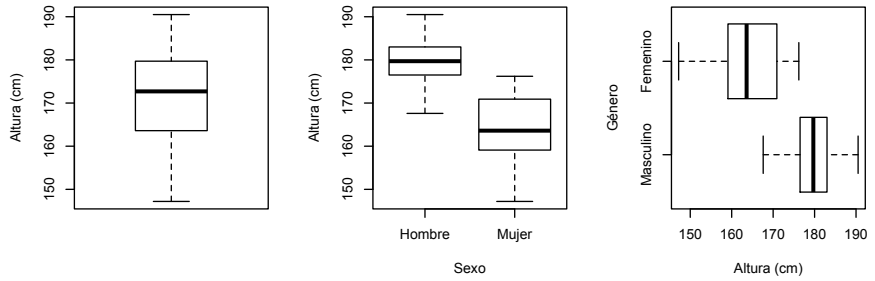

Figura 2.1: Boxplot para la variable altura.

En la Figura 2.1 se presentan los boxplots obtenidos con las instrucciones anteriores. El segundo y tercer boxplot son el mismo, lo único que se modificó fueron los nombres o etiquetas a colocar debajo de cada boxplot por medio del argumento `names` y la orientación.

2.3. Función `hist`

La función `hist` sirve para crear el histograma para una variable cuantitativa. Como argumentos esta función recibe un vector con los datos y opcionalmente puede ingresarse como argumento el número de intervalos a ser graficados o en su defecto el número de clases se determina con la fórmula de Sturges.

Nota: los programas de computador usualmente construyen los histogramas

automáticamente, sin embargo, un buen programa debe permitirnos elegir el número de intervalos del histograma. Si usted posee un programa que no le permite hacer cambios, cambie de programa.

La estructura de la función `hist` con los argumentos más comunes de uso se muestran a continuación.

- `x`: vector numérico de valores para construir el histograma.
- `breaks`: puede ser un número entero que indica el número aproximado de clases o un vector cuyos elementos indican los límites de los intervalos.
- `freq`: argumento lógico; si se especifica como `TRUE`, el histograma presentará frecuencias absolutas o conteo de datos para cada intervalo; si se especifica como `FALSE` el histograma presentar las frecuencias relativas (en porcentaje). Por defecto, este argumento toma el valor de `TRUE` siempre y cuando los intervalos sean de igual ancho.
- `include.lowest`: argumento lógico; si se especifica como `TRUE`, un `x[i]` igual a los equal a un valor `breaks` se incluirá en la primera barra, si el argumento `right = TRUE`, o en la última en caso contrario.
- `right`: argumento lógico; si es `TRUE`, los intervalos son abiertos a la izquierda y cerrados a la derecha $(a, b]$. Para la primera clase o intervalo si `include.lowest=TRUE` el valor más pequeño de los datos será incluido en éste. Si es `FALSE` los intervalos serán de la forma $[a, b)$ y el argumento `include.lowest=TRUE` tendrá el significado de incluir el "más alto".
- `col`: para definir el color de las barras. Por defecto, `NULL` produce barras sin fondo.
- `border`: para definir el color de los bordes de las barras.
- `plot`: argumento lógico. Por defecto es `TRUE`, y el resultado es el gráfico del histograma; si se especifica como `FALSE` el resultado es una lista de conteos por cada intervalo.
- `labels`: argumento lógico o carácter. Si se especifica como `TRUE` coloca etiquetas arriba de cada barra.
- `...`: parámetros gráficos adicionales a `title` y `axis`.

Ejemplo

Vamos a construir varios histogramas para los tiempos de 180 corredores de la media maratón de CONAVI realizada hace algunos años. A continuación se muestra la forma de ingresar los 180 datos.

```
maraton <- c(
10253, 10302, 10307, 10309, 10349, 10353, 10409, 10442, 10447,
10452, 10504, 10517, 10530, 10540, 10549, 10549, 10606, 10612,
10646, 10648, 10655, 10707, 10726, 10731, 10737, 10743, 10808,
10833, 10843, 10920, 10938, 10949, 10954, 10956, 10958, 11004,
```

```
11009, 11024, 11037, 11045, 11046, 11049, 11104, 11127, 11205,
11207, 11215, 11226, 11233, 11239, 11307, 11330, 11342, 11351,
11405, 11413, 11438, 11453, 11500, 11501, 11502, 11503, 11527,
11544, 11549, 11559, 11612, 11617, 11635, 11655, 11731, 11735,
11746, 11800, 11814, 11828, 11832, 11841, 11909, 11926, 11937,
11940, 11947, 11952, 12005, 12044, 12113, 12209, 12230, 12258,
12309, 12327, 12341, 12413, 12433, 12440, 12447, 12530, 12600,
12617, 12640, 12700, 12706, 12727, 12840, 12851, 12851, 12937,
13019, 13040, 13110, 13114, 13122, 13155, 13205, 13210, 13220,
13228, 13307, 13316, 13335, 13420, 13425, 13435, 13435, 13448,
13456, 13536, 13608, 13612, 13620, 13646, 13705, 13730, 13730,
13730, 13747, 13810, 13850, 13854, 13901, 13905, 13907, 13912,
13920, 14000, 14010, 14025, 14152, 14208, 14230, 14344, 14400,
14455, 14509, 14552, 14652, 15009, 15026, 15242, 15406, 15409,
15528, 15549, 15644, 15758, 15837, 15916, 15926, 15948, 20055,
20416, 20520, 20600, 20732, 20748, 20916, 21149, 21714, 23256)
```

Los datos están codificados como por seis números en el formato hmmss, donde h corresponde a las horas, mm a los minutos y ss a los segundos necesarios para completar la maratón. Antes de construir los histogramas es necesario convertir los tiempos anteriores almacenados en `maraton` a horas, para esto se utiliza el siguiente código.

```
horas <- maraton %/% 10000
min <- (maraton - horas * 10000) %/% 100
seg <- maraton - horas * 10000 - min * 100
Tiempos <- horas + min / 60 + seg / 3600
```

A continuación se muestra el código para construir cuatro histogramas con 2, 4, 8 y 16 intervalos para los tiempos a partir de la variable `Tiempos`.

```
par(mfrow=c(2,2))

hist(x=Tiempos, breaks=2, main="", xlab="Tiempo (horas)",
     ylab="Frecuencias", las=1)
mtext("(A)", side=1, line=4, font=1)

hist(x=Tiempos, breaks=4, main="", xlab="Tiempo (horas)",
     ylab="Frecuencias", las=1)
mtext("(B)", side=1, line=4, font=1)

hist(x=Tiempos, breaks=8, main="", xlab="Tiempo (horas)",
     ylab="Frecuencias")
```

```
mtext("(C)", side=1, line=4, font=1)

hist(x=Tiempos, breaks=16, main="", xlab="Tiempo (horas)",
     ylab="Frecuencias")
mtext("(D)", side=1, line=4, font=1)
```

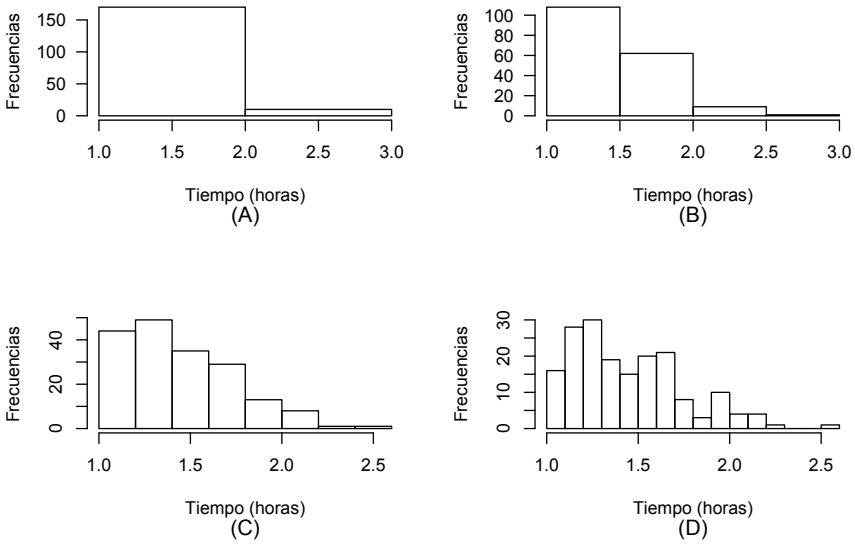

Figura 2.2: Histogramas para el tiempo en la media maratón de CONAVI. A: histograma con dos intervalos, B: histograma con cuatro intervalos, C: histograma con seis intervalos, C: histograma con 18 intervalos.

En la Figura 2.2 se presentan los cuatro histogramas. El histograma C, con 8 barras, muestra más claramente la asimetría (este es el que la mayoría de los programas produce por defecto, ya que la regla de Sturges para este conjunto de datos aproxima a 8 barras). Si consideramos más barras por ejemplo 16, como tenemos en D, se refina más la información y empezamos a notar multimodalidad. En el código anterior se incluyó `las = 1` para conseguir que los número del eje Y queden escritos de forma horizontal, ver A y B en Figura 2.2.

A continuación vamos a construir cuatro histogramas: el primero con dos intervalos intervalos y puntos de corte dados por el mínimo, la mediana y el máximo; el segundo con tres intervalos y puntos de corte dados por el mínimo, cuartiles 1, 2, 3 y máximo; el cuarto con diez intervalos y puntos de corte dados por los deciles; y el último con veinte intervalos y puntos de corte dados

por cuantiles 5, 10, ..., 95. En el código mostrado a continuación se presenta la creación de los puntos de corte y los cuatro histogramas.

```
puntos1 <- c(quantile(Tiempos, probs=c(0, 0.5, 1)))
puntos2 <- c(quantile(Tiempos, probs=c(0, 0.25, 0.5, 0.75, 1)))
puntos3 <- c(quantile(Tiempos, probs=seq(0, 1, by=0.1)))
puntos4 <- c(quantile(Tiempos, probs=seq(0, 1, by=0.05)))

par(mfrow=c(2, 2))
hist(Tiempos, breaks=puntos1, freq=FALSE, ylim=c(0,2), labels=TRUE,
     main="", ylab="Densidad")
mtext("(A)", side=1, line=4, font=1)
hist(Tiempos, breaks=puntos2, freq=FALSE, ylim=c(0,2), labels=TRUE,
     main="", ylab="Densidad")
mtext("(B)", side=1, line=4, font=1)
hist(Tiempos, breaks=puntos3, freq=FALSE, ylim=c(0,2),
     main="", ylab="Densidad")
mtext("(C)", side=1, line=4, font=1)
hist(Tiempos, breaks=puntos4, freq=FALSE, ylim=c(0,2),
     main="", ylab="Densidad")
mtext("(D)", side=1, line=4, font=1)
```

Nota: En estos histogramas, las alturas corresponden a las intensidades (frec. relativa/long. intervalo). Por tanto, el área de cada rectángulo da cuenta de las frecuencias relativas. Para el caso (A) ambos intervalos tienen igual área y cada uno contiene 50 % de los datos. esto puede verificarse así:

```
Intensidad primera clase = 1.4869888 = 0.5 / (1.384306 - 1.048056)
Intensidad segunda clase = 0.4293381 = 0.5 / (2.548889 - 1.384306)
```

2.4. Función `qqnorm` y `qqplot`

Los gráficos cuantil cuantil (quantile-quantile plot) son una ayuda para explorar si un conjunto de datos o muestra proviene de una población con cierta distribución.

La función `qqnorm` sirve para explorar la normalidad de una muestra mientras que la función `qqplot` es de propósito más general, sirve para crear el gráfico cuantil cuantil para cualquier distribución.

La estructura de las funciones con los argumentos más comunes de uso se muestran a continuación.

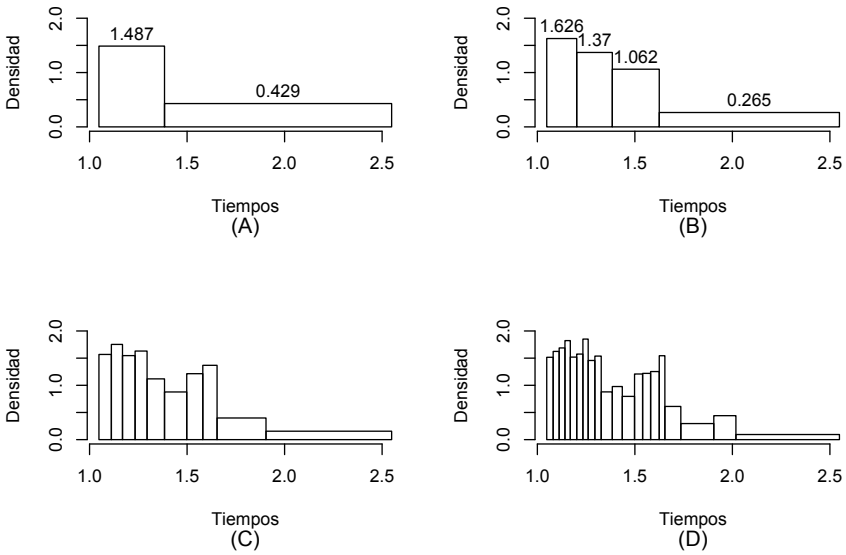

Figura 2.3: Histogramas para el tiempo en la media maratón de CONAVI. A: histograma con dos intervalos, B: histograma con cuatro intervalos, C: histograma con diez intervalos, C: histograma con veinte intervalos.

```
qqnorm(y, ...)
qqplot(y, x, ...)
```

La función `qqnorm` sólo necesita que se le ingrese el vector con la muestra por medio del parámetro `y`, la función `qqplot` necesita de la muestra en el parámetro `y` y que se ingrese en el parámetro `x` los cuantiles de la población candidata.

Existe otra función útil y es `qqline`, esta función sirve para agregar una línea de referencia al gráfico cuantil cuantil obtenido con `qqnorm`.

Ejemplo

Simular 30 observaciones de una distribución $N(\mu = 10, \sigma = 4)$ y construir el gráfico cuantil cuantil.

El código para simular la muestra y crear el gráfico cuantil cuantil se muestra a continuación.

```
muestra <- rnorm(n=30, mean=10, sd=4)

par(mfrow=c(1, 2))

qqnorm(y=muestra)
qqline(y=muestra)

qqnorm(y=muestra, main='', ylab='Cuantiles muestrales',
       xlab='Cuantiles teóricos', las=1)
qqline(y=muestra, col='blue', lwd=2, lty=2)
```

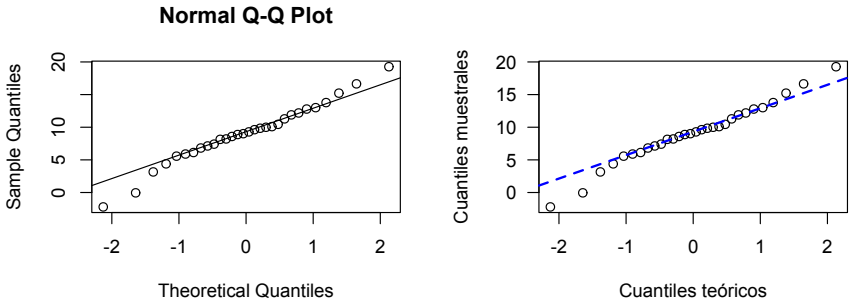

Figura 2.4: Gráfico cuantil cuantil para una muestra generada de una población normal.

En la izquierda de la Figura 2.4 está el gráfico cuantil cuantil sin editar, en la derecha se encuentra el gráfico luego de modificar los nombres de los ejes, grosor y color de la línea de referencia.

Ejemplo

Simular 100 observaciones de una distribución $Weibull(1, 1)$ y construir dos gráficos cuantil cuantil, el primero tomando como referencia los cuantiles de una $N(0, 1)$ y el segundo tomando los cuantiles de la $Weibull(1, 1)$.

El código para simular la muestra y crear los gráficos cuantil cuantil se muestra a continuación.

```
n <- 100
muestra <- rweibull(n=n, shape=1, scale=1)
```

```
par(mfrow=c(1, 2))
qqplot(y=muestra, x=qnorm(ppoints(n)))
qqplot(y=muestra, x=qweibull(ppoints(n), shape=1, scale=1))
```

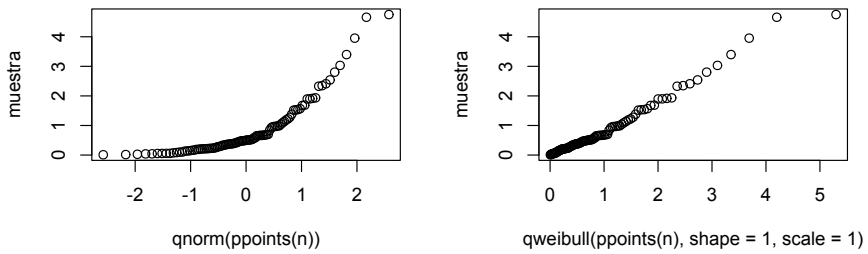

Figura 2.5: Gráfico cuantil cuantil para una muestra generada de una población Weibull.

En la Figura 2.5 están los gráficos cuantil cuantil solicitados. Del pánel izquierdo de la figura vemos que los puntos NO están alineados, esto indica que la muestra no proviene de la distribución $N(0, 1)$, esto es un resultado esperado ya que sabíamos que la muestra no fue generada de una normal. En el pánel derecho de la misma figura vemos que los puntos SI están alineados, esto indica que la muestra generada puede provenir de una población $Weibull(1, 1)$. Los nombres de los ejes en la Figura 2.5 pueden ser editados para presentar una figura con mejor apariencia.

2.5. Función `density`

Los gráficos de densidad son muy útiles porque permiten ver el(los) intervalo(s) donde una variable cuantitativa puede ocurrir con mayor probabilidad.

La función `density` crea la información de la densidad y la función `plot` dibuja la densidad.

La estructura de la función `density` con los argumentos más comunes de uso se muestra a continuación.

```
density(x, bw, adjust=1, kernel='gaussian', na.rm=FALSE)
```

Los argumentos de la función `density` son:

- `x`: vector con los datos para los cuales se quiere la densidad.
- `bw`: ancho de banda.
- `kernel`: núcleo de suavización a usar, los posibles valores son `gaussian`, `rectangular`, `triangular`, `epanechnikov`, `biweight`, `cosine` o `optcosine`, el valor por defecto es `gaussian`.
- `na.rm`: valor lógico, si es `TRUE` se eliminan los valores con `NA` para construir la densidad, el valor por defecto es `FALSE`.

Ejemplo

Simular mil observaciones de una $N(0, 1)$, aplicar la función `density` al vector y explorar el contenido de la salida.

Primero se generan las observaciones y se almacenan en el objeto `y`, luego se aplica la función `density` y el resultado se guarda en el objeto `res`, para explorar lo que almacena `res` se usa la función `names`. A continuación el código utilizado.

```
y <- rnorm(n=1000)
res <- density(y)
names(res)
```

```
## [1] "x"         "y"         "bw"        "n"
## [5] "call"      "data.name" "has.na"
```

De la salida anterior se observa que la lista `res` tiene 7 elementos, los dos primeros son los vectores con las coordenadas para dibujar la densidad, los restantes elementos con información adicional.

Ejemplo

Con los datos generados en el ejemplo anterior construir la densidad para varios núcleo y para varios valores de ancho de banda.

En el siguiente código se construyen 4 densidades para diferentes núcleos.

```
par(mfrow=c(2, 2))
plot(density(y, kernel='gaussian'))
```

```
plot(density(y, kernel='triangular'))
plot(density(y, kernel='cosine'))
plot(density(y, kernel='rectangular'))
```

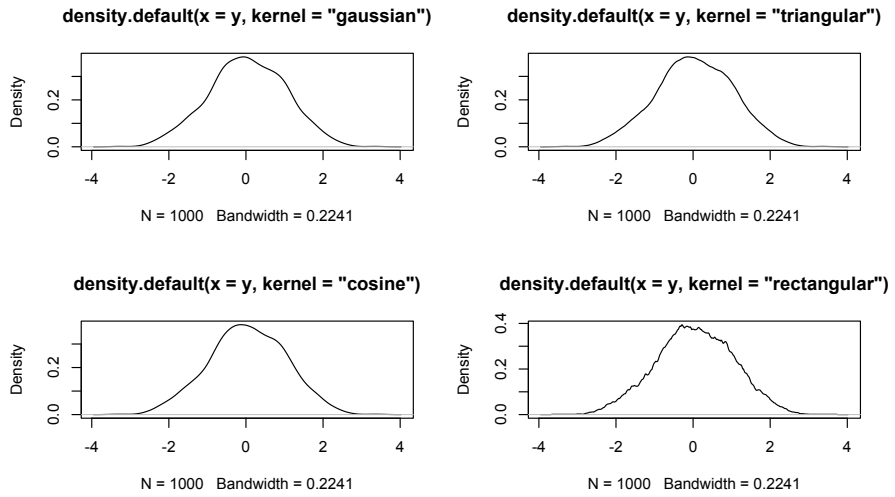

Figura 2.6: Densidad para una muestra aleatoria de una N(0, 1) cambiando el núcleo de la densidad.

En la Figura 2.6 se muestran las densidades para 4 elecciones del núcleo. En la práctica se usa el núcleo que está por defecto (`gaussian`) ya que el objetivo de una densidad es ver la zonas donde es más probable encontrar observaciones de la variable.

En el siguiente código se construyen 4 densidades para diferentes anchos de banda.

```
par(mfrow=c(2, 2))
plot(density(y, bw=0.1))
plot(density(y, bw=0.2241))  # bw obtenido antes
plot(density(y, bw=0.5))
plot(density(y, bw=1))
```

En la Figura 2.7 se muestran las densidades para 4 elecciones del parámetro ancho de banda `bw`, el valor de 0.2241 fue el valor calculado automáticamente por R y fue obtenido de la Figura 2.6, los otros valores fueron elegidos arbitrariamente para ver los cambios en la densidad. El usar un ancho de banda

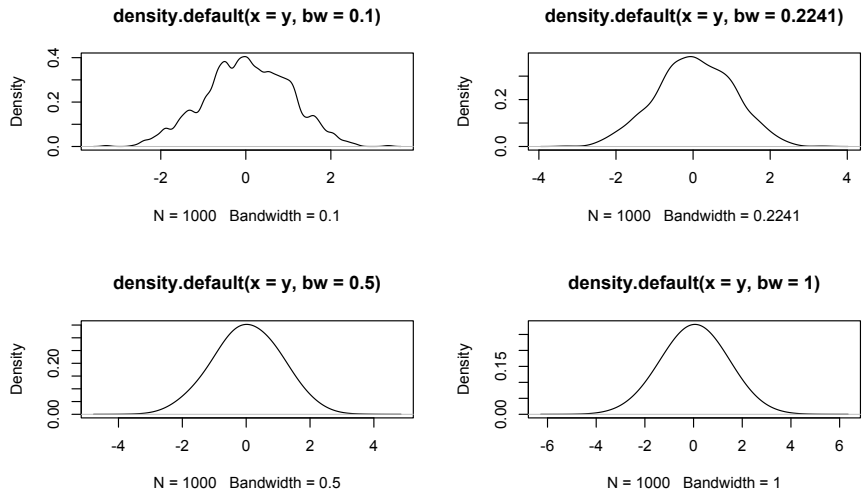

Figura 2.7: Densidad para una muestra aleatoria de una N(0, 1) cambiando el ancho de banda.

pequeño la densidad queda muy rugosa y usar un valor muy grande la suaviza, se recomienda usar el valor automático.

Ejemplo

Construir un gráfico de densidad para la variable peso corporal de la base de datos `medidas_cuerpo`, luego construir la densidad para la misma variable pero diferenciando por sexo.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/medidas_cuerpo'
datos <- read.table(file=url, header=T)

par(mfrow=c(1, 2))
plot(density(datos$peso), main='Distribución del peso corporal',
     xlab='Peso corporal (kg)', ylab='Densidad', lwd=4)

den.hom <- with(datos, density(peso[sexo == 'Hombre']))
den.muj <- with(datos, density(peso[sexo == 'Mujer']))

plot(den.hom, xlim=c(20, 120),
     main='Distribución del peso corporal por género', ylab='Densidad',
     xlab='Peso corporal (kg)', lwd=4, col='blue')
```

```
lines(den.muj, lwd=4, col='red')
legend('topright', legend=c('Hombres', 'Mujeres'), bty='n',
       lwd=3, col=c('blue', 'red'))
```

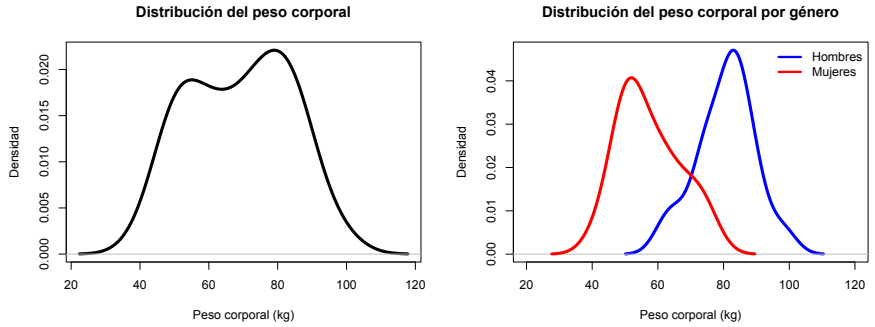

Figura 2.8: Densidad para la variable peso en la izquierda, densidad para el peso diferenciando por sexo a la derecha.

En el panel izquierdo de la Figura 2.8 se muestra la densidad para la variable peso, de esta figura se observa que tiene dos sectores de mayor densidad, alrededor de 50 kg y alrededor de 80 kg. En el panel izquierdo están la densidades del peso corporal para hombres y mujeres, aquí se observa claramente la diferencia entre los pesos de hombres y mujeres.

3

Gráficos para varias variables cuantitativas

En este capítulo se presentan funciones para la creación de gráficos que involucran varias variables.

3.1. Función `plot`

Los gráficos de dispersión son muy útiles porque permiten ver la relación que existe entre dos variables cuantitativas, la función `plot` permite crear este tipo de gráficos. La estructura de la función `plot` con los argumentos más usuales se muestran a continuación

```
plot(x, y, type, main, sub, xlab, ylab)
```

Los argumentos de la función `plot` son:

- `x`: vector numérico con las coordenadas del eje horizontal.
- `y`: vector numérico con las coordenadas del eje vertical
- `type`: tipo de gráfico a dibujar. Las opciones son:
  - `'p'` para obtener puntos, esta es la opción por defecto.
  - `'l'` para obtener líneas.
  - `'b'` para obtener los puntos y líneas que unen los puntos.
  - `'c'` para obtener sólo las líneas y dejando los espacios donde estaban los puntos obtenidos con la opción `'b'`.
  - `'o'` para obtener los puntos y lineas superpuestas.
  - `'h'` para obtener líneas verticales desde el origen hasta el valor y_i de cada punto, similar a un histograma.
  - `'s'` para obtener escalones.
  - `'S'` similar al anterior.
  - `'n'` para que no dibuje.
- `...`: otros parámetros gráficos que pueden ser pasados como argumentos para `plot`.

Ejemplo

Crear 16 parejas de puntos tales que $x = -5, -4, \ldots, 9, 10$ con $y = -10 + (x - 3)^2$, dibujar los nueve diagramas de dispersión de y contra x usando todos los valores posibles para el parámetro `type`.

A continuación se muestra el código para crear las 16 parejas de x e y. Los nueve diagramas de dispersión se observan en la Figura 3.1, de esta figura se observa claramente el efecto que tiene el parámetro `type` en la construcción del diagrama de dispersión.

```
x <- -5:10
y <- -10 + (x-3)^2
par(mfrow=c(3, 3))
plot(x=x, y=y, type='p', main="con type='p'")
plot(x=x, y=y, type='l', main="con type='l'")
plot(x=x, y=y, type='b', main="con type='b'")
plot(x=x, y=y, type='c', main="con type='c'")
plot(x=x, y=y, type='o', main="con type='o'")
plot(x=x, y=y, type='h', main="con type='h'")
plot(x=x, y=y, type='s', main="con type='s'")
plot(x=x, y=y, type='S', main="con type='S'")
plot(x=x, y=y, type='n', main="con type='n'")
```

Ejemplo

Como ilustración vamos a crear un diagrama de dispersión entre el precio de apartamentos usados en la ciudad de Medellín y el área de los apartamentos. El código necesario para cargar la base de datos y construir el diagrama de dispersión se muestra a continuación.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/aptos2015'
datos <- read.table(file=url, header=T)

par(mfrow=c(1, 2))
plot(x=datos$mt2, y=datos$precio)
plot(x=datos$mt2, y=datos$precio, pch='.',
     xlab='Área del apartamento (m2)', ylab='Precio (millones de pesos)')
```

En la Figura 3.2 se presenta el diagrama de dispersión entre precio y área de los apartamentos, de este diagrama se observa claramente que a medida que los apartamentos tienen mayor área el precio promedio y la variabilidad del precio aumentan. Para el diagrama de dispersión de la derecha se usó el parámetro

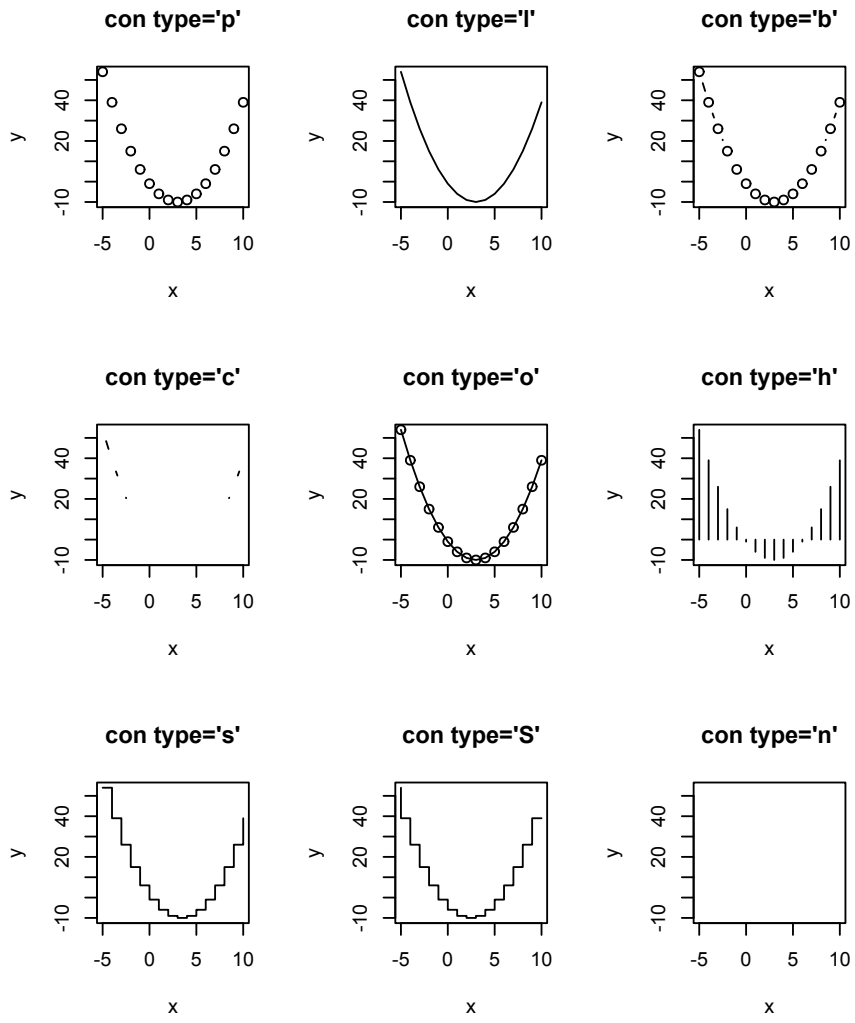

Figura 3.1: Efecto del parámetro `type` en la función `plot`.

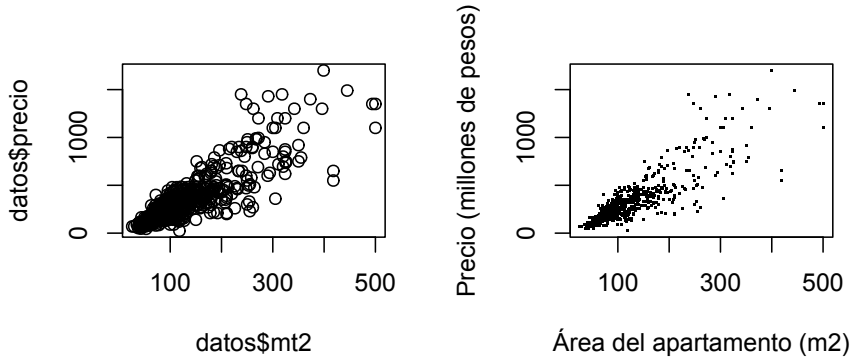

Figura 3.2: Diagrama de dispersión del precio del apartamento versus área del apartamento. A la izquierda el diagrama de dispersión sin editar y a la derecha el diagrama de dispersión mejorado

`pch='.'` con el objetivo de obtener pequeños puntos que representen cada apartamento y que no se traslapen debido a que se tienen 694 observaciones en la base de datos.

3.2. Función `pairs`

Las matrices de dispersión obtenidas con la función `pairs` proporcionan un método simple de presentar las relaciones entre pares de variables cuantitativas y son la versión múltiple de la función `plot`. Este gráfico consiste en una matriz donde cada entrada presenta un gráfico de dispersión sencillo. Un inconveniente es que si tenemos muchas variables el tamaño de cada entrada se reduce demasiado impidiendo ver con claridad las relaciones entre los pares de variables. La celda (i, j) de una matriz de dispersión contiene el gráfico de dispersión de la columna i versus la columna j de la matriz de datos.

En la Figura 3.3 se muestra un ejemplo de una matriz de dispersión para un conjunto de datos, en la diagonal están los nombres de las variables y por fuera de la diagonal están los diagramas de dispersión para cada combinación de variables.

La estructura de la función `pairs` con los argumentos más usuales se muestran a continuación.

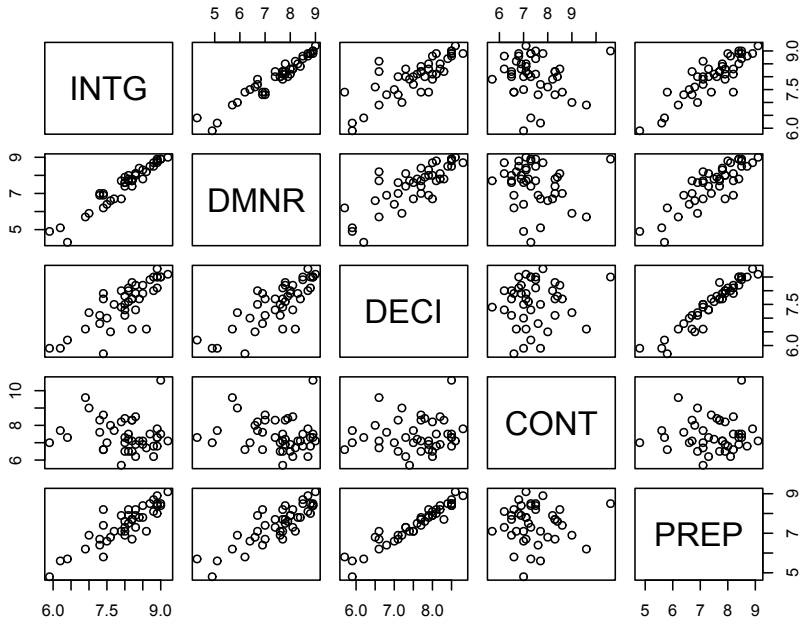

Figura 3.3: Ilustración de una matriz de dispersión.

```
pairs(x, labels, panel = points, ...,
      horInd = 1:nc, verInd = 1:nc,
      lower.panel = panel, upper.panel = panel,
      diag.panel = NULL, text.panel = textPanel,
      label.pos = 0.5 + has.diag/3, line.main = 3,
      cex.labels = NULL, font.labels = 1,
      row1attop = TRUE, gap = 1, log = "")
```

Los argumentos de la función `pairs` son:

- `x`: matriz o marco de datos con la información de las variables cuantitativas a incluir en la matriz de dispersión.
- `labels`: vector opcional con los nombres a colocar en la diagonal, por defecto se usan los nombres de columnas del objeto `x`.
- `panel`: función usual de la forma `function(x,y,...)` a ser usada para determinar el contenido de los páneles. Por defecto es `points`, indicando que se graficarán los puntos de los pares de variables. Es posible utilizar aquí otras funciones diseñadas por el usuario.

- `...`: Indica que es posible agregar otros parámetros gráficos, tales como `pch` y `col`, con los cuales puede especificarse un vector de símbolos y colores a ser usados en los scatterplots.
- `lower.panel, upper.panel`: función usual de la forma `function(x,y,...)` para definir lo que se desea dibujar en los paneles abajo y arriba de la diagonal.
- `diag.panel`: función usual de la forma `function(x,y,...)` para definir lo que se desea dibujar en la diagonal.
- `text.panel`: Es opcional. Permite que una función: `function(x, y, labels, cex, font, ...)` sea aplicada a los paneles de la diagonal.
- `label.pos`: Para especificar la posición y de las etiquetas en el text panel.
- `cex.labels, font.labels`: Parámetros para la expansión de caracteres y fuentes a usar en las etiquetas de las variables.
- `row1attop`: Parámetro lógico con el cual se especifica si el gráfico para especificar si el diseño lucirá como una matriz con su primera fila en la parte superior o como un gráfico con fila uno en la base. Por defecto es lo primero.

Ejemplo

Dibujar una matriz de dispersión para las variables precio, área, número de alcobas y número de baños de la base de datos sobre apartamentos en Medellín.

A continuación se muestra el código usado para crear el gráfico solicitado. El objeto `datos` corresponde a la base de datos completa mientras que `datos.num` es el marco de datos sólo con las variables de interés precio, área, número de alcobas y número de baños.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/aptos2015'
datos <- read.table(file=url, header=T)
datos.num <- datos[, c('precio', 'mt2', 'alcobas', 'banos')]

pairs(datos.num)
```

En la Figura 3.4 se muestra la matriz de dispersión para las variables del marco de datos `datos.num`.

Ejemplo

Volver a construir la Figura 3.4 editando los nombres de las variables, usando cruces rojas en lugar de puntos, en escala logaritmica, con marcas horizontales en el eje vertical y eliminando los diagramas de dispersión abajo de la diagonal.

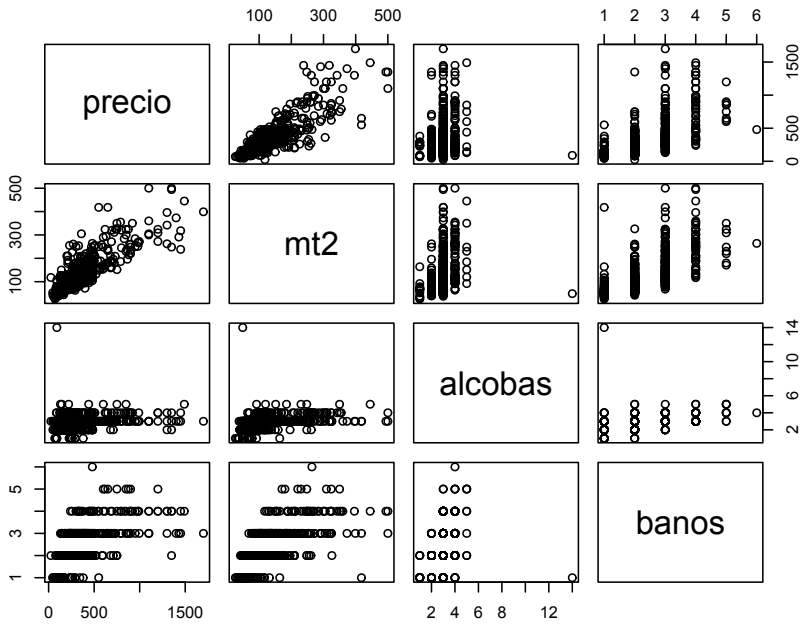

Figura 3.4: Matriz de dispersión para las variables precio, área, número de alcobas y número de baños de la base de datos sobre apartamentos en Medellín.

Para obtener la nueva matriz de dispersión con los cambios solicitados se usa el siguiente código. En la Figura 3.5 se presenta la nueva matriz de dispersión.

```
pairs(datos.num, lower.panel=NULL, cex.labels=1.5, log='xy',
      main='Matriz de dispersión', las=1,
      labels=c('Precio', 'Área', 'Num alcobas', 'Num baños'),
      pch=3, cex=0.6, col='red')
```

Ejemplo

Construir una matriz de dispersión con las variables precio, área y avaluo para apartamentos que cumplan la condición $100m^2 < area < 130m^2$. Adicionalmente, se deben diferenciar los apartamentos sin parqueadero con color rojo y los apartamentos con parqueadero con color verde.

Para crear una matriz de dispersión se puede tambien usar la base de datos

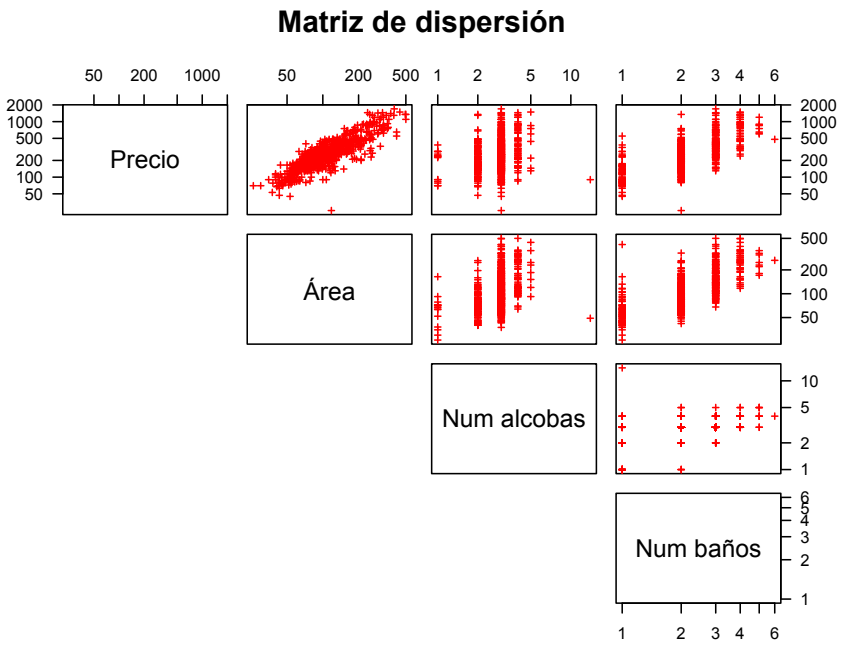

Figura 3.5: Matriz de dispersión modificando los parámetros adicionales de la función pairs.

original llamada `datos` que contiene todas las variables y usar una fórmula con la ayuda del operador `~` para indicar las variables de interés. La fórmula **NO** debe contener nada del lado izquierdo mientras que en el lado derecho se colocan todas las variables a considerar en la matriz de dispersión, por esta razón es que en el códido mostrado abajo se inicia con la instrucción `~ precio + mt2 + avaluo`. Para incluir condiciones se usa el parámetro `subset` de la siguiente manera: `subset=mt2 > 100 & mt2 < 130`. A continuación el código completo para construir la matriz de dispersión solicitada.

```
col1 <- ifelse(datos$parqueadero == 'no', 'red', 'green3')
pairs(~ precio + mt2 + avaluo, data=datos,
      lower.panel=NULL, col=col1,
      subset=mt2 > 100 & mt2 < 130, pch=19, cex=0.8,
      main="Matriz de dispersión para aptos con
      100 < área < 130 mt2")
```

En la Figura 3.6 se presenta la matriz de dispersión solicitada, los puntos rojos

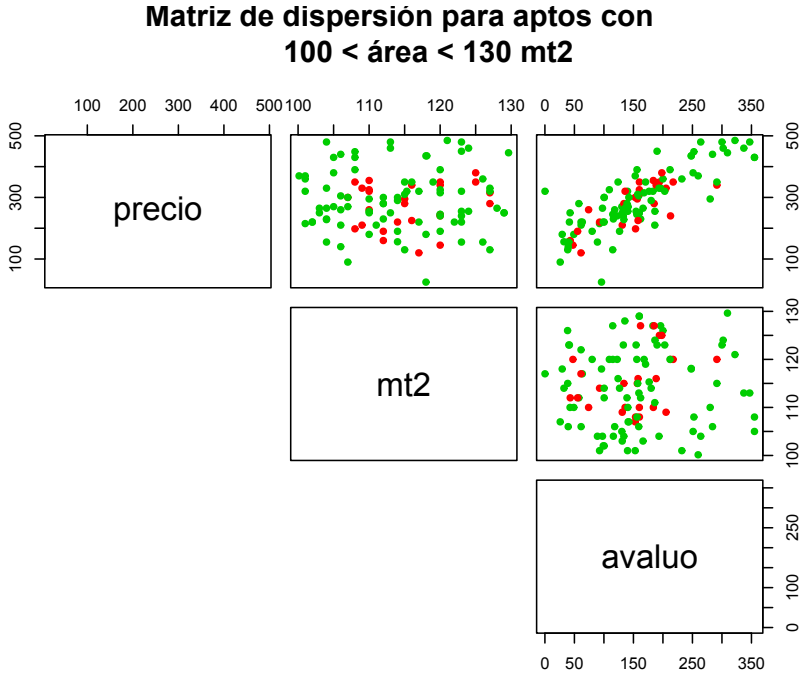

Figura 3.6: Matriz de dispersión con un subconjunto de los datos y con colores para identificar los puntos.

representan los apartamento sin parqueadero mientras que los puntos verdes son los apartamento que si tienen parqueadero.

Ejemplo

¿Es posible agregar una leyenda a una matriz de dispersión?

Claro que es posible, se construye la matriz de dispersión y se deja en el lienzo del dibujo un espacio para colocar la leyenda. A continuación se muestra un ejemplo disponible en Stackoverflow[1]. A continuación se muestra el código para el ejemplo y en la Figura 3.7 se presenta el resultado.

```
pairs(iris[1:4], main="Anderson's Iris Data -- 3 species",
      pch=21, bg=c("red", "green3", "blue")[iris$Species],
      oma=c(4, 4, 6, 12))
```

[1] http://stackoverflow.com/questions/14948852/how-to-use-the-pairs-function-combined-with-layout-in-r

```
par(xpd=TRUE)
legend(0.85, 0.7, as.vector(unique(iris$Species)), bty='n',
       fill=c("red", "green3", "blue"))
```

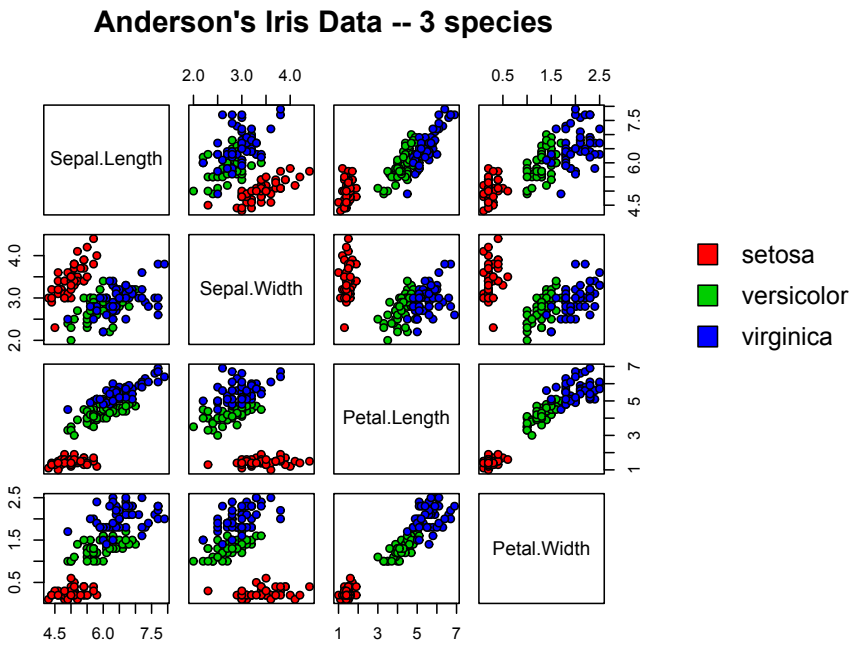

Figura 3.7: Matriz de dispersión con leyenda.

Ejemplo

¿Es posible modificar el contenido de los páneles de una matriz de dispersión?

Claro que es posible, para hacer esto se definen funciones que hagan lo que se desea ver tanto en la diagonal como arriba y abajo de la misma.

Como ejemplo vamos a construir una matriz de dispersión que cumpla:

- sobre la diagonal un diagrama de dispersión para las variables involucradas y la recta de regresión ajustada,
- en la diagonal un histograma para la variable,
- debajo de la diagonal el coeficiente de correlación entre las variables involucradas y usando un tamaño de fuente proporcional a la fuerza de correlación.

Para obtener esta matriz de dispersión especial se definen a continuación las funciones `panel.reg`, `panel.hist` y `panel.cor`, a continuación el código utilizado. Luego se usa la función `pairs` y se indica qué función debe actuar en cada uno de los parámetros `upper.panel`, `diag.panel` y `lower.panel`.

```
# Función para dibujar los puntos y agregar la recta de regresión
panel.reg <- function (x, y)
{
  points(x, y, pch=20)
  abline(lm(y ~ x), lwd=2, col='dodgerblue2')
}

# Función para crear el histograma
panel.hist <- function(x, ...)
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5) )
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks; nB <- length(breaks)
  y <- h$counts; y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col="dodgerblue2", ...)
}

# Función para obtener la correlación
panel.cor <- function(x, y, digits=2, prefix="", cex.cor)
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y))
  txt <- format(c(r, 0.123456789), digits=digits)[1]
  txt <- paste(prefix, txt, sep="")
  if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
  text(0.5, 0.5, txt, cex = cex * r)
}

pairs(datos.num,
      upper.panel = panel.reg,
      diag.panel = panel.hist,
      lower.panel = panel.cor)
```

En la Figura 3.8 se presenta la matriz de dispersión con las modificaciones en cada uno de los páneles. Cualquier usuario puede modificar las funciones `panel.reg`, `panel.hist` y `panel.cor` para personalizar la apariencia de los contenidos.

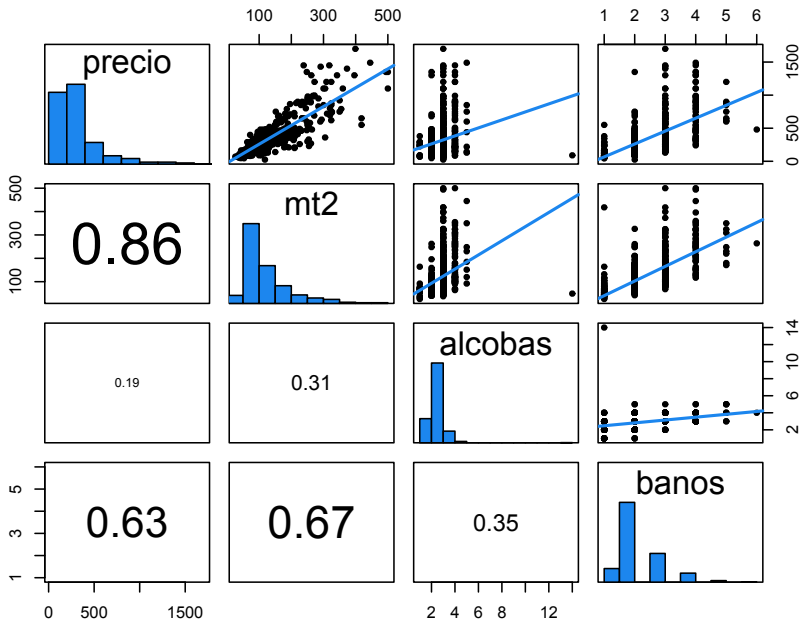

Figura 3.8: Matriz de dispersión con páneles modificados.

La función `panel.smooth` está disponible en R para que el usuario pueda incluir arriba o abajo de la diagonal un diagrama de dispersión con una línea resultado de un ajuste suavizado. Abajo se muestra el código de cómo incluir la función `panel.smooth` y en la Figura 3.9 se muestra gráfico obtenido.

```
pairs(datos.num,
      upper.panel = panel.reg,
      diag.panel = panel.hist,
      lower.panel = panel.smooth)
```

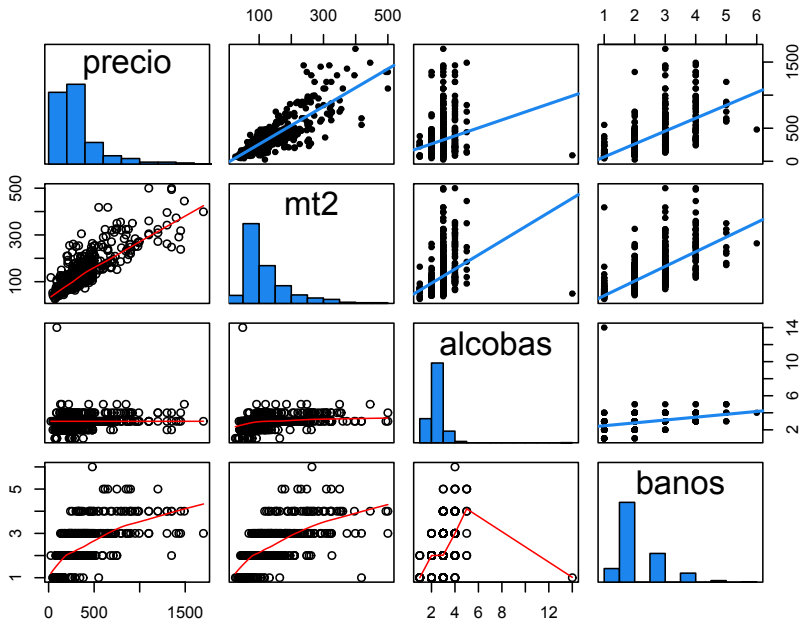

Figura 3.9: Matriz de dispersión usando la función panel.smooth.

3.3. Función `persp`

La función `persp` dibuja superfices en tres dimensiones y es posible rotar la superficie para obtener una perpectiva apropiada. La estructura de la función `persp` con los argumentos más usuales se muestran a continuación.

```
persp(x, y, z, main, sub, theta, phi, r, col,
      border, box, axes, nticks)
```

Los argumentos de la función `plot` son:

- `x`: vector numérico con los valores de x donde fue evaluada la función o superficie.
- `y`: vector numérico con los valores de y donde fue evaluada la función o superficie.
- `z`: matriz que contiene las alturas z de la supercifie para cada combinación de x e y.

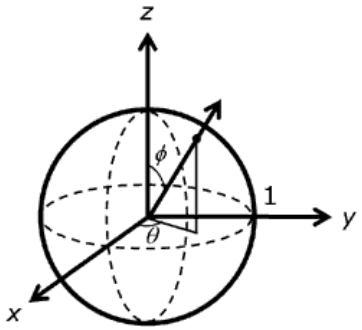

Figura 3.10: Ilustración de los ángulos theta y phi para la función persp. Figura tomada de https://i-msdn.sec.s-msft.com/dynimg/IC412528.png

- `main`: vector numérico con las coordenadas del eje vertical.
- `sub`: vector numérico con las coordenadas del eje vertical.
- `theta, phi`: ángulo para la visión de la superficie, `theta` para la dirección azimutal y `phi` para latitud. Ver Figura 3.10 para una ilustración de los ángulos.
- `r`: distancia entre el centro de la caja de dibujo al punto de vista.
- `col`: color de la superficie.
- `border`: color para el borde de la superficie.
- `box`: valor lógico para indicar si se quiere dibujar la caja que contiene la superficie, por defecto es `TRUE`.
- `axes`: valor lógico para indicar si se desean marcas en los ejes y nombres de los ejes, por defecto es `TRUE`. Si `box='FALSE'` no aparecen marcas ni nombres de los ejes.
- `expand`: factor de expansión aplicado a los valores en el eje `z`.
- `ticktype`: tipo de marcas a colocar en los ejes, `simple` no dibuja nada y `detailed` coloca números a los ejes.
- `nticks`: número aproximado de marcas en los ejes.

Ejemplo

Dibujar la superficie asociada a la función $f(x, y) = sen(x^2 + y^2)$ para $-2 \leq x \leq 2$ y $-2 \leq y \leq 2$. Usar 4 combinaciones de los parámetros `theta` y `phi` para obtener un buen punto de vista de la superficie.

Lo primero que se debe hacer es crear la función $f(x, y)$ la cual se va a llamar `fun`. Luego se definen los vectores `x` e `y` tomando por ejemplo 25 puntos equiespaciados en el intervalo $[-2, 2]$. Luego se usa la función `outer` para crear la rejilla o matriz que contiene los valores de $f(x, y)$ para cada combinación de `x` e `y`, los resultados se almacenan en el objeto `z`. Por último se dibujan 4

perspectivas de la función variando los parámetros `theta` y `phi` de la función `persp`. A continuación el código utilizado.

```
fun <- function(x, y)   sin(x^2 + y^2)
x <- seq(from=-2, to=2, length.out=25)
y <- seq(from=-2, to=2, length.out=25)
z <- outer(x, y, fun)

par(mfrow=c(2, 2), mar=c(1, 1, 2, 1))
persp(x, y, z, zlim=c(-1, 1.5), theta=0, phi=0, col='aquamarine',
      main='(A) theta=0, phi=0')
persp(x, y, z, zlim=c(-1, 1.5), theta=15, phi=15, col='lightpink',
      main='(B) theta=15, phi=15')
persp(x, y, z, zlim=c(-1, 1.5), theta=45, phi=30, col='yellow1',
      main='(c) theta=45, phi=30')
persp(x, y, z, zlim=c(-1, 1.5), theta=60, phi=50, col='lightblue',
      main='(D) theta=60, phi=50')
```

En la Figura 3.11 se presentan las 4 perspectivas de la función $f(x, y) = sen(x^2 + y^2)$. De los 4 páneles se nota que (C) y (D) muestran mejor la superficie de interés.

Al aumentar el valor del parámetro `length.out` en la creación de los vectores `x` e `y` se obtendrá una rejilla más tupida, se recomienda modificar este valor para obtener una superficie apropiada.

Ejemplo

Dibujar la superficie de una distribución normal bivariada con vector de medias $\boldsymbol{\mu} = (5, 12)^\top$, varianzas unitarias y covarianza con valor de -0.8. Explorar el efecto de los parámetros `ticktype`, `nticks`, `expand`, `axes` y `box`.

Primero se define el vector de medias y la matriz de varianzas y covarianzas, luego se carga el paquete `mvtnorm` que contiene la función `dmvnorm` que calcula la densidad dado el vector de medias y la matriz de varianzas y covarianzas. Se construye la función `fun` y se vectoriza para luego obtener las alturas de la superficie con la ayuda de `outer`. Por último se dibujan tres perspectivas diferentes para la densidad modificando los parámetros `ticktype`, `nticks`, `expand`, `axes` y `box`, a continuación el código usado.

```
media <- c(5, 12)
varianza <- matrix(c(1, -0.8, -0.8, 1), ncol=2)

require(mvtnorm)
```

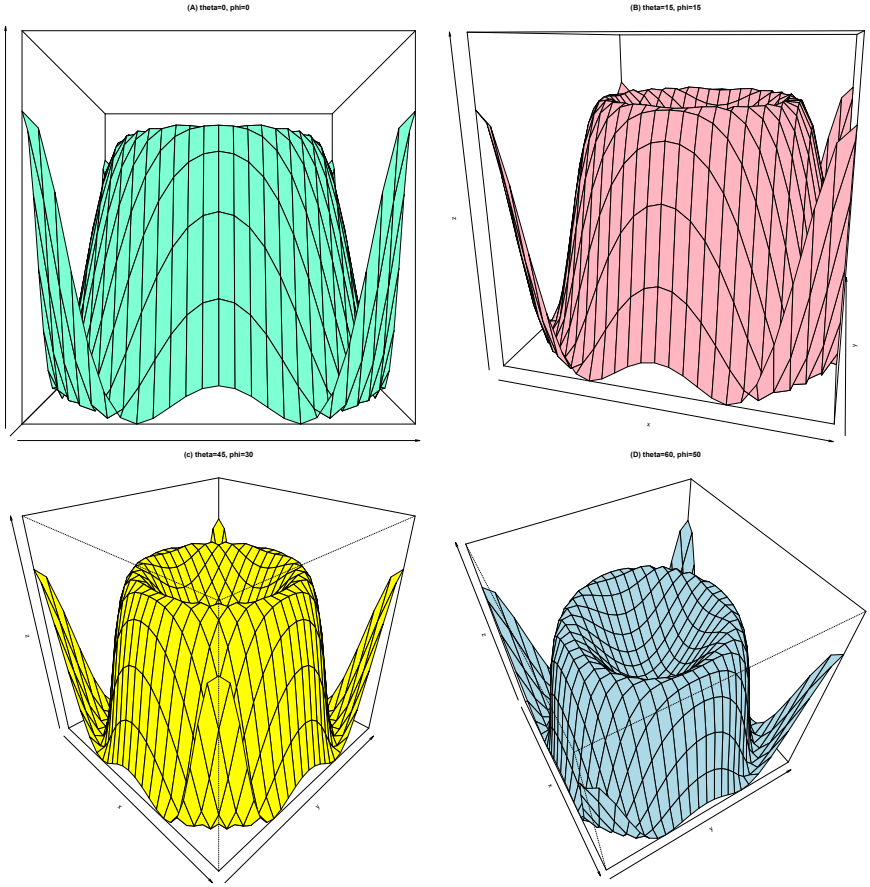

Figura 3.11: Superficie generada con `persp` y diferentes valores de theta y phi.

```
fun <- function(x, y) dmvnorm(c(x, y), mean=media, sigma=varianza)
fun <- Vectorize(fun)

x <- seq(from=2, to=8, length.out=30)
y <- seq(from=9, to=15, length.out=30)
z <- outer(x, y, fun)

par(mfrow=c(1, 3), mar=c(1, 1, 2, 1))
persp(x, y, z, theta=30, phi=30, ticktype = "detailed", nticks=4)
persp(x, y, z, theta=30, phi=30, col='salmon1', expand=0.5, axes=FALSE)
persp(x, y, z, theta=30, phi=30, col='springgreen1', expand=0.2, box=FALSE)
```

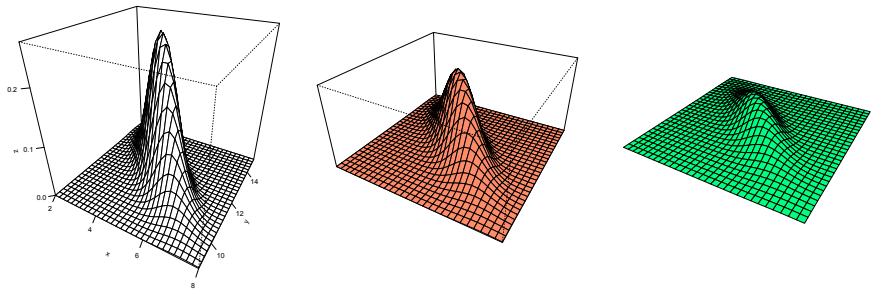

Figura 3.12: Distribución normal bivariada.

En la Figura 3.12 se presentan las 3 perspectivas para la densidad. Note los efectos que `ticktype`, `nticks`, `expand`, `axes` y `box` tienen sobre los dibujos de las perspectivas.

3.4. Función `contour`

La función `contour` dibuja gráficos contornos. La estructura de la función `contour` con los argumentos más usuales se muestran a continuación.

```
contour(x, y, z,
        xlim, ylim, zlim,
        levels, nlevels=20, col)
```

Los argumentos de la función son:

- `x, y`: vectores numéricos en los cuales se evaluó la función de interés para construir el objeto `z`. Ambos vectores deben estar ordenados.
- `z`: matriz con las alturas de la función de interés, por lo general creada con la función `outer`.
- `xlim, ylim, zlim`: límites de los ejes x, y e z respectivamente.
- `nlevels`: número aproximado de niveles o cortes en la superficie a representar.
- `col`: color a usar en las líneas de contornos.

La función `contour` tiene otros parámetros adicionales que el lector puede consultar en la ayuda usando `help(contour)`.

Ejemplo

Generar una muestra aleatoria de 50 observaciones de una distribución normal con parámetros $\mu = 170$ y $\sigma^2 = 25$. Dibujar un gráfico de contornos para la superficie de log-verosimilitud.

La muestra aleatoria se genera con el siguiente código.

```
y <- rnorm(n=50, mean=170, sd=5)  # sd es desviación
```

Para dibujar los contornos solicitados se debe primero construir la función de log-verosimilitud llamada `ll`. A continuación el código para crear `ll`, mayores detalles de cómo construir funciones de log-verosimilitud se pueden consultar en Hernández (2018).

```
ll <- function(a, b) sum(dnorm(x=y, mean=a, sd=b, log=TRUE))
ll <- Vectorize(ll)  # Para vectorizar la función
```

Una vez construída la función `ll` se deben construir los vectores con las coordenadas horizontal y vertical donde se evalua la función `ll`. En el código mostrado abajo se tienen dos vectores `xx` e `yy` obtenidos como secuencias desde el menor valor hasta el mayor valor para cada uno de los parámetros μ y σ de la distribución normal, el valor `by=0.5` indica el tamaño de paso de la secuencia. Luego se construye la matriz `zz` usando la función `outer` evaluando `ll` en `xx` e `yy`. Por último la función `contour` se aplica sobre los elementos `xx`, `yy` e `zz`. En la Figura 3.13 se muestra el gráfico de contornos con aproximadamente 50 niveles.

```
xx <- seq(from=160, to=180, by=0.5)
yy <- seq(from=3, to=7, by=0.5)
zz <- outer(X=xx, Y=yy, ll)
contour(x=xx, y=yy, z=zz, nlevels=50,
        col=gray(0.3), lwd=2, lty='solid',
        xlab=expression(mu), ylab=expression(sigma))
```

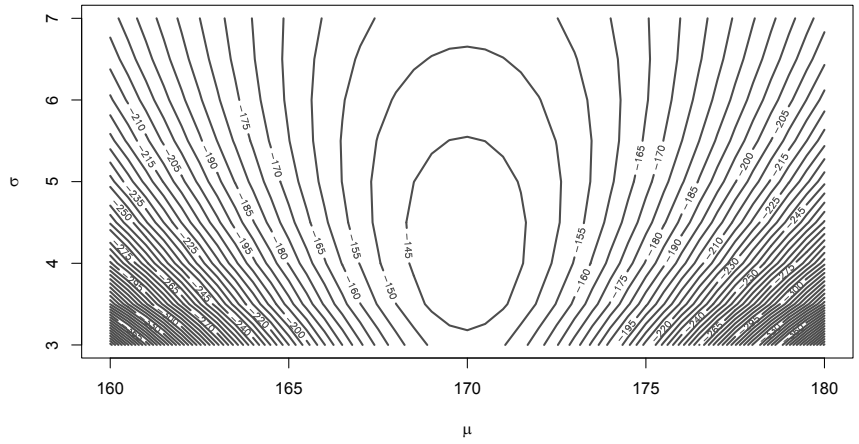

Figura 3.13: Gráfico de contornos para la función de log-verosimilitud para el ejemplo sobre normal.

3.5. Función `filled.contour`

La función `filled.contour` dibuja una especie de gráficos contornos pero usando una paleta de colores. La estructura de la función `filled.contour` con los argumentos más usuales se muestran a continuación.

```
filled.contour(x, y, z,
               xlim, ylim, zlim,
               levels, nlevels=20,
               color.palette=cm.colors, col)
```

Los argumentos de la función son:

- `x, y`: vectores numéricos en los cuales se evaluó la función de interés para construir el objeto `z`. Ambos vectores deben estar ordenados.
- `z`: matriz con las alturas de la función de interés, por lo general creada con la función `outer`.
- `xlim, ylim, zlim`: límites de los ejes x, y e z respectivamente.
- `nlevels`: número aproximado de niveles o cortes en la superficie a representar.

- `color.palette`: paleta de colores a usar. Por defecto es `cm.colors` pero el usuario puede elegir entre `heat.colors`, `terrain.colors` o `topo.colors`.

La función `filled.contour` tiene otros parámetros adicionales que el lector puede consultar en la ayuda usando `help(filled.contour)`.

Ejemplo

Para la muestra aleatoria obtenida en el ejemplo anterior, dibujar un gráfico de nivel para la superficie de log-verosimilitud.

Usando los objetos `xx`, `yy` e `zz` creados en el ejemplo anterior se puede construir el gráfico de niveles, a continuación el código utilizado. En la Figura 3.14 se muestra el gráfico de niveles con aproximadamente 20 niveles.

```
filled.contour(x=xx, y=yy, z=zz, nlevels=20,
               xlab=expression(mu), ylab=expression(sigma),
               color = topo.colors)
```

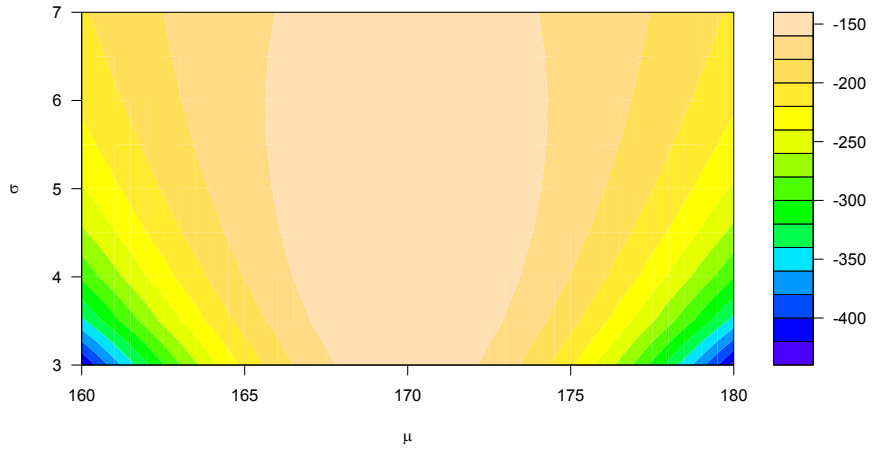

Figura 3.14: Gráfico de nivel para la función de log-verosimilitud para el ejemplo sobre normal.

3.6. Función `interaction.plot`

La función `interaction.plot` dibuja gráficos de interacción. La estructura de la función `interaction.plot` con los argumentos más usuales se muestran a continuación.

```
interaction.plot(response, x.factor, trace.factor, fun,
                 legend, trace.label)
```

Los argumentos de la función son:

- `response`: vector numérico con la variable respuesta.
- `x.factor`: factor 1 a ubicar en el eje horizontal.
- `trace.factor`: factor 2 para diferenciar las líneas.
- `fun`: función a aplicar para a `response` para cada combinación de `x.factor` y `trace.factor`.
- `legend`: valor lógico para incluir o no leyenda.
- `trace.label`: ombre a colocar en la leyenda.

La función `interaction.plot` tiene otros parámetros adicionales que el lector puede consultar en la ayuda usando `help(interaction.plot)`.

Ejemplo

Se realizó un experimento para determinar cómo influye el material de la batería y la temperatura del medio ambiente sobre la duración en horas de la batería. Se desea construir un gráfico de interacción entre Temperatura y Material para ver el efecto sobre la duración promedio de las baterías. Los datos y el código para generar el gráfico solicitado se muestran a continuación.

```
horas <- c(130, 155, 74, 180, 150, 188, 159, 126, 138, 110, 168,
           160, 34, 40, 80, 75, 136, 122, 106, 115, 174, 120, 150,
           139, 20, 70, 82, 58, 25, 70, 58, 45, 96, 104, 82, 60)

temperatura <- rep(c(15, 70, 125), each=12)
material    <- rep(1:3, each=4, times=3)

interaction.plot(x.factor=temperatura, trace.factor=material,
                 response=horas, trace.label='Material',
                 xlab='Temperatura',
                 ylab='Duración promedio (horas)',
```

```
                col=c('blue', 'black', 'red'),
                fun=mean, lwd=3, las=1, fixed=T)
```

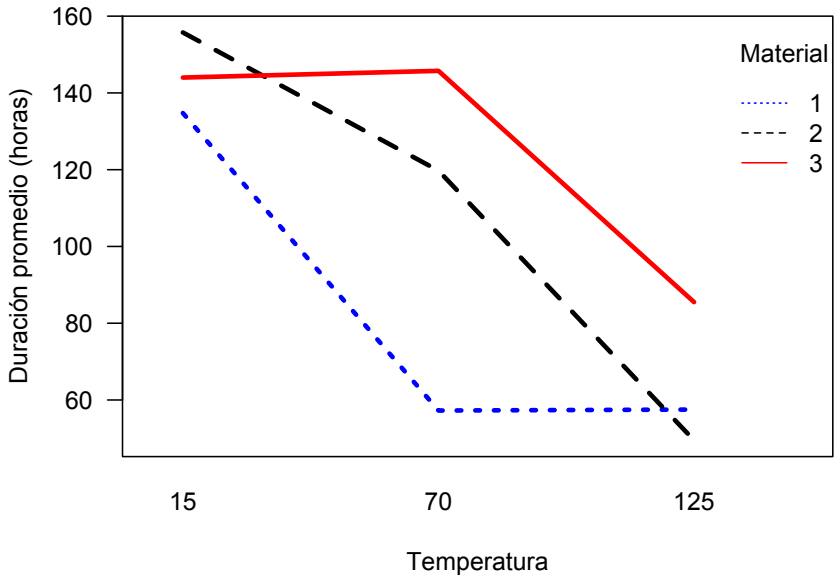

Figura 3.15: Gráfico de interacción entre Temperatura y Material sobre la duración promedio de las baterías.

3.7. Gráfico de espagueti

Los gráficos de espagueti son usados para representar la evolución de una variable medida para un grupo de sujetos en diferentes momentos del tiempo. La función `interaction.plot` se puede usar para obtener este tipo de gráficos, a continuación un ejemplo.

Ejemplo

El ejemplo aquí presentado fue tomado de este enlace[2]. El objetivo es crear un gráfico de espagueti para mostrar la evolución de la variable tolerancia a través del tiempo para cada uno de los 16 individuos estudiados. El código para descargar la base de datos y construir el gráfico se muestran a continuación.

```
dt <- read.table("http://www.ats.ucla.edu/stat/r/faq/tolpp.csv",
                        sep=",", header=T)

interaction.plot(response=dt$tolerance,
                 x.factor=dt$time, col=1:8, lwd=2,
                 trace.factor=dt$id, las=1, lty=1,
                 xlab="Tiempo", ylab="Tolerancia", legend=F)
```

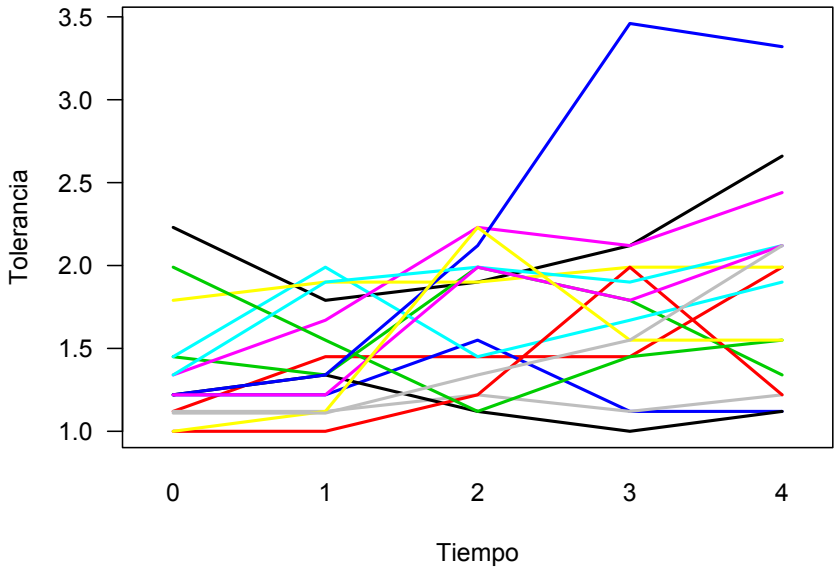

Figura 3.16: Gráfico de espagueti para ver la evolución de la variable tolerancia.

[2] http://stats.idre.ucla.edu/r/faq/how-can-i-make-spaghetti-plots/

4

Gráficos para variables cualitativas

En este capítulo se presentan funciones para la creación de gráficos para variables cualitativas.

4.1. Función `barplot`

Los gráficos de barras son útiles para representar las frecuencias absolutas o relativas asociadas a los niveles de una variable cualitativa y la función `barplot` se usa para obtener un gráfico de barras. La estructura de la función `barplot` con los argumentos más comunes de uso se muestra a continuación.

```
barplot(height, beside, horiz)
```

Los argumentos de la función `barplot` son:

- `height`: vector o matriz con la información de las frecuencias absolutas o relativas.
- `beside`: valor lógico para indicar si las barras deben estar al lado cuando la información ingresada es una matriz.
- `horiz`: valor lógico para indicar si el diagrama de barras debe ser horizontal, por defecto es `FALSE`.

La función `barplot` tiene otros parámetros que pueden ser consultados en la ayuda de la función por medio de la instrucción `?barplot`.

Ejemplo

Suponga que queremos construir un diagrama de barras para las frecuencias relativas de la variable estrato socioeconómico del apartamento de la base de datos sobre apartamentos usados en Medellín.

A continuación se muestra el código necesario para cargar la base de datos

`aptos2015`. Antes de construir el diagrama de barras solicitado es necesario construir la tabla de frecuencias para la variable estrato, para esto se usa la función `table` y los resultados se almacenan en el objeto `tabla1` que contiene las frecuencias **absolutas**. Para obtener las frecuencias **relativas** se usa luego la función `prop.table` sobre el objeto `tabla1`.

```
url <- 'https://raw.githubusercontent.com/fhernanb/datos/master/aptos2015'
datos <- read.table(file=url, header=T)
tabla1 <- table(datos$estrato)
tabla1 <- prop.table(tabla1)
tabla1
```

```
##
##       2       3       4       5       6
## 0.01153 0.23199 0.19885 0.20893 0.34870
```

Una vez se tiene el objeto con la información de las frecuencias relativas se puede dibujar el diagrama de barras usando el siguiente código.

```
barplot(tabla1, xlab='Estrato socioeconómico',
        ylab='Frecuencia relativa', las=1)
```

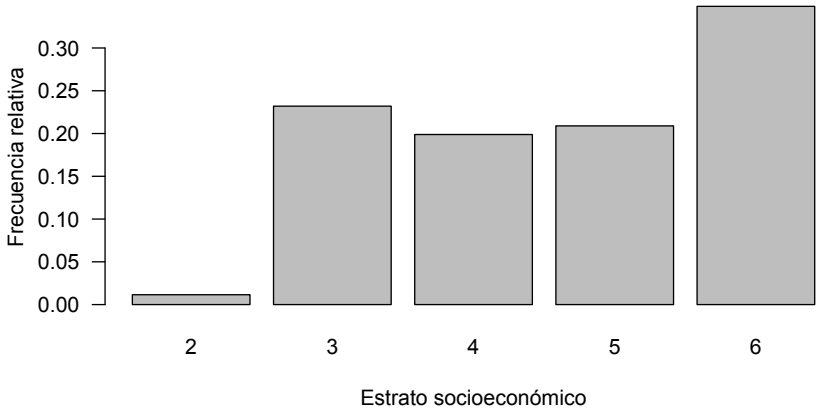

Figura 4.1: Diagrama de barras para el estrato socioeconómico de los apartamentos usados.

En la Figura 4.1 se presenta el diagrama de barras solicitado. Se observa que hay pocos apartamentos (1.15 %) pertenecientes al estrato dos, los estratos

tres, cuatro y cinco aportan porcentajes similares a la base de datos y que el estrato 6 es el que más apartamentos aporta a la base de datos, 34.87 %.

Algunas veces se acostumbra a colocar las frecuencias relativas sobre la parte superior de las barras para facilitar la lectura. A continuación se presenta el código para replicar la Figura 4.1 con las frecuencias relativas. Lo primero que se hace es dibujar el diagrama de barras y almacenar la información de él en el objeto `xx` para luego poder usar la ubicación de cada una de las barras. Note que se agregó también `ylim=c(0, 0.45)` para conseguir una ampliación del eje vertical, esto para lograr que se vea el número sobre la barra del estrato 6. Luego se usa la función `text` para incluir un texto en las coordenadas `x=xx` y `y=tabla1`, el parámetro `pos=3` coloca el texto en la parte superior de las coordenadas y el parámetro `label` sirve para indicar lo que se desea escribir en las coordenadas indicadas, en este caso son las frecuencias relativas almacenadas en `tabla1`.

```
xx <- barplot(tabla1, ylim=c(0, 0.45), col=gray(0.9),
              xlab='Estrato socioeconómico',
              ylab='Frecuencia relativa', las=1)

text(x=xx, y=tabla1, pos=3, cex=0.8, col="red",
     label=round(tabla1, 4))
```

Figura 4.2: Diagrama de barras para el estrato socioeconómico de los apartamentos usados con las frecuencias relativas sobre las barras.

En la Figura 4.2 se muestra el diagrama de barras modificado. Note que si no

se hubiese usado `ylim=c(0, 0.45)` al dibujar el diagrama, la marca sobre la última barra no se podría ver.

Ejemplo

Suponga que queremos construir un diagrama de barras para comparar la variable presencia de parqueadero con el estrato socioeconómico en la base de datos sobre apartamentos usados en Medellín.

La función `barplot` también puede ser usada para representar una tabla de frecuencia con dos variables. Para obtener la tabla de frecuencia para relacionar parqueadero con estrato se usa el siguiente código.

```
tabla2 <- table(datos$parqueadero, datos$estrato)
tabla2
```

```
##
##        2   3   4   5   6
##   no   5  88  24   8   1
##   si   3  73 114 137 241
```

El anterior resultado es la tabla de contingencia entre las variables parqueadero y estrato, de esta tabla vemos que para estratos superiores el número de apartamentos que si tienen parqueadero es mayor que los apartamentos sin parqueadero. La tabla anterior se puede representar gráficamente usando el siguiente código.

```
barplot(tabla2)
```

En la Figura 4.3 se muestra el gráfico de barras sin editar, el color negro representa la frecuencia de los apartamentos sin parqueadero (no) y el color gris representa los apartamentos con parqueadero (si), las barras están una encima de la otra y la comparación no es tan clara como debería. Para mejorar la comparación se usa el parámetro `besides=TRUE`, a continuación el código utilizado.

```
barplot(tabla2, beside=TRUE)
```

En la Figura 4.4 está el gráfico de barras obtenido agregando `besides=TRUE` para que las barras se ubiquen una junto a la otra. Este gráfico se puede mejorar aún más colocando una leyenda para identificar las barras, nombrando los ejes y usando otros colores, a continuación el código utilizado.

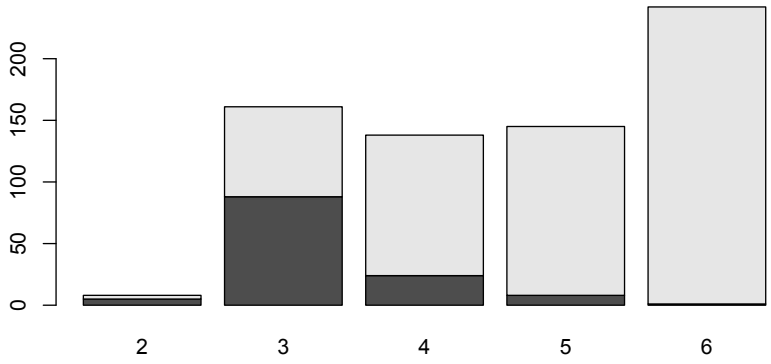

Figura 4.3: Diagrama I de barras la relación entre parqueadero y estrato.

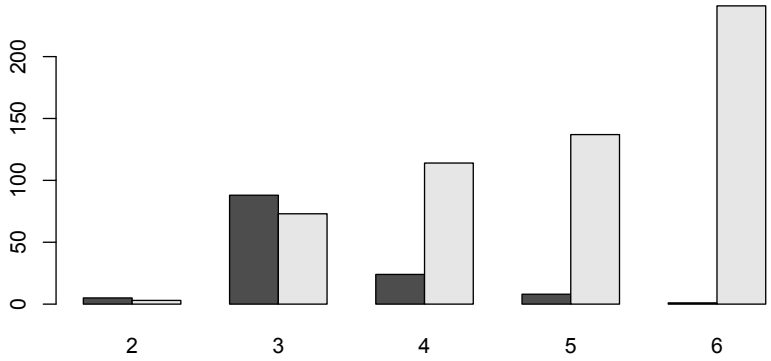

Figura 4.4: Diagrama II de barras la relación entre parqueadero y estrato.

```
barplot(tabla2, beside = TRUE, las=1,
        xlab='Estrato', ylab='Frecuencia',
        col = c("lightblue", "mistyrose"),
        ylim = c(0, 250))
legend('topleft', legend=rownames(tabla2), bty='n',
       fill=c("lightblue", "mistyrose"))
```

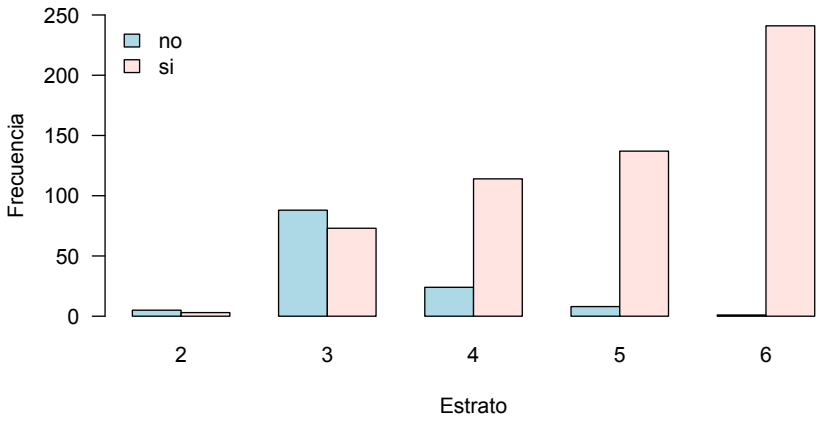

Figura 4.5: Relación entre la presencia de parqueadero y el estrato socioeconómico.

En la Figura 4.5 se observa el gráfico de barras solicitado, se observa claramente que en los estratos 4, 5 y 6 predominan los aparatamentos con parqueadero.

Es posible construir una tabla de contingencia de frecuencia relativa para ver cómo es el comportamiento de tener o no parquedadero dentro de cada estrato, el siguiente código construye la `tabla3` con la información necesaria. La función `prop.table` permite obtener la tabla de frecuencias **relativas** a partir de una tabla de frecuencias absolutas, el parámetro `margin` sirve para indicar si las frecuencias **relativas** se deben obtener por fila (`margin=1`) o por columnas (`margin=2`).

```
tabla3 <- prop.table(tabla2, margin=2)
tabla3
```

```
##
##              2         3         4         5         6
```

```
##    no 0.625000 0.546584 0.173913 0.055172 0.004132
##    si 0.375000 0.453416 0.826087 0.944828 0.995868
```

De la anterior tabla se ve que el porcentaje de apartamentos con parqueadero supera enormemente el los apartamentos sin parqueadero para los estratos 6, 5 y 4. El código para generar un gráfico asociado a la `tabla3` se muestra a continuación.

```
barplot(tabla3,
        beside = TRUE, las=1,
        xlab='Estrato', ylab='Frecuencia relativa',
        col = c("lightblue", "mistyrose"),
        ylim = c(0, 1))
legend('topleft', legend=rownames(tabla2), bty='n',
       fill=c("lightblue", "mistyrose"))
```

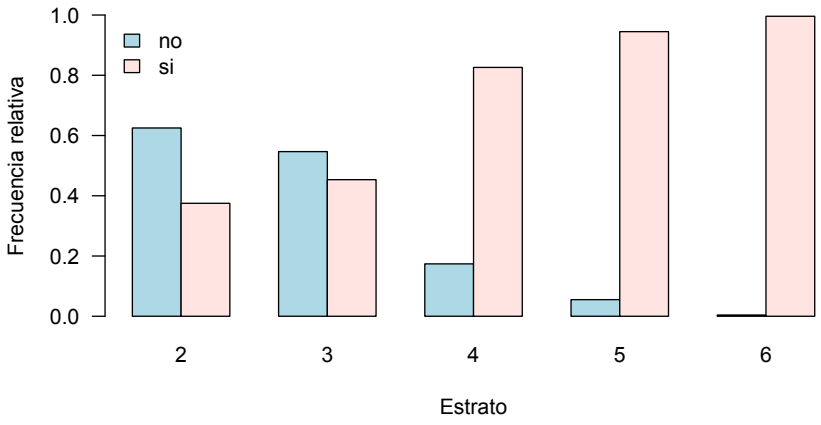

Figura 4.6: Relación entre la presencia de parqueadero y el estrato socioeconómico.

¿Cuáles son las ventajas y/o desventajas de las figuras 4.5 y 4.6 al ser presentadas en un informe?

4.2. Función `pie`

En R es posible construir gráficos de pastel para representar una tabla de frecuencia relativa o absoluta, sin embargo este tipo de gráficos no es recomendable y en la ayuda de la función se hace la siguiente advertencia:

> Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.

La estructura de la función `pie` con los argumentos más comunes de uso se muestra a continuación.

```
pie(x, labels)
```

Los argumentos de la función `pie` son:

- `x`: vector con elementos no negativos que representan las frecuencias de los niveles de la variable cualitativa.
- `labels`: vector con los nombres a colocar en cada parte del pastel, por defecto se usan los nombres del vector `x`.

Ejemplo

Dibujar un gráfico de pastel para las frecuencias relativas de la variable estrato socioeconómico del apartamento de la base de datos sobre apartamentos usados en Medellín.

La `tabla1` construída en el primer ejemplo de `barplot` se utiliza para construir el gráfico solicitado. Abajo el código necesario para construir el gráfico.

```
nombres <- paste('Estrato ', 2:6)
pie(x=tabla1, labels=nombres,
    main='Gráfico de pastel NO recomendado!!!')
```

La Figura 4.7 presenta el gráfico de pastel construído con la instrucción anterior.

Figura 4.7: Gráfico de pastel para las frecuencias relativas del estrato socioeconómico.

EJERCICIOS

Use funciones o procedimientos (varias líneas) de R para responder cada una de las siguientes preguntas.

Todas las preguntas siguientes están relacionadas con la base de datos sobre apartamentos[1] usados en la ciudad de Medellín.

1. Construya un diagrama de barras para representar las frecuencias ABSOLUTAS de la variable ubicación.
2. Vuelva a construir el mismo diagrama de barras anterior pero de forma horizontal y agregando números de color azul para indicar las frecuencias.
3. Construya una tabla de dos vías para las variables ubicación y parqueadero.
4. Construya una tabla de frecuencias relativas para ver cómo se comporta la variable parqueadero dentro de cada ubicación.
5. Dibuje un diagrama de barras para la tabla de frecuencias del punto anterior.

[1] https://raw.githubusercontent.com/fhernanb/datos/master/aptos2015

5

Función `par`

En este capítulo se presentan las posibilidades que ofrece la función `par` para la elaboración de gráficos. La función `par` tiene 72 parámetros y a continuación se muestran.

```
par(xlog, ylog, adj, ann, ask, bg, bty, cex, cex.axis, cex.lab,
    cex.main, cex.sub, cin, col, col.axis, col.lab, col.main,
    col.sub, cra, crt, csi, cxy, din, err, family, fg, fig, fin,
    font, font.axis, font.lab, font.main, font.sub, lab, las,
    lend, lheight, ljoin, lmitre, lty, lwd, mai, mar, mex, mfcol,
    mfg, mfrow, mgp, mkh, new, oma, omd, omi, page, pch, pin,
    plt, ps, pty, smo, srt, tck, tcl, usr, xaxp, xaxs, xaxt, xpd,
    yaxp, yaxs, yaxt, ylbias)
```

Para conocer los valores que tienen por defecto cada uno de estos parámetros se puede utilizar el siguiente código.

```
par()
```

Al ejecutar el código anterior se obtendrá una lista con 72 objetos en la cual se tendrán los valores que cada uno de los parámetros asume inicialmente en una sesión de R. Luego de modificar uno o alguno de los parámetros de la función `par`, todos los gráficos que se construyan de ahí en adelante estarán afectados por el cambio realizado.

Una buena práctica para retornar a los valores iniciales del objeto `par()` es cerrar la ventana gráfica.

La utilidad de cada uno de los parámetros para personalizar los gráficos se mostrará por medio de ejemplos.

5.1. Parámetro ann

Este parámetro sirve para indicar si se quiere un gráfico con nombres en los ejes y título principal, por defecto asume el valor `TRUE`.

Ejemplo

Dibujar la densidad para una distribución χ^2_1 usando `ann=FALSE` dentro de la función `par`.

El código necesario para obtener la figura se muestra a continuación. Note que a pesar de haber solicitado título principal y nombres de los ejes, éstos no aparecen en el resultado final porque se usó antes `par(ann=FALSE)`. En la Figura 5.1 se muestra el gráfico solicitado.

```
par(ann=FALSE)
curve(dchisq(x, df=1), col='salmon1', lwd=4,
      main='Chi cuadrada',
      xlab='x', ylab='Densidad')
```

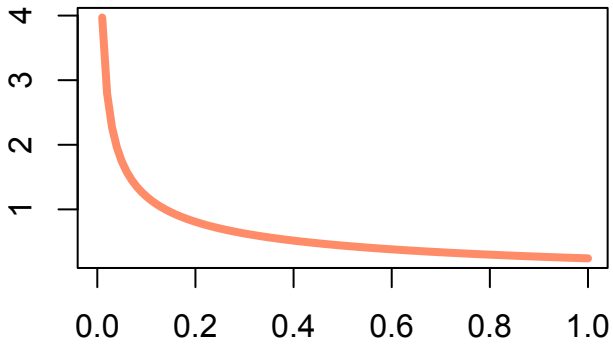

Figura 5.1: Efecto del parámetro `ann`.

La opción `ann=FALSE` es muy útil para gráficos sin nombres en los ejes y sin título principal. Esta opción evita el uso de `xlab='', ylab='', main=''` dentro de las funciones gráficas.

5.2. Parámetro `adj`

Este parámetro sirve para modificar la justificación del texto cuando se usan las funciones `text`, `mtext` y `title`. Su valor por defecto es 0.5 que indica que el texto debe quedar centrado en las coordenadas x e y indicadas, un valor de 0 significa justificación a izquierda mientras que 1 significa justificación a derecha.

Ejemplo

Dibujar un gráfico vacío, ubicar las palabra *hola mi mundo* horizontalmente en el gráfico y explorar el efecto del parámetro `adj`.

El código necesario para obtener lo solicitado se muestra a continuación. Se agregaron líneas a trazos de color azul para indicar el sitio exacto donde se quería el texto. Observando la Figura 5.2 se nota claramente que cuando `adj=0.5` la palabra queda centrada mientras que con otros valores cambia la justificación del texto.

```
plot(NULL, xlim=c(0, 0.6), ylim=c(0, 1))

par(adj=0)  # Para justificar a izquierda
text(x=0.1, y=0.6, 'hola', cex=3)

par(adj=0.5)  # Para justificar centrado
text(x=0.3, y=0.6, 'mi', cex=3)

par(adj=1)  # Para justificar a derecha
text(x=0.5, y=0.6, 'mundo', cex=3)

abline(h=0.6, v=c(0.1, 0.3, 0.5),
       lty=3, col='deepskyblue3')
```

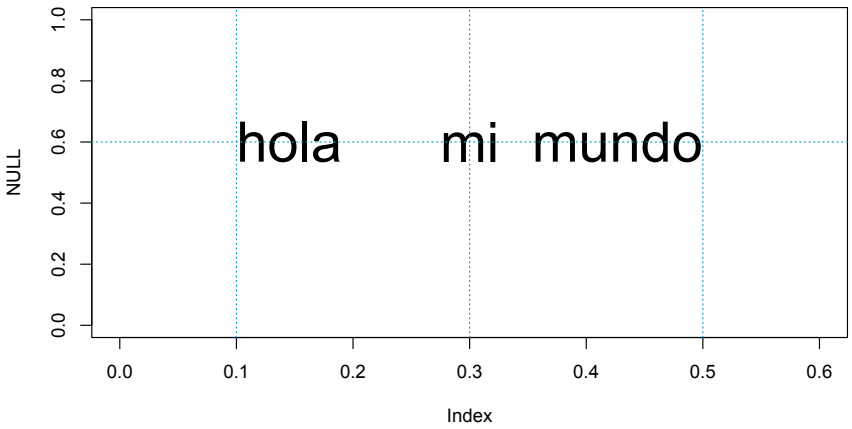

Figura 5.2: Efecto del parámetro `adj`.

5.3. Parámetro `bg`

Este parámetro sirve para modificar el color del fondo donde se va a dibujar.

Ejemplo

Dibujar la densidad de una normal estándar usando un color verde para el fondo.

A continuación el código necesario para modficar el color del fondo. En esta página[1] se puede encontrar una paleta de colores disponibles en R. En la Figura 5.3 se muestra el resultado, el color de fondo está por toda la ventana gráfica.

```
par(bg='darkseagreen1')
curve(dnorm, lwd=6, col='blue', xlim=c(-4, 4))
```

[1] http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf

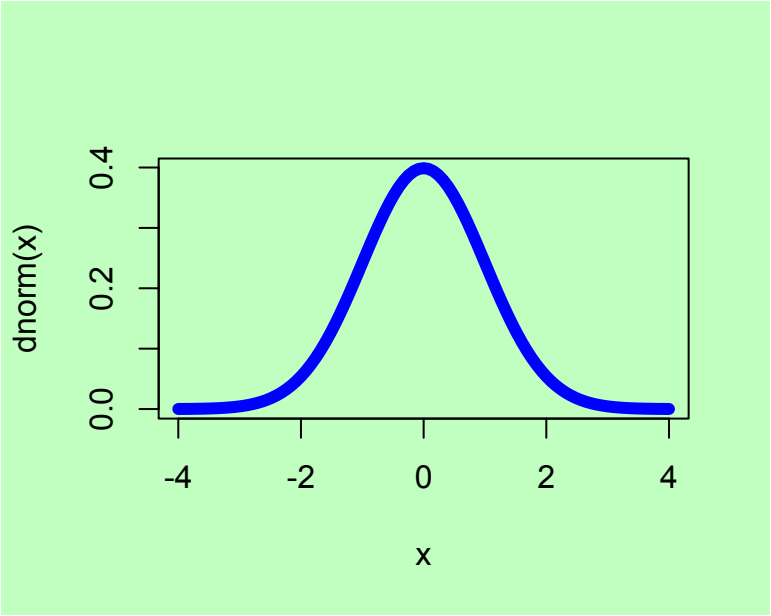

Figura 5.3: Efecto del parámetro `bg`.

5.4. Parámetro `mfrow`

Este parámetro sirve para dividir la ventana gráfica en forma de matriz para almacenar en cada celda un gráfico diferente. La forma para modificar este parámetro es `mfrow=c(nf, nc)`, donde `nf` es el número de filas y `nc` el número de columnas en las cuales se va a dividir la ventana gráfica.

Ejemplo

Dibujar la distribución de probabilidad para una distribución Poisson con $\lambda = 1, 3, 7, 15$.

Note que el objetivo es repetir el mismo gráfico para 4 valores diferentes del paramétro λ, por lo tanto se escribirá una sola vez el código de interés pero se repetirá automáticamente 4 veces para cada valor de λ. Para construir el gráfico solicitado iniciamos partiendo la ventana gráfica en una matriz de 2×2 usando `par(mfrow=c(2, 2))`. Luego se construye el vector `lambdas` con los valores de λ y por último se coloca el `plot` de interés dentro de una sentencia `for`. Abajo el código utilizado.

```
par(mfrow=c(2, 2))

lambdas <- c(1, 3, 7, 15)

for (i in 1:4) {
  plot(dpois(x=0:30, lambda=lambdas[i]), lwd=4,
       type='h', xlab='x', ylab='Probabilidad')
  title(bquote(~ lambda == .(lambdas[i])))
}
```

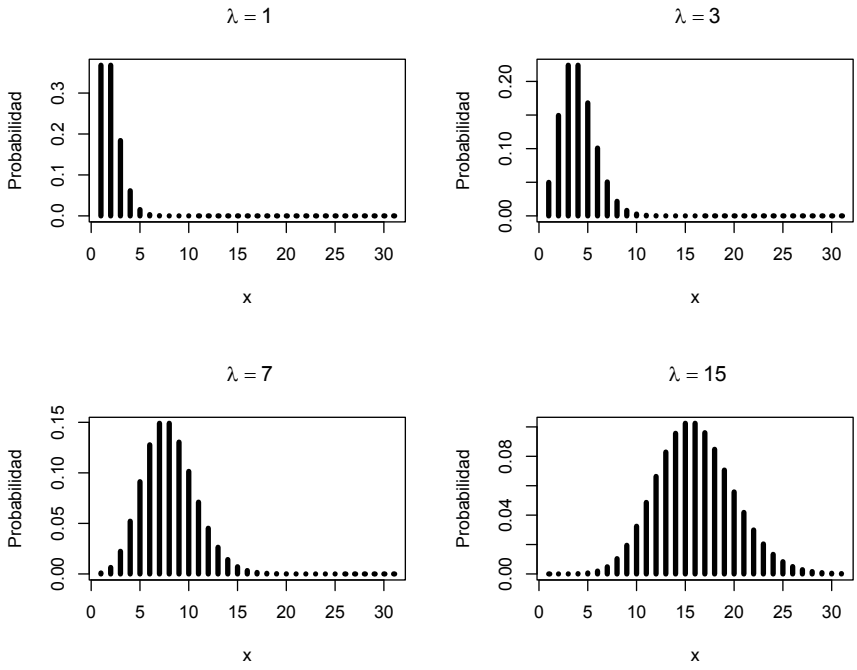

Figura 5.4: Efecto del parámetro `mfrow`.

En la Figura 5.4 se muestra la distribución de probabilidad para cada uno de los 4 valores de λ en la misma figura.

La función `bquote` sirve para construir mensajes que sean una mezcla de texto, expresiones matemáticas y valores de un objeto.

5.5. Parámetro `bty`

Este parámetro sirve para modficar la caja alrededor del gráfico construído. Los posibles valores para este parámetro son: `'o'` valor por defecto para obtener la caja usual; se pueden usar también los símboloes `'l'`, `'7'`, `'c'`, `'u'` o `']'`, el resultado será una caja con la forma del símbolo; se puede usar `'n'` para suprimir la caja.

Ejemplo

Dibujar la densidad de una χ^2_5 modificando el parámetro `bty` de `par`.

A continuación el código para dibujar la misma densidad 4 veces modificando el valor para `bty`. En la Figura 5.5 se muestra el efecto que tienen los valores elegidos. Observe que, además de los ejes, hay una caja cuya forma depende el valor seleccionado, cuando `bty='u'` el resultado es una caja con esa misma forma.

```
par(bty='n', mfrow=c(2, 2))
curve(dchisq(x, df=5), xlim=c(0, 20))
title("Usando bty='n'")
par(bty='o')
curve(dchisq(x, df=5), xlim=c(0, 20))
title("Usando bty='o'")
par(bty='u')
curve(dchisq(x, df=5), xlim=c(0, 20))
title("Usando bty='u'")
par(bty='l')
curve(dchisq(x, df=5), xlim=c(0, 20))
title("Usando bty='l'")
```

5.6. Parámetro `cex`

Este parámetro sirve para modificar el tamaño de los símbolos, nombres de los ejes, marcas de los ejes y títulos. El valor por defecto es `cex=1`, valores mayores a uno generan gráficos con símbolos, títulos y marcas grandes, incluso desproporcionados; valores pequeños de este parámetro permiten obtener

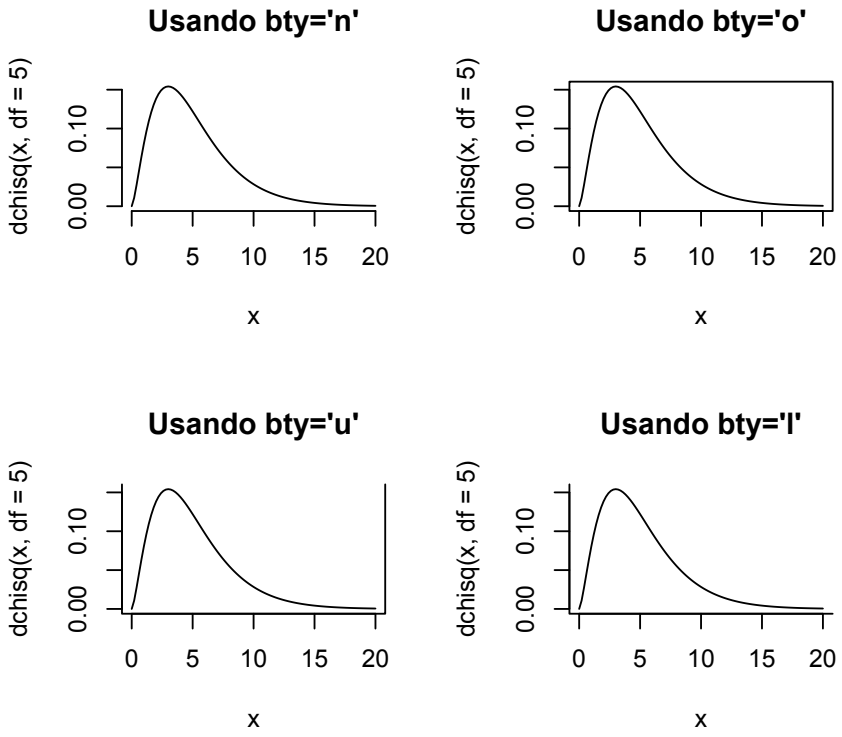

Figura 5.5: Efecto del parámetro `bty`.

mejores figuras. A continuación un ejemplo para ver el efecto que tiene este parámetro sobre las figuras obtenidas.

Ejemplo

Hacer un gráfico de dispersión sencillo para explorar el efecto del parámetro `cex`.

Se construirá un gráfico de dispersión dos veces, en la primera con parámetro `cex=1` y en la segunda ocasión usando `cex=0.6`. Abajo el código utilizado. En el pánel izquierdo de la Figura 5.6 está el diagrama de dispersión obtenido con `cex=1.4` mientras que el de la derecha fue obtenido con `cex=0.6`. Observe cómo se modificaron los nombres de los ejes, título principal, marcas de los ejes y tamaño del símbolo.

```
x <- -3:3
par(mfrow=c(1, 2), cex=1.4)
plot(x=x, y=x, main='Usando \n cex=1.4')

par(cex=0.6)
plot(x=x, y=x, main='Usando \n cex=0.6')
```

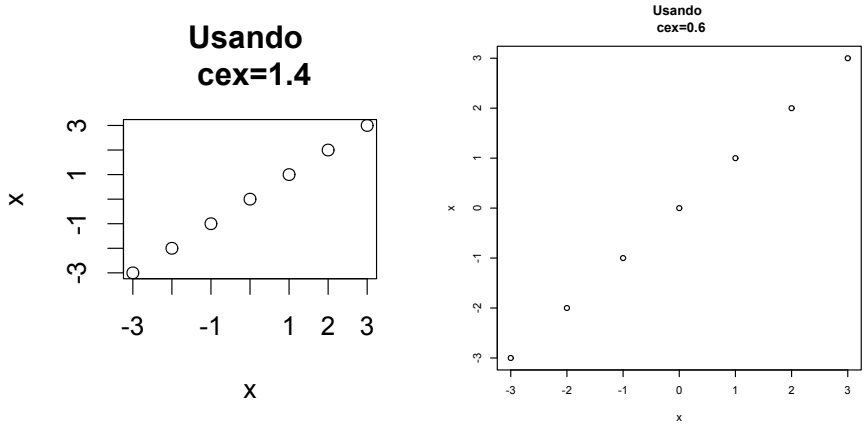

Figura 5.6: Efecto del parámetro `cex`.

El parámetro `cex` tiene un efecto importante en las figuras, se recomienda al usuario que cada vez que construya una figura explore diferentes opciones para obtener una figura que exprese la esencia de lo que desea comunicar.

5.7. Parámetros `cex.axis`, `cex.lab`, `cex.main` y `cex.sub`

Los parámetros `cex.axis`, `cex.lab`, `cex.main` y `cex.sub` sirven para modificar el tamaño de las marcas en los ejes, el tamaño de los nombres de los ejes, el tamaño del título principal y el tamaño del subtítulo respectivamente. El parámetro `cex` explicado en la sección anterior, modifica el tamaño de los anteriores elementos simultáneamente mientras que los parámetros explicados en esta sección modifican cada uno de los elementos por aparte.

Ejemplo

Hacer un gráfico cualquiera y modificar los parámetros `cex.axis`, `cex.lab`, `cex.main` y `cex.sub` de la función `par` con el objetivo de ver el efecto que ellos tiene sobre el gráfico.

A continuación se construye un gráfico simple y se modifican todos los parámetros `cex.algo` para ver el efecto que ellos tienen sobre el gráfico resultante, abajo el código usado.

En la Figura 5.7 se muestra el resultado. Observe que las marcas y números en los ejes quedaron grandes debido a que se usó `cex.axis=1.8`, los nombres que identifican los ejes casi no se ven porque se usó `cex.lab=0.3`. El título principal quedó muy pequeño debido a que `cex.main=0.7` mientras que el subtítulo quedó demasiado grande porque se usó `cex.sub=2`. Este ejemplo muestra que el usuario tiene todo el control para construir gráficos personalizados en R.

```
x <- -3:3
par(cex.axis=1.8, cex.lab=0.3, cex.main=0.7, cex.sub=2.0)
plot(x=x, y=x,
     xlab='Valores de x', ylab='Valores de y',
     main='Título principal',
     sub='Subtítulo')
```

5.8. Parámetro `col`, `col.axis`, `col.lab`, `col.main` y `col.sub`

Estos parámetros sirven para definir el color a usar en los ejes, en los nombres de los ejes, en el título y en el subtítulo.

Ejemplo

Hacer un gráfico cualquiera y modificar los parámetros `col`, `col.axis`, `col.lab`, `col.main` y `col.sub` de la función `par` con el objetivo de ver el efecto que ellos tiene sobre el gráfico.

A continuación se construye un gráfico simple y se modifican todos los parámetros para ver el efecto que ellos tienen sobre el gráfico resultante, abajo el código usado.

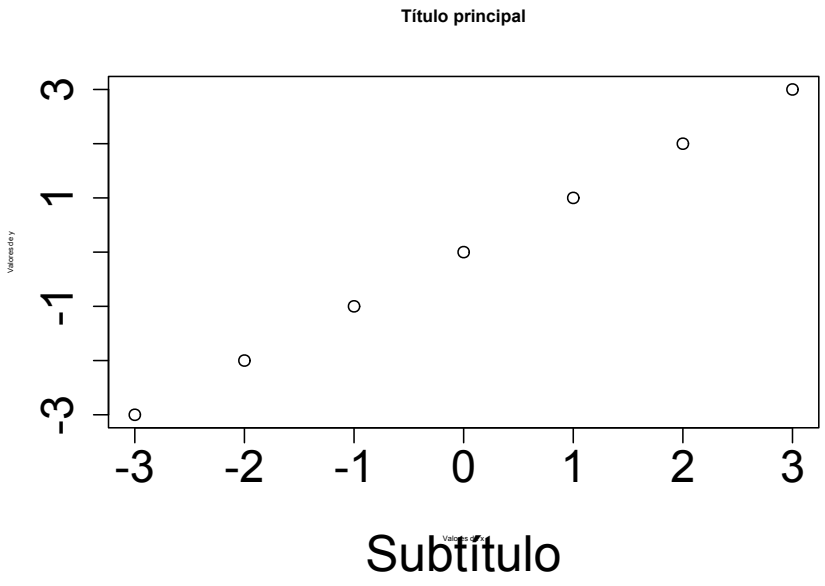

Figura 5.7: Efecto de los parámetros `cex.axis`, `cex.lab`, `cex.main` y `cex.sub`.

```
par(col='blue', col.axis='red', col.lab='orange',
    col.main='darkgreen', col.sub='purple', bty='n')
plot(1:10, pch=20, cex=2, las=1,
     main='Título', xlab='Nombre eje X',
     ylab='Nombre eje Y', sub='Subtítulo')
```

En la Figura 5.8 se muestra el efecto de los parámetros `col`, `col.axis`, `col.lab`, `col.main` y `col.sub` de la función `par`.

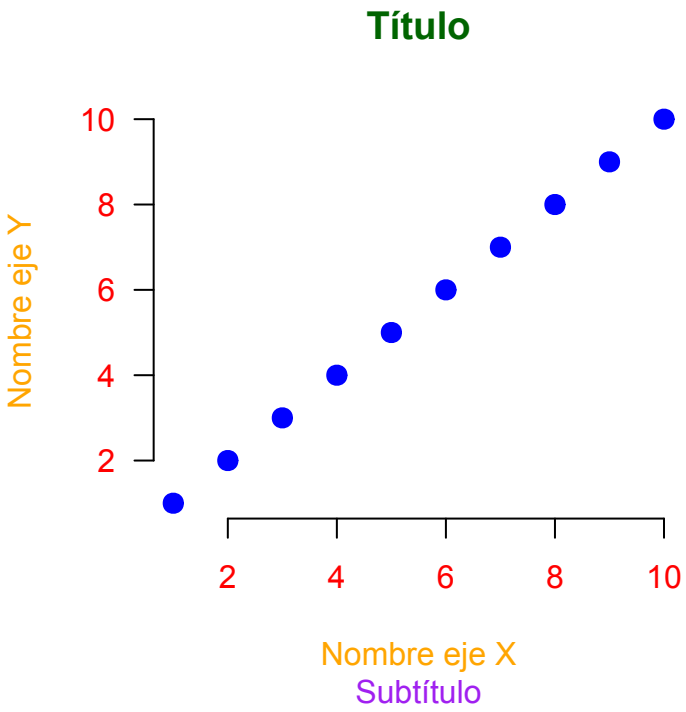

Figura 5.8: Efecto de los parámetros `col`, `col.axis`, `col.lab`, `col.main` y `col.sub`.

6

Funciones auxiliares

En este capítulo se presentan funciones auxiliares que son útiles para complementar los gráficos generados en R.

6.1. Función `segments`

Esta función es muy útil para digujar segmentos. La estructura de la función se muestra a continuación.

```
segments(x0, y0, x1 = x0, y1 = y0, ...)
```

Los argumentos de la función son:

- `xo, yo`: coordenadas del punto de inicio del segmento.
- `x1, y1`: coordenadas del punto de fin del segmento.
- `...`: otros parámetros gráficos.

A continuación de muestra el código para dibujar con segmentos la sigla de la Universidad Nacional de Colombia (UN), en la Figura 6.1 se puede ver el resultado.

```
plot(c(-11, 3), c(-10, 10), type="n", xlab="", ylab="")
grid()

segments(-9, 9, -9, -9, lwd=4, col=3)
segments(-9, -9, -5, -9, lwd=4, col=3)
segments(-5, -9, -5, 9, lwd=4, col=3)

segments(-4, -9, -4, 9, lwd=4)
segments(-4, 9, 0, -9, lwd=4)
segments(0, -9, 0, 9, lwd=4)
```

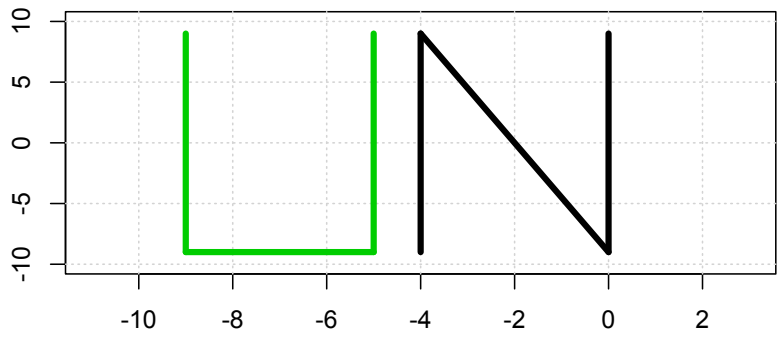

Figura 6.1: Ejemplos de segmentos.

6.2. Función `rect`

Esta función es muy útil para digujar rectángulos. La estructura de la función se muestra a continuación.

```
rect(xleft, ybottom, xright, ytop,
     density = NULL, angle = 45, ...)
```

Los argumentos de la función son:

- `xleft`: vector o escalar con la posición de x a izquierda.
- `ybottom`: vector o escalar con la posición de y abajo.
- `xright`: vector o escalar con la posición de x a derecha.
- `ytop`: vector o escalar con la posición de y arriba.
- `density`: número de líneas por pulgada con la cuales se rellenará el rectángulo.
- `angle`: ángulo de inclinación de la líneas de relleno.
- `col`: color para el fondo del rectángulo.
- `border`: color para el borde del rectángulo, un valor posible es `'transparent'` cuando no se desea borde.
- `...`: otros parámetros gráficos.

En la Figura 6.2 se muestran 7 rectángulos de ejemplo que fueron obtenidos

al variar los parámetros de la función `rect`, a continuación el código usado para obtener esa figura.

```
plot(NA, xlim=c(10, 20), ylim=c(10, 30), las=1, xlab='', ylab='')
rect(10, 28, 12, 30)
rect(11, 25, 13, 27, col='red')
rect(12, 22, 14, 24, density=5)
rect(13, 19, 15, 21, density=5, angle=15)
rect(14, 16, 16, 18, col='pink', border='blue')
rect(15, 13, 17, 15, lty='dashed')
rect(16, 10, 18, 12, lwd=3)

text(14, 26, "col='red'")
text(15, 23, "density=5")
text(17, 20, "density=5, angle=15")
text(18.2, 17, "col='pink', border='blue'")
text(18.2, 14, "lty='dashed'")
text(18.8, 11, "lwd=3")
```

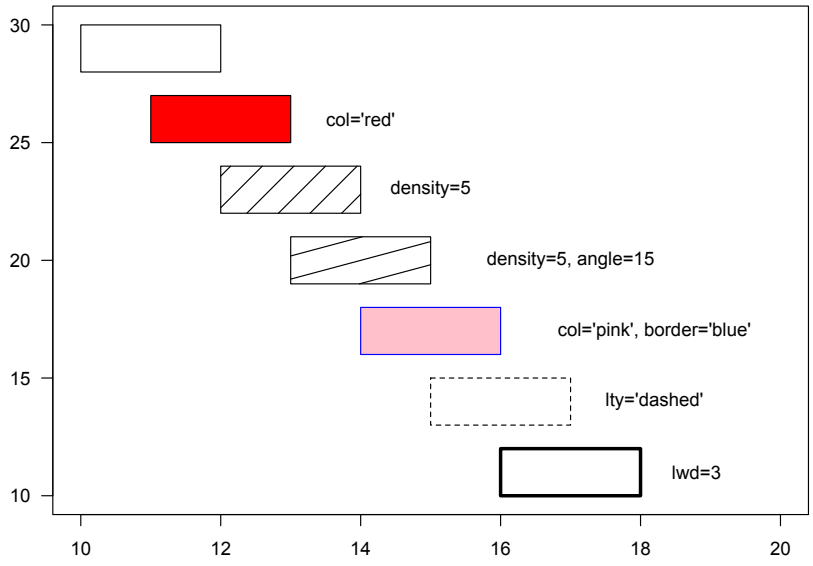

Figura 6.2: Ejemplos de rectángulos.

El código mostrado abajo pertenece a la ayuda de la función `rect` y el resultado es la Figura 6.3. De esta figura se observa que es posible dibujar varios rectángulos ingresando las coordenadas como vectores.

```
## set up the plot region:
plot(c(100, 250), c(300, 450), type="n", xlab="", ylab="",
     main="2 x 11 rectangles; 'rect(100+i,300+i, 150+i,380+i)'")
i <- 4*(0:10)
## draw rectangles with bottom left (100, 300)+i
## and top right (150, 380)+i
rect(100+i, 300+i, 150+i, 380+i, col=rainbow(11, start=0.7, end=0.1))
rect(240-i, 320+i, 250-i, 410+i, col=heat.colors(11), lwd=i/5)
```

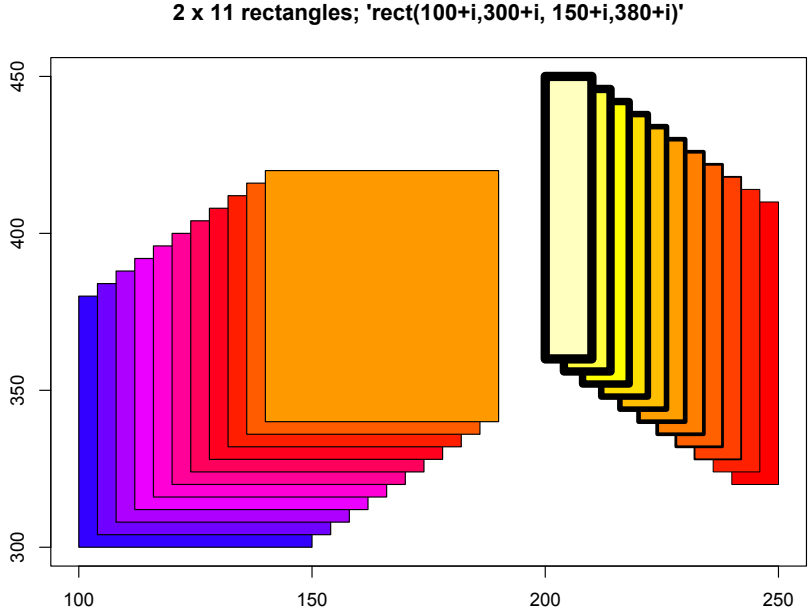

Figura 6.3: Ejemplo de rectángulos obtenidos con `rect`.

6.3. Función `polygon`

Esta función es muy útil para digujar polígonos. La estructura de la función se muestra a continuación.

```
polygon(x, y, density = NULL, angle = 45,
        border, col, lty, ...)
```

Los argumentos de la función son:

- `x, y`: vectores con las coordenadas de ubicación de los puntos que forman el polígono.
- `density`: número de líneas por pulgada con la cuales se rellenará el polígono.
- `angle`: ángulo de inclinación de la líneas de relleno.
- `border`: color para el borde del polígono, un valor posible es `'transparent'` cuando no se desea borde.
- `col`: color para el fondo del rectángulo.
- `lty`: tipo de línea a usar para el borde.
- `...`: otros parámetros gráficos.

A continuación se muestra como usar la función `polygon` para dibujar un triángulo y un cuadrilátero. En la Figura 6.4 se muestran las figuras obtenidas.

```
plot(0:5, 0:5, type="n", xlab="", ylab="")
abline(v=0:5, h=0:5, col=gray(0.8), lty='dashed')

polygon(x=c(0, 1, 2), y=c(1, 5, 2),          # Para el triangulo
        col='blue', border='red', lwd=4)

polygon(x=c(3, 5, 5, 2), y=c(5, 4, 1, 1), # Para el cuadrilatero
        col='orange', border='darkgreen', lwd=4)
```

6.4. Para dibujar un círculo

En la base de R no hay una función específica para crear círculos, sin embargo, es posible usar unas pocas líneas para obtener el círculo con un radio y centro deseado por medio de la función `polygon`. A continuación se muestra el código para dibujar un círculo de radio 7 unidades con centro en el punto (-2, 1) con

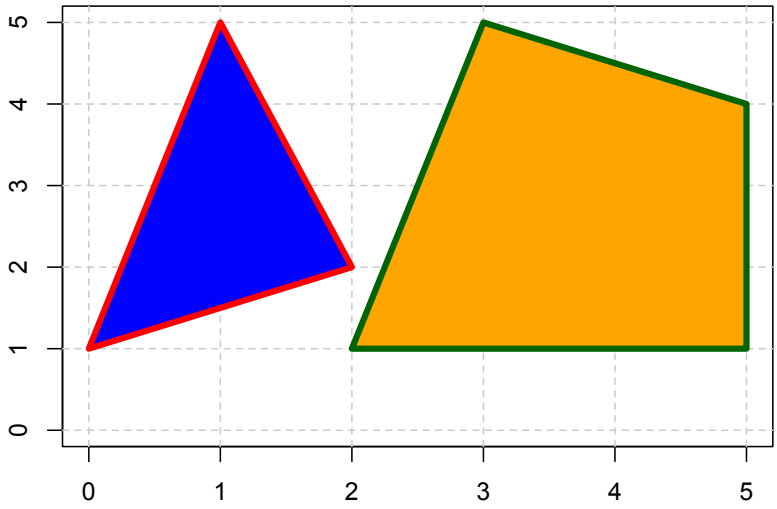

Figura 6.4: Ejemplo de formas obtenidas con `polygon`.

fondo azul claro y borde de color azul obscuro. En la Figura 6.5 se muestra el círculo solicitado.

```
rad     <- 7   # Valor del radio
xcenter <- -2  # Coordenada en x del centro
ycenter <- 1   # Coordenada en y del centro

plot(c(-10, 10), c(-10, 10), type="n", xlab="", ylab="")
theta <- seq(0, 2 * pi, length = 200)
polygon(x=rad * cos(theta) + xcenter, y=rad * sin(theta) + ycenter,
      lwd=3, col='steelblue1', border='steelblue4')
grid()  # Para incluir una cuadrícula de guía
```

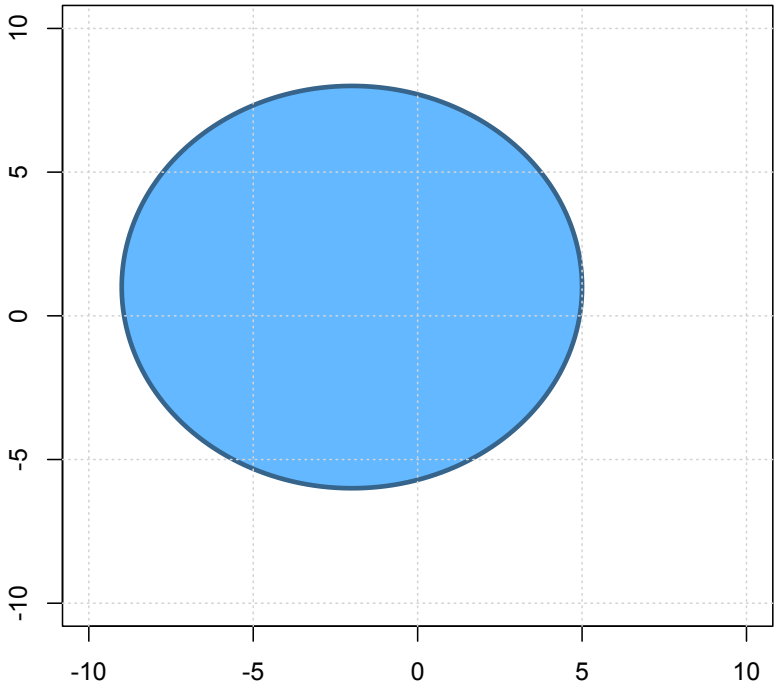

Figura 6.5: Ejemplo de un círculo.

6.5. Función arrows

Esta función es muy útil para incluir flechas en una figura. La estructura de la función se muestra a continuación.

```
arrows(x0, y0, x1=x0, y1=y0, length=0.25, angle=30, ...)
```

Los argumentos de la función son:

- `x0, y0`: coordenadas de inicio de la flecha.
- `x1, y1`: coordenadas de fin de la flecha.

- `length`: longitud (en pulgadas) de los bordes de punta de la flecha, el valor por defecto es 0.25.
- `angle`: ángulo para la punta de la flecha, a mayor ángulo más abiertas las puntas.
- `code`: número entero para indicar el tipo de flecha a dibujar. El valor de 0 para una flecha sin punta (entonces no sería una flecha), el valor de 1 para una flecha apuntando al revés (no es frecuente), el valor de 2 (valor por defecto) para una flecha usual y valor de 3 para colocar puntas a ambos lados de la flecha. El valor por defecto son 30 grados.
- `...`: otros parámetros gráficos.

En la Figura 6.6 se muestran los cuatro tipos de flechas que se obtienen al variar el parámetro `code` de la función `arrows`. Note que la flecha de la derecha fue construída subiendo, pero como `code=1` ella quedó bajando.

```
plot(NA, xlab='X', ylab='Y',      # Para crear un gráfico vacío
     xlim=c(0, 6), ylim=c(0, 5))
arrows(x0=2, y0=1, x1=4, y1=1, code=0)
arrows(x0=5, y0=1, x1=5, y1=4, code=1)
arrows(x0=2, y0=4, x1=4, y1=4, code=2)
arrows(x0=1, y0=1, x1=1, y1=4, code=3)
text(3, 0.8, 'code=0')
text(5.4, 2.5, 'code=1')
text(3, 4.2, 'code=2')
text(0.6, 2.5, 'code=3')
```

En la Figura 6.7 se muestran 4 flechas para varios valores del parámetro `angle`, de la figura se observa que entre mayor el ángulo, la punta de la fecha es más achatada.

```
plot(NA, xlab='X', ylab='Y',      # Para crear un gráfico vacío
     xlim=c(0, 6), ylim=c(0, 5))

arrows(x0=1, y0=4, x1=4, y1=4)
arrows(x0=1, y0=3, x1=4, y1=3, angle=45)
arrows(x0=1, y0=2, x1=4, y1=2, angle=60)
arrows(x0=1, y0=1, x1=4, y1=1, angle=90)
text(4.5, 3, 'angle=45')
text(4.5, 2, 'angle=60')
text(4.5, 1, 'angle=90')
```

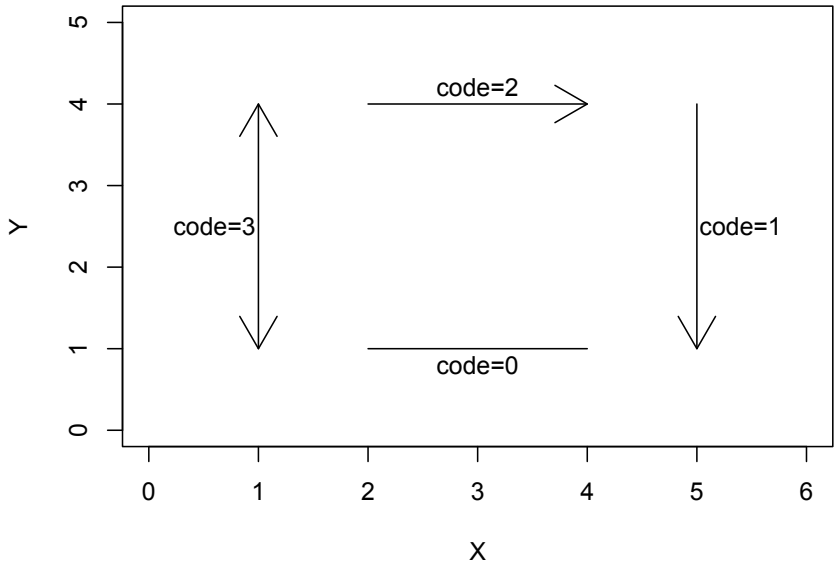

Figura 6.6: Ejemplos de flechas variando el parámetro `code`.

6.6. Función `grid`

La función `grid` es muy útil para construir rejillas sobre un gráfico, estas rejillas se pueden usar como referencia para facilitar la interpretación o como guía para ubicar elementos en un dibujo.

La estructura de la función se muestra a continuación.

```
grid(nx = NULL, ny = nx, col = "lightgray", lty = "dotted",
     lwd = par("lwd"), equilogs = TRUE)
```

Los argumentos de la función son:

- `nx, ny`: número de celdas a dibujar tanto en el eje horizontal y vertical.
- `col`: color de la rejilla.
- `lty`: tipo de línea a usar.
- `lwd`: grosor de la rejilla.

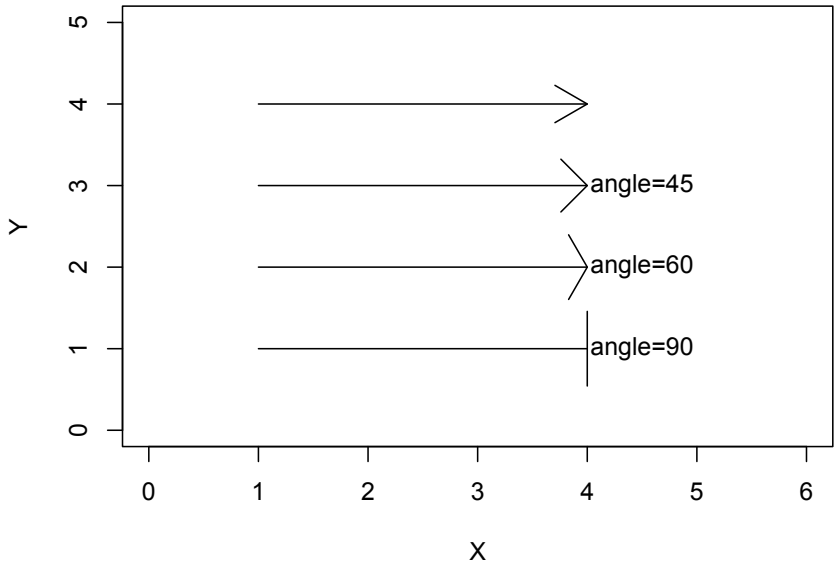

Figura 6.7: Ejemplos de flechas variando el parámetro `angle`.

A continuación se muestran dos ejemplos de como usar `grid`. En el primer ejemplo se dibuja un gráfico vacío y se le agrega la rejila estándar que se puede obtener con `grid()` sin definir ningún parámetro. En el segundo ejemplo se tiene una rejilla que divide en 4 partes el eje horizontal (`nx=3`), en 4 partes el eje vertical (`ny=4`), de color rosado, grosor 2 y con guiones largos. Abajo el código utilizado y en la Figura 6.8 se observan los resultados.

```
par(mfrow=c(1, 2), mar=c(2, 2, 0, 0), cex=0.5)
plot(c(-10, 10), c(-10, 10), type="n", xlab="", ylab="")
grid()

plot(c(-10, 10), c(-10, 10), type="n", xlab="", ylab="")
grid(nx=3, ny=4, col="pink", lwd=2, lty="longdash")
```

Para dibujar rejillas en puntos seleccionados por el usuario se puede utilizar el procedimiento explicado en la sección 7.5.

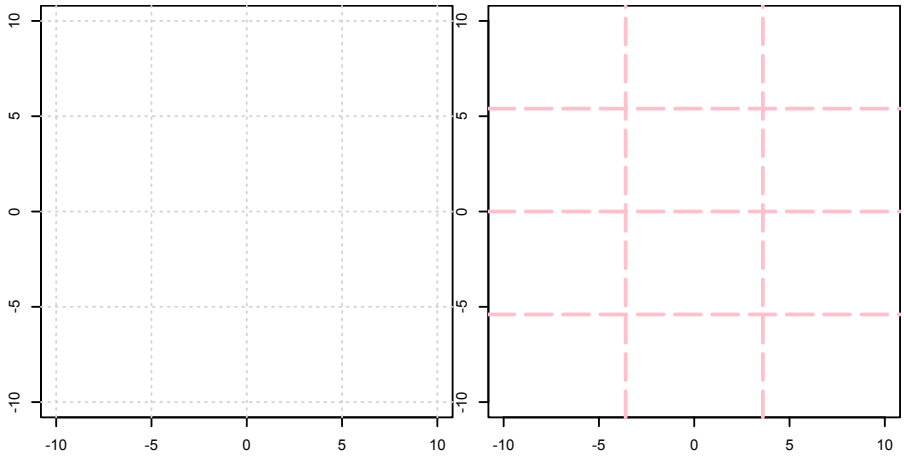

Figura 6.8: Ejemplos de rejillas con `grid`.

6.7. Función `points`

Esta función es útil para agregar puntos a una figura ya creada. La estructura de la función es la siguiente.

```
points(x, y, pch, col, cex)
```

Los argumentos de la función son:

- `x, y`: vectores con las coordenadas de ubicación de los puntos.
- `pch`: vector numérico con el tipo de punto a usar, por defecto `pch=1`.
- `col`: vector con los colores para cada punto.
- `cex`: número para modificar el tamaño de los puntos, por defecto es `cex=1` y al aumentar su valor aumenta el tamaño de los puntos.

Los diferentes tipos de puntos que se pueden obtener al variar el parámetro `pch` se muestran en la Figura 6.9.

Es posible usar otros símbolos personalizados en el parámetro `pch`, a continuación se muestra el código para incluir 5 puntos usando los símbolos @, ., $, % y w. En la Figura 6.10 se observan los 5 puntos con los símbolos usados. Observe que cuando se usó `pch='.'` aparece un pequeño punto, casi imperceptible, esta opción es muy usada cuando se tiene muchos puntos que se traslapan entre si.

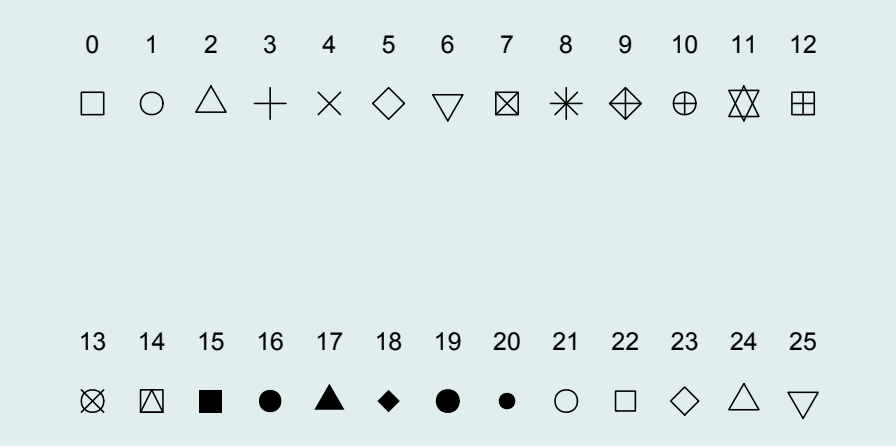

Figura 6.9: Ejemplos de los tipos de puntos obtenidos al variar `pch`.

```
par(mar=c(2, 2, 0, 0))
plot(c(0.5, 5.5), c(0.5, 5.5), xlab="", ylab="", type='n')
points(x=1:5, y=1:5,
       pch=c('@', '.', '$', '%', 'w'),
       col=1:5)
```

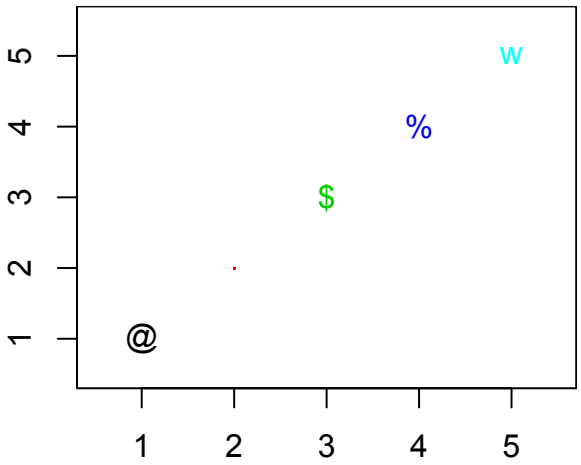

Figura 6.10: Ejemplos de símbolos personalizados con `pch`.

6.8. Función curve

Esta función sirve para dibujar una curva en un intervalo dado. La estructura de la función es la siguiente.

```
curve(expr, from = NULL, to = NULL, n = 101, add = FALSE,
      type = "l", log = NULL, xlim = NULL, ...)
```

Los argumentos de la función son:

- `expr`: nombre de la función que se desea evaluar.
- `from, to`: valores mínimo y máximo donde se va a evaluar la función.
- `n`: número de puntos en los cuales se va a evaluar la función, su valor por defecto es 101.
- `add`: valor lógico para indicar que se desea agregar la curva a un gráfico ya existente.
- `type`: tipo de representación para la función, se disponen de 9 opciones que se pueden consultar en descripción de `type` para la función `plot` en la sección 3.1.
- `xlim`: intervalo para construir la ventana gráfica, no confundir con `from` y `to`.

Ejemplo

Dibujar las funciones f_1, f_2 y f_3 usando la función `curve`, las ecuaciones para las funciones son:

$$f_1(x) = x^2 - 10 \tag{6.1}$$

$$f_2(x) = \begin{cases} x^2 - 10, & \text{si } x < 1 \\ 5x, & \text{si } x \geq 1 \end{cases} \tag{6.2}$$

$$f_3(x) = 15 - 4x \tag{6.3}$$

Dibujar f_1 en el intervalo [-2, 7], f_2 en el intervalo [-2, 4] y f_3 en el intervalo [-5, 0].

El código necesario para definir las dos funciones se muestra a continuación. Para dibujar f_2 y f_3 se usó `add=TRUE` con el objetivo de que quedaran en la misma ventana gráfica de f_1. Note que se usó `xlim=c(-5, 10)` para indicar que se deseaba una ventana gráfica en ese intervalo para el eje horizontal. En la Figura 6.11 se muestra el dibujo de las tres funciones.

```
f1 <- function(x) x^2 - 10
f2 <- function(x) ifelse(x < 1, -x^2, 5*x)
f3 <- function(x) 15 - 4 * x

curve(expr=f1, from=-2, to=7, type="l",
      col="blue", lwd=4, ylab='f(x)', xlim=c(-5, 10))

curve(expr=f2, from=-2, to=4, type="p",
      col="green4", add=TRUE)

curve(expr=f3, from=-5, to=0, type="b",
      col='red', add=TRUE, n=5)

text(x=7, y=9, "f1(x) usando type='l'", col="blue")
text(x=-2, y=4, "f2(x) usando type='p'", col="green4")
text(x=-1, y=30, "f3(x) usando type='b' y n=5", col="red")
```

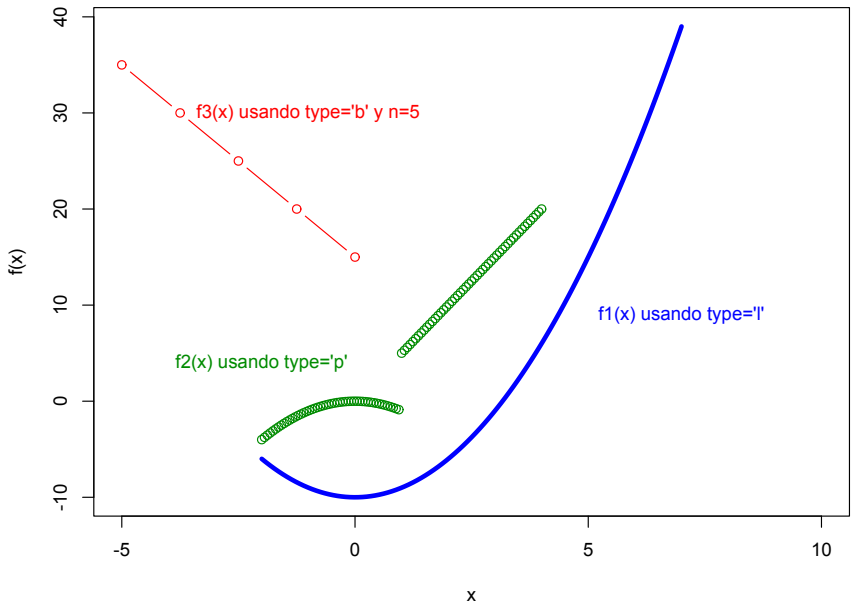

Figura 6.11: Dibujo de varias funciones en una misma ventana.

Ejemplo

Dibujar la figura de la relación entre las variables x y y dada por la siguiente expresión paramétrica:

$$x = 16\cos^3(t) \tag{6.4}$$
$$y = 13\cos(t) - 5\cos(2t) - 2\cos(3t) - \cos(4t) \tag{6.5}$$

El código para realizar la Figura 6.12 fue tomado de Stackoverflow[1] y se muestra a continuación.

```
dat <- data.frame(t=seq(0, 2*pi, by=0.1))
xhrt <- function(t) 16*sin(t)^3
yhrt <- function(t) 13*cos(t)-5*cos(2*t)-2*cos(3*t)-cos(4*t)
dat$y <- yhrt(dat$t)
dat$x <- xhrt(dat$t)
with(dat, plot(x,y, type="l", lwd=10))
with(dat, polygon(x, y, col="hotpink"))
```

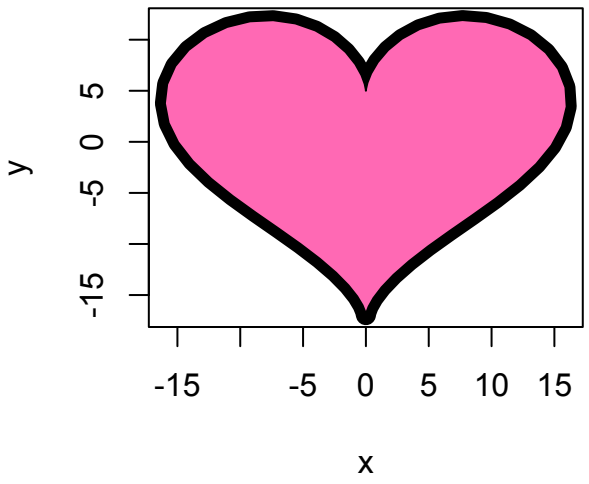

Figura 6.12: Dibujo de un cardiode.

[1] http://stackoverflow.com/questions/8082429/plot-a-heart-in-r

EJERCICIOS

1. Crear un gráfico vacío de manera que el eje horizontal tome valores desde 0 a 30 y el eje vertical valores de 0 a 20, usar los pasos de la sección 7.1 para construir el gráfico solicitado.

2. Para el gráfico vacío creado en el paso anterior, agregar una rejilla o cuadrícula que pase por los valores enteros, use los pasos de la sección 7.5 para incluir la rejilla en los valores solicitados. El gráfico resultante de los ejercicios 1 y 2 debe ser similar al presentando en la Figura 6.13.

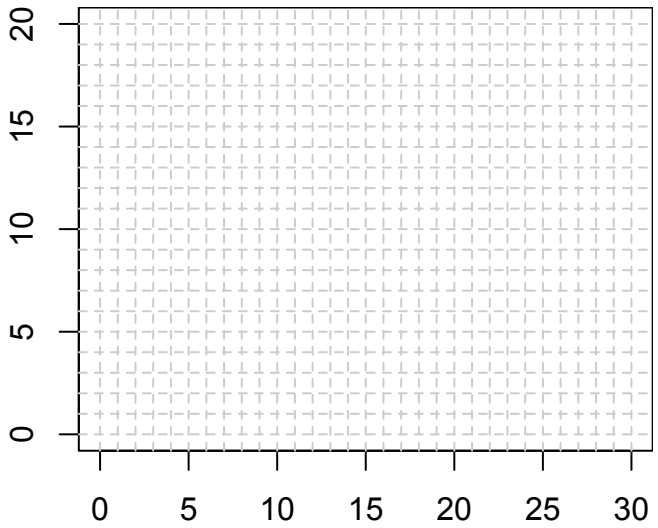

Figura 6.13: Plantilla para el dibujo.

3. Incluir en el gráfico vacío y con rejillas anterior el dibujo mostrado en la Figura 6.14. Use las funciones vistas en este capítulo para replicar el dibujo del paisaje e incluya colores para mejorar la apariencia del paisaje.

4. Dibujar las siguientes dos líneas en un mismo gráfico, la primera de color azul y la segunda de color verde.

$$2x - 5y = 8$$
$$3x + 9y = -12$$

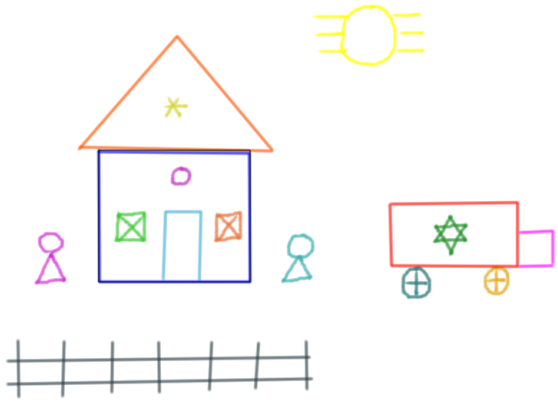

Figura 6.14: Paisaje a dibujar.

5. Dibujar la siguiente figura dada en forma parametrica.

$$x = \cos(t)$$
$$y = \text{sen}(t)$$

7

Respuestas a preguntas frecuentes

En este capítulo se presentan las respuestas a preguntas frecuentes que tienen los usuarios de R sobre la construcción de gráficos personalizados.

7.1. ¿Cómo crear un gráfico vacío?

Es muy frecuente que se necesite crear un gráfico vacío para luego agregar ciertos elementos y construir un nuevo gráfico de forma incremental. Hacer esto en R es muy sencillo, solo se debe dibujar el gráfico con `plot` incluyendo `type='n'` para que no dibuje nada.

A continuación se presenta el código para crear un gráfico vacío con valores de X entre -5 y 5, valores de Y entre 0 y 1, sin nombres en los ejes. En el pánel izquierdo de la Figura 7.1 se muestra el gráfico resultante, no se observa nada adentro porque esa era la idea, un gráfico vacío. Es posible crear un gráfico más vacío aún, sin ejes ni la caja alrededor del gráfico. En el pánel derecho de la Figura 7.1 se muestra el gráfico resultante con un mensaje para alertar que no es un error en la figura.

```
par(mfrow=c(1, 2))
plot(c(-5, 5), c(0, 1), xlab="", ylab="", type='n')

plot(c(-5, 5), c(0, 1), xlab="", ylab="", type='n',
     xaxt='n', yaxt='n', bty='n')
text(x=0, y=0.5,
     label='Gráfico completamente vacío.')
```

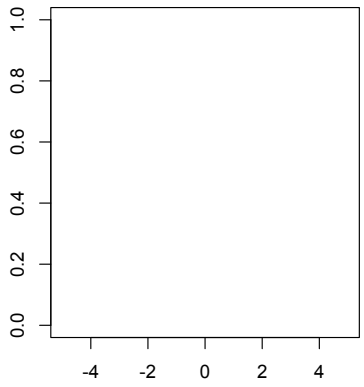

Gráfico completamente vacío.

Figura 7.1: Ejemplo de gráfico vacío.

7.2. ¿Cómo personalizar los valores a mostrar en los ejes?

Supongamos que tenemos un gráfico de dispersión y que queremos que el eje horizontal presente sólo tres marcas en los valores 2, 6 y 10, y que el eje vertical presente también tres marcas en los valores 0, 15 y 25.

Para realizar esto construimos el diagrama de dispersión de la forma usual con la función `plot` pero le agregamos dentro lo siguiente `xaxt='n', yaxt='n'`, esto se agrega para evitar que `plot` coloque marcas en los ejes. Luego se usa la función `axis(side, at, labels)` para colocar las marcas en cada uno de los ejes. Los parámetros básicos de la función `axis` son:

- `side`: número para indicar el eje a completar, 1 para eje horizontal inferior, 2 para el eje vertical izquierdo, 3 para el eje horizontal superior y 4 para el eje vertical a la derecha.
- `at`: vector con los puntos donde se desean las marcas.
- `labels`: vector con las etiquetas a colocar en las marcas.

A continuación se muestra un ejemplo de cómo colocar tres marcas tanto en el eje horizontal como vertical. En la Figura 7.2 se muestra el resultado, a la izquierda está el diagrama obtenido por defecto y a la derecha el mismo diagrama pero con las tres marcas en los ejes.

```
x <- 1:10          # Los datos para el dibujo
y <- (x - 5) ^ 2   # Los datos para el dibujo

par(mfrow=c(1, 2)) # Para dividir la ventana gráfica
plot(x=x, y=y, main='Dibujo por defecto')  # Dibujo por defecto
plot(x=x, y=y, xaxt='n', yaxt='n',
     main='Con marcas personalizadas')  # Sin marcas
axis(side=1, at=c(2, 6, 10), labels=c(2, 6, 10))
axis(side=2, at=c(0, 15, 25), labels=c(0, 15, 25))
```

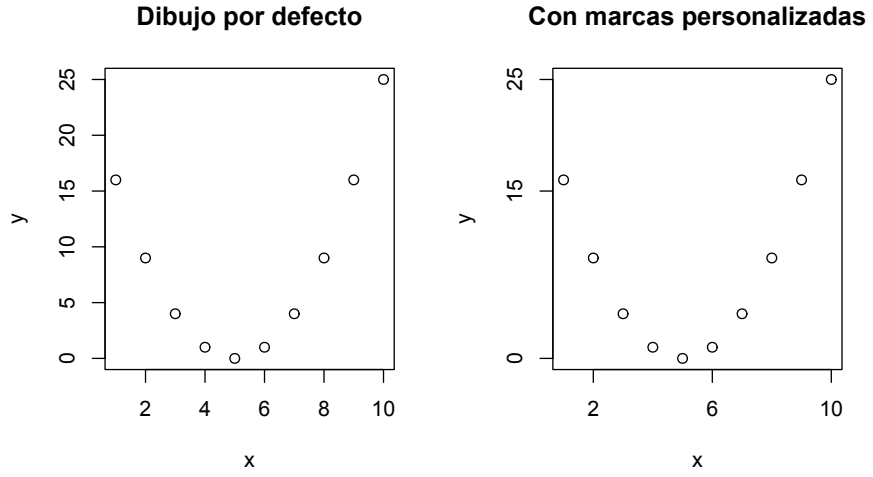

Figura 7.2: Personalizando los valores a mostrar en los ejes.

7.3. ¿Cómo dibujar varios QQplot en una misma figura?

Para colocar varios QQplot en una misma figura lo primero que se debe hacer es aplicar la función `qqnorm` a los datos pero usando `plot.it=FALSE` para que no se dibuje nada, luego se almacenan los resultados en un objeto. Ese objeto resultante será una lista con las coordenadas de ubicación de los puntos. Se hace un gráfico vacío y sobre éste se colocar los puntos y las líneas de referencia.

Supongamos que tenemos dos muestras aleatorias `m1` y `m2` para las cuales

queremos dibujar QQplots. El código para construir el gráfico solicitado se muestra a continuación y el resultado es la Figura 7.3.

```
m1 <- rnorm(n=15, mean=170, sd=10)  # Simulando las muestras
m2 <- rnorm(n=20, mean=160, sd=15)

q1 <- qqnorm(m1, plot.it=FALSE)
q2 <- qqnorm(m2, plot.it=FALSE)
plot(range(q1$x, q2$x), range(q1$y, q2$y), type='n', las=1,
     xlab='Theoretical Quantiles', ylab='Sample Quantiles')
points(q1, pch=19, col='slateblue3')
points(q2, pch=19, col='seagreen4')
qqline(m1, col='slateblue3')
qqline(m2, col='seagreen4')
legend('topleft', legend=c('Muestra 1', 'Muestra 2'), bty='n',
       col=c('slateblue3', 'seagreen4'), pch=19)
```

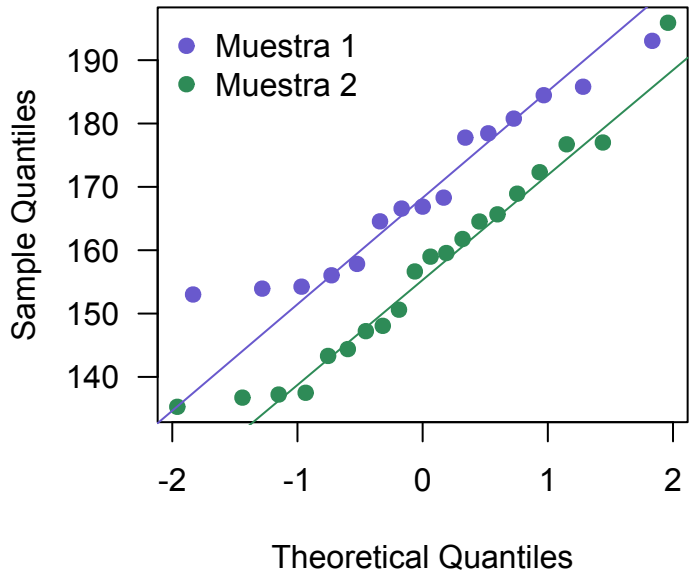

Figura 7.3: QQplots simultáneos.

7.4. ¿Cómo dibujar varias densidades en una misma figura?

Para dibujar varias densidades en una misma figura se procede de forma similar al ejemplo anterior. Se construyen las densidades pero sin dibujarlas, luego se crea un gráfico y se van agregando una a una las densidades. Abajo el código para dibujar dos densidades a partir de dos muestras aleatorias `m1` y `m2`. En la Figura 7.4 se muestran las dos densidades.

```
m1 <- rnorm(n=100, mean=3, sd=1)        # Generando los datos
m2 <- rgamma(n=100, shape=2, scale=1)

f1 <- density(m1)  # Calculando las densidades
f2 <- density(m2)

plot(f1, main='', las=1, lwd=2,
     xlim=range(f1$x, f2$x), ylim=range(f1$y, f2$y),
     xlab='Variable', ylab='Densidad')
lines(f2, lwd=2, col='red')  # Para agregar f2
legend('topright', col=c('black', 'red'),
       lwd=2, bty='n', legend=c('Muestra 1', 'Muestra2'))
```

7.5. ¿Cómo incluir una rejilla a una figura en ciertos puntos?

La función `grid` explicada en la sección 6.6 se usa para incluir una cuadrícula o rejilla, sin embargo, no se le puede indicar los sitios donde queremos que aparezcan las líneas de referencia. La función `abline` es útil en este caso.

A continuación se muestra el código de ejemplo para construir una rejilla que pase por los puntos -9, -6, -3, 0, 3, 6 y 9 del eje horizontal, y que pase por los puntos -10,-5, 0, 5 y 10 del eje vertical. Los parámetros `v` y `h` se usan para colocar el vector con las posiciones donde se desea la rejilla. El color de la rejilla se puede personalizar con el parámetro `col` y el tipo de línea a dibujar con `lty`. En la Figura 7.5 se observa el resultado.

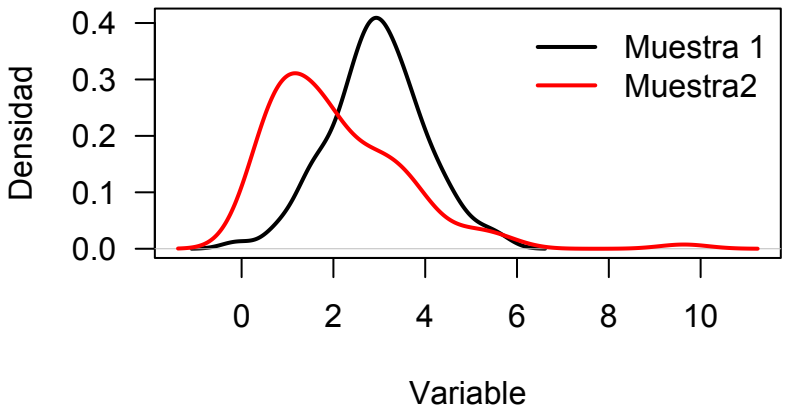

Figura 7.4: Densidades simultáneas.

```
plot(c(-10, 10), c(-10, 10), type="n", xlab="", ylab="")
abline(v=c(-9, -6, -3, 0, 3, 6, 9),
       h=c(-10,-5, 0, 5, 10), col=gray(0.8), lty='dashed')
```

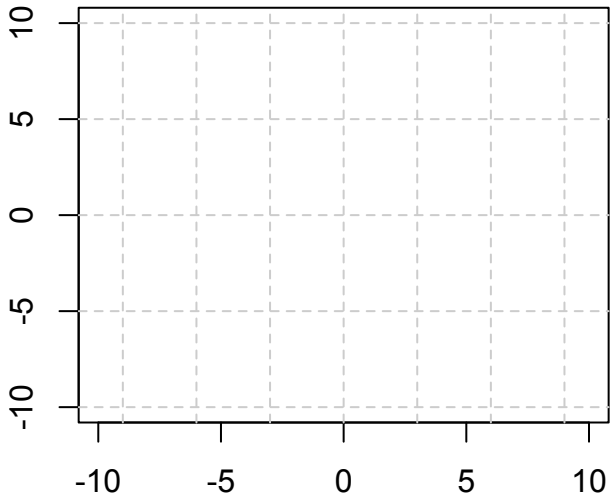

Figura 7.5: Rejilla personalizada.

8

Extras

```
datos <- matrix(sample(0:7,10000,
                       prob=c(260, 354, 200, 100, 60, 20, 5, 1),
                       replace=T), ncol=20)
medias<-apply(datos, 1, mean)

library(MASS)
resu1 <- fitdistr(x=medias, densfun='gamma')$estimate
resu1

hist(medias, col='yellow', freq=F, las=1,
     main='Distribución A priori Aproximada del Parámetro',
     ylab='Densidad', xlab=expression(lambda))
xx <- seq(0,3,length=100)
yy <- dgamma(xx, resu1[1], rate=resu1[2])
points(xx, yy, type='l')
legend(1.8, 1.2, bty='n',
       c(expression(alpha),expression(beta)))
legend(1.87, 1.2, c('=','='), bty='n')
resu11 <- format(resu1, digits=6) # Función format()
legend(1.91, 1.2, resu11, bty='n')
```

Bibliografía

Hernández, F. & Usuga, O. (2018). *Manual de R*. Universidad Nacional de Colombia, Medellín, Colombia, primera edition. ISBN xxx-xxxxxxxxx.

Wickham, H. (2015). *R Packages*. O'Reilly Media, Inc.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.3.

Índice alfabético