Juan Carlos Correa Morales
Freddy Hernández Barajas

Gráficos con R

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 ...

¿Por qué leer este libro?

Este libro es importante porque ...

Estructura del libro

En el capítulo **??** 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). La información de la sesión de R usada se muestra abajo:

```
sessionInfo()
```

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 7 x64 (build 7601) Service Pack 1
##
## locale:
## [1] LC_COLLATE=English_Australia.1252
```

```
## [2] LC_CTYPE=English_Australia.1252
## [3] LC_MONETARY=English_Australia.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_Australia.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets
## [6] methods   base
##
## loaded via a namespace (and not attached):
##  [1] backports_1.0.4 bookdown_0.3    magrittr_1.5
##  [4] rprojroot_1.1   tools_3.3.2     htmltools_0.3.5
##  [7] rstudioapi_0.6  yaml_2.1.14     Rcpp_0.12.8
## [10] stringi_1.1.2   rmarkdown_1.3   knitr_1.15.1
## [13] stringr_1.1.0   digest_0.6.11   evaluate_0.10
```

Package names are in bold text (e.g., **rmarkdown**), and inline code and filenames are formatted in a typewriter font (e.g., `knitr::knit('foo.Rmd')`). Function names are followed by parentheses (e.g., `bookdown::render_book()`).

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.

Juan Carlos Correa Morales
Freddy Hernández Barajas

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.

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

1

Introducción

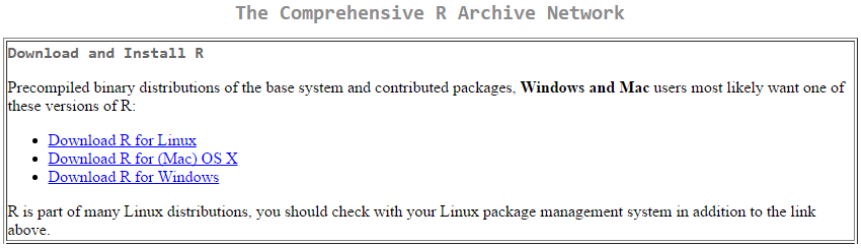
1.1. Orígenes

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.


The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

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/

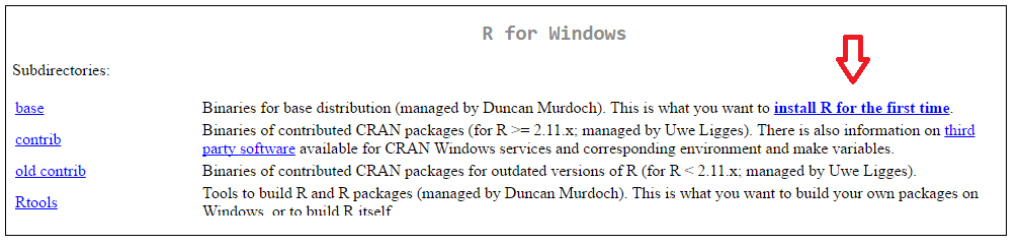


Figura 1.3: Página de instalación para la primera ocasión.

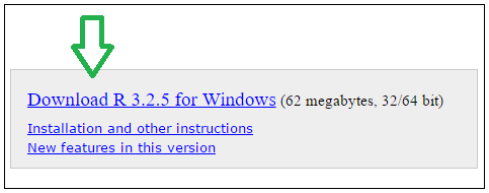


Figura 1.4: Página de descarga.

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

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

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

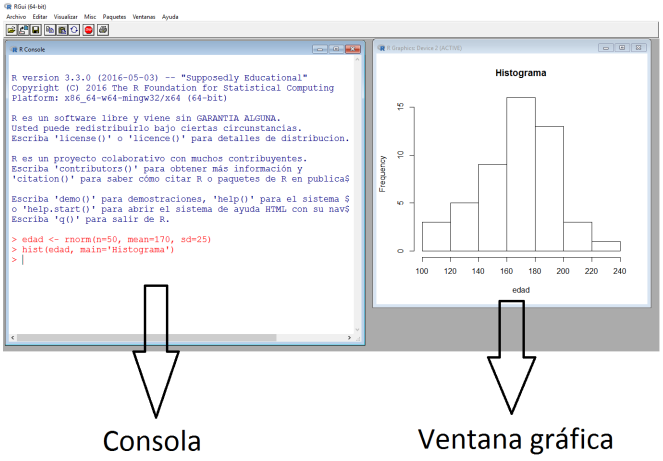


Figura 1.6: Apariencia de R.

consola, para hacer esto se deben explorar las opciones de *editar* en la barra de herramientas.

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 cualitativa) 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 las comillas `' '`.

Nota: 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

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

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

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

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

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.

```r
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'))
```

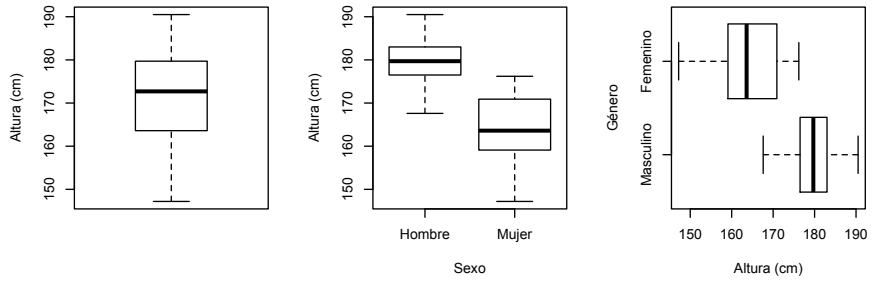


Figura 2.1: Boxplot para la variable altura.

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)
```

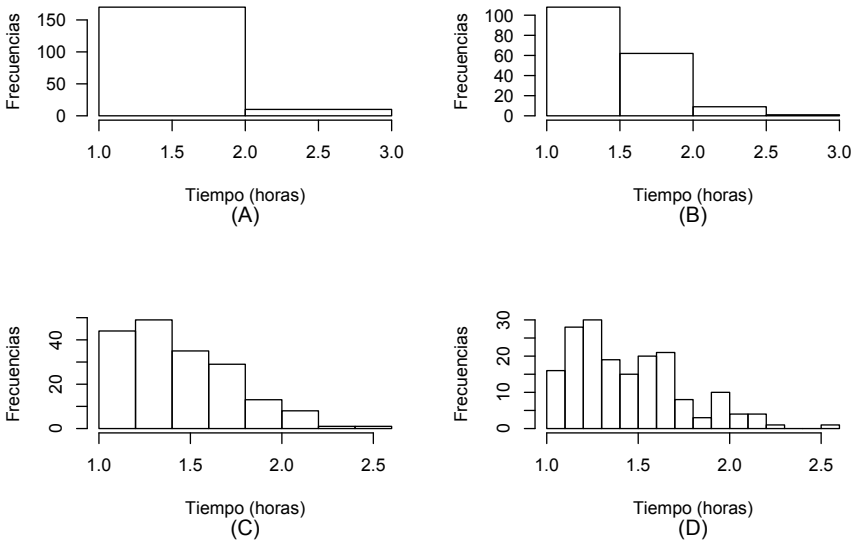


Figura 2.2: Histogramas para el tiempo en la media maratón de CONAVI. A: histograma con dos intervalos, B: histograma con cuatro intervalos, C: histograma con seis intervalos, C: histograma con 18 intervalos.

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.

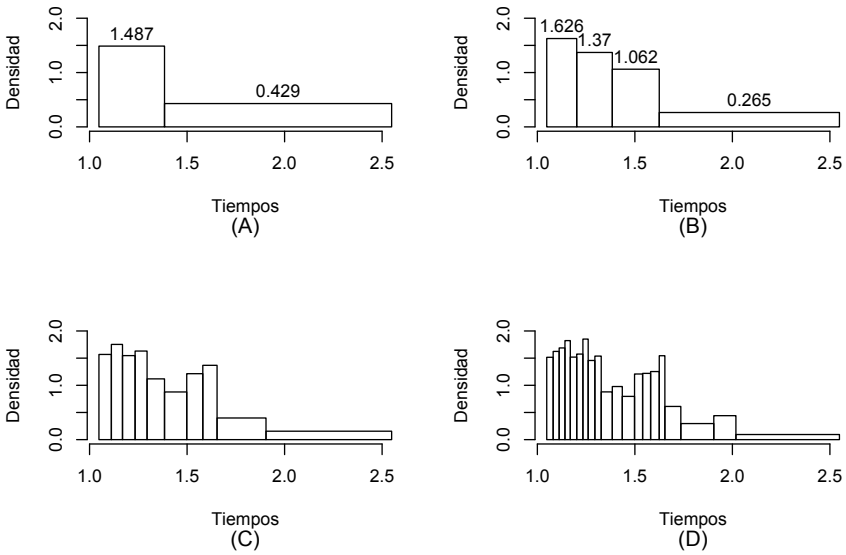


Figura 2.3: Histogramas para el tiempo en la media maratón de CONAVI. A: histograma con dos intervalos, B: histograma con cuatro intervalos, C: histograma con diez intervalos, C: histograma con veinte intervalos.

```
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)
```

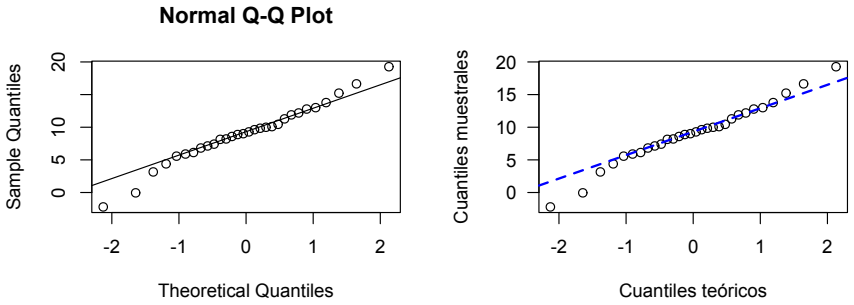


Figura 2.4: Gráfico cuantil cuantil para una muestra generada de una población normal.

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))
```

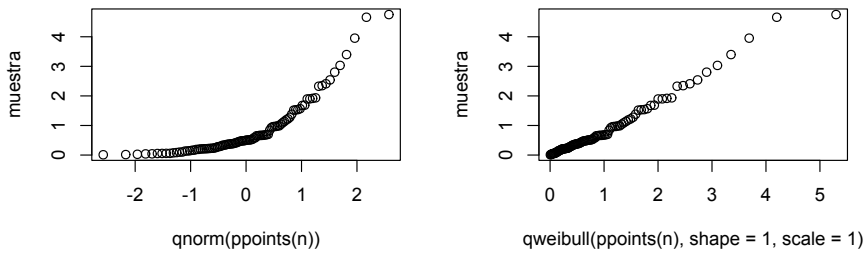


Figura 2.5: Gráfico cuantil cuantil para una muestra generada de una población Weibull.

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'))
```

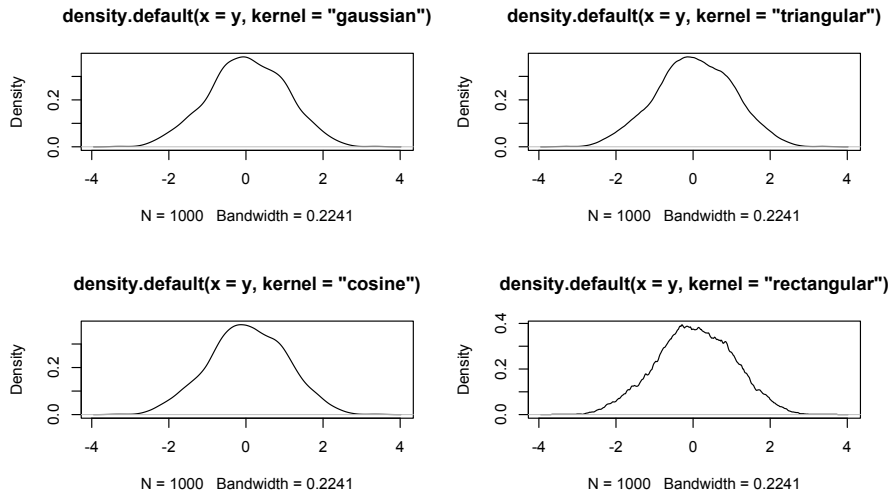


Figura 2.6: Densidad para una muestra aleatoria de una N(0, 1) cambiando el núcleo de la densidad.

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

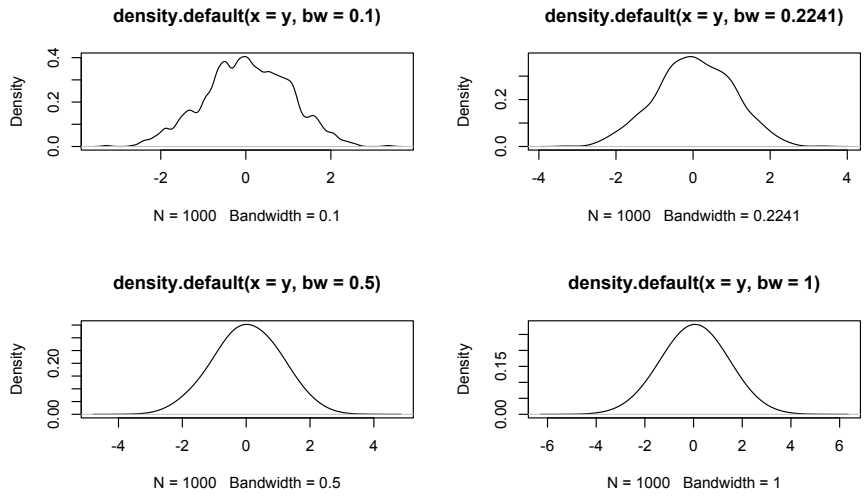


Figura 2.7: Densidad para una muestra aleatoria de una N(0, 1) cambiando el ancho de banda.

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='Densidad para el 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='Densidad para el peso corporal', 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'))
```

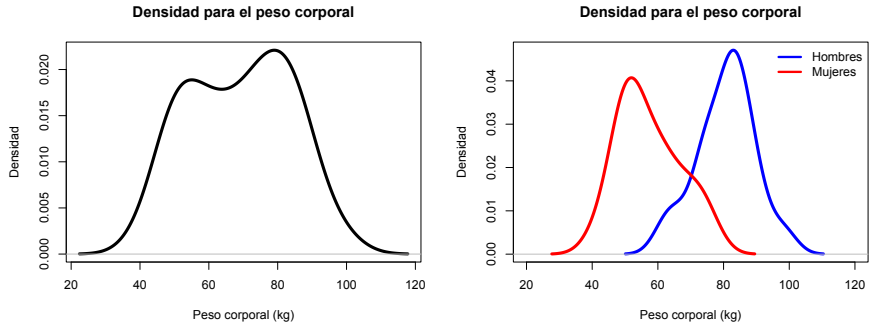


Figura 2.8: Densidad para la variable peso en la izquierda, densidad para el peso diferenciando por sexo a la derecha.

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

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

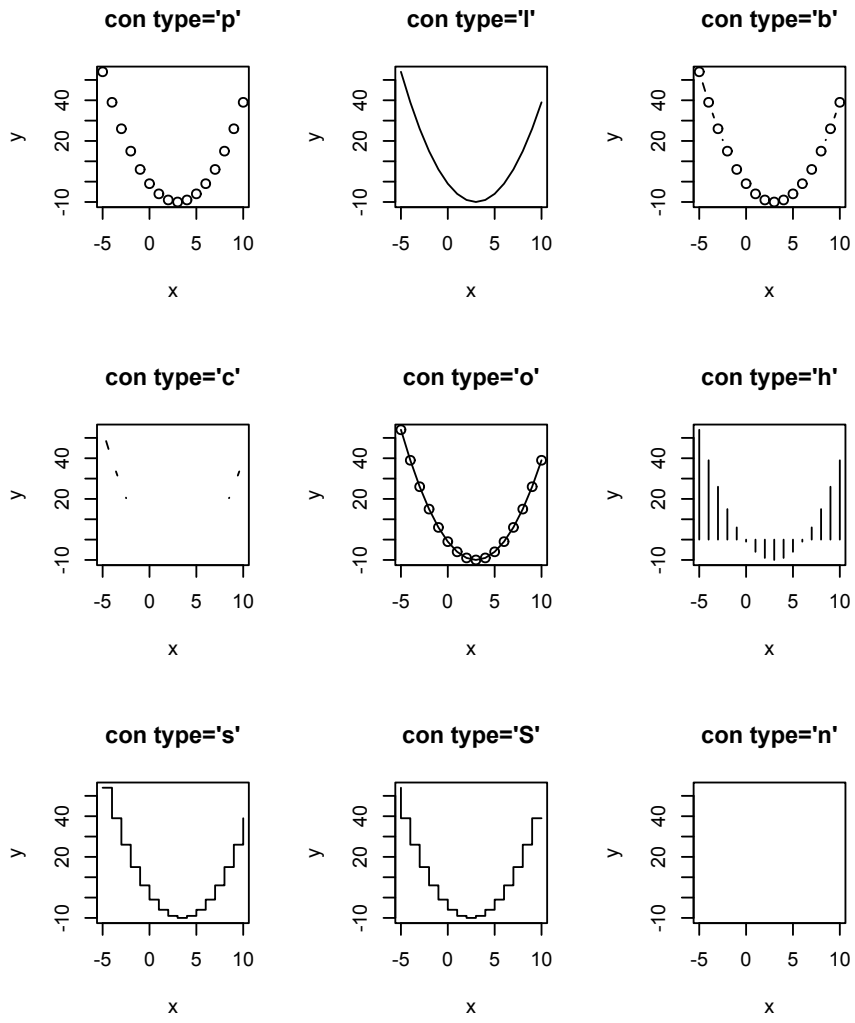


Figura 3.1: Efecto del parámetro `type` en la función `plot`.

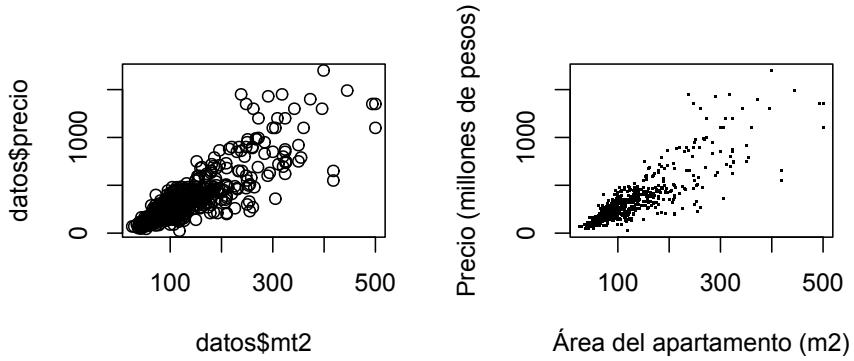


Figura 3.2: Diagrama de dispersión del precio del apartamento versus área del apartamento. A la izquierda el diagrama de dispersión sin editar y a la derecha el diagrama de dispersión mejorado

`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 `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.
- `main`: vector numérico con las coordenadas del eje vertical.
- `sub`: vector numérico con las coordenadas del eje vertical.

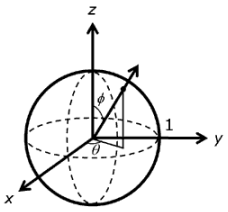


Figura 3.3: Ilustración de los angulos theta y phi para la función persp. Figura tomada de https://i-msdn.sec.s-msft.com/dynimg/IC412528.png

- `theta, phi`: ángulo para la visión de la superficie, `theta` para la dirección azimutal y `phi` para latitud. Ver Figura 3.3 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')
```

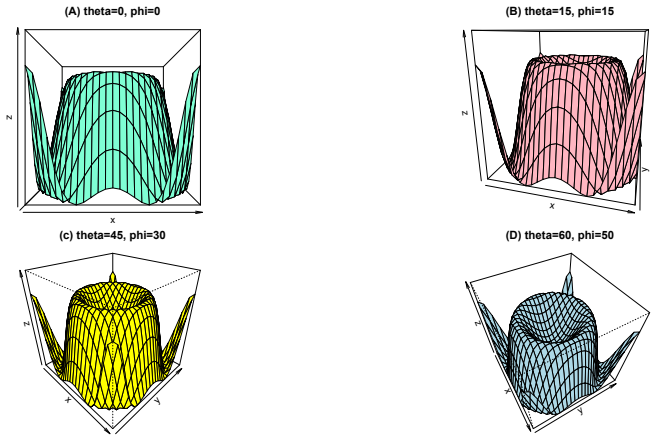


Figura 3.4: Superficie generada con `persp` y diferentes valores de theta y phi.

En la Figura 3.4 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)
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)
```

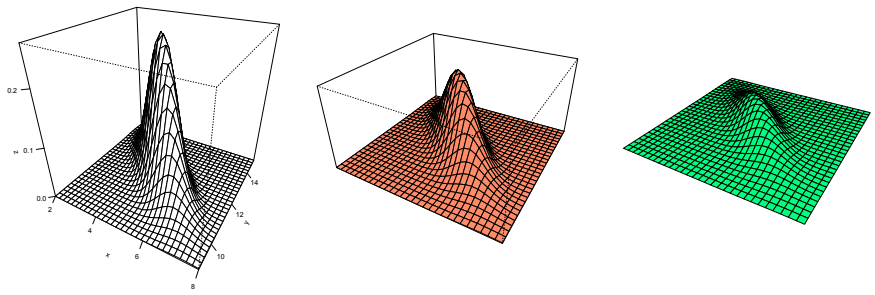


Figura 3.5: Distribución normal bivariada.

En la Figura 3.5 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.

3.3. 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 3.6 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.

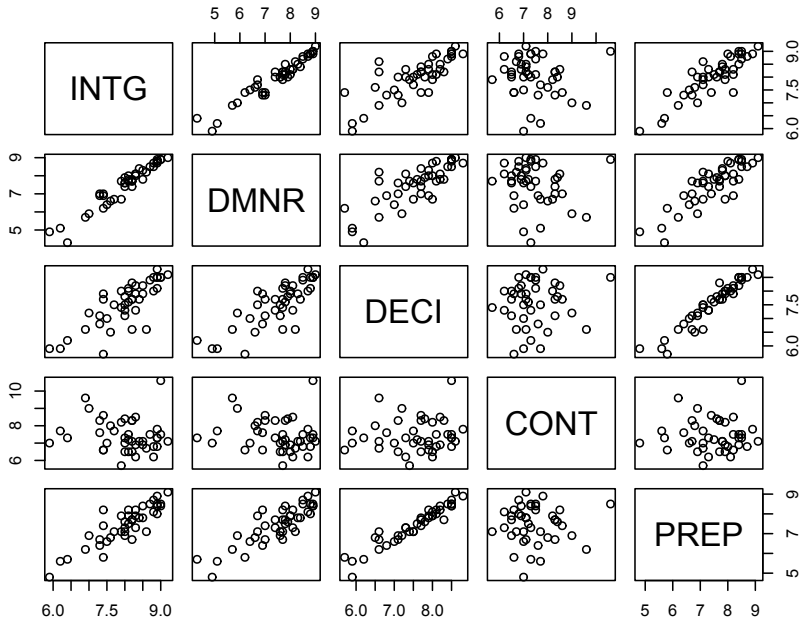


Figura 3.6: Ilustración de una matriz de dispersión.

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

```
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)
```

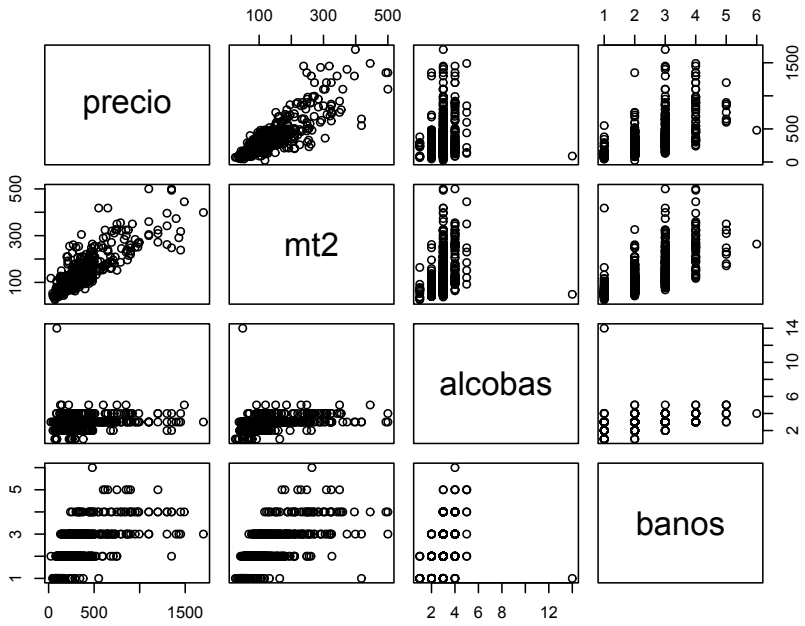


Figura 3.7: Matriz de dispersión para las variables precio, área, número de alcobas y número de baños de la base de datos sobre apartamentos en Medellín.

En la Figura 3.7 se muestra la matriz de dispersión para las variables del marco de datos `datos.num`.

Ejemplo

Volver a construir la Figura 3.7 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.

Para obtener la nueva matriz de dispersión con los cambios solicitados se usa el siguiente código. En la Figura 3.8 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')
```

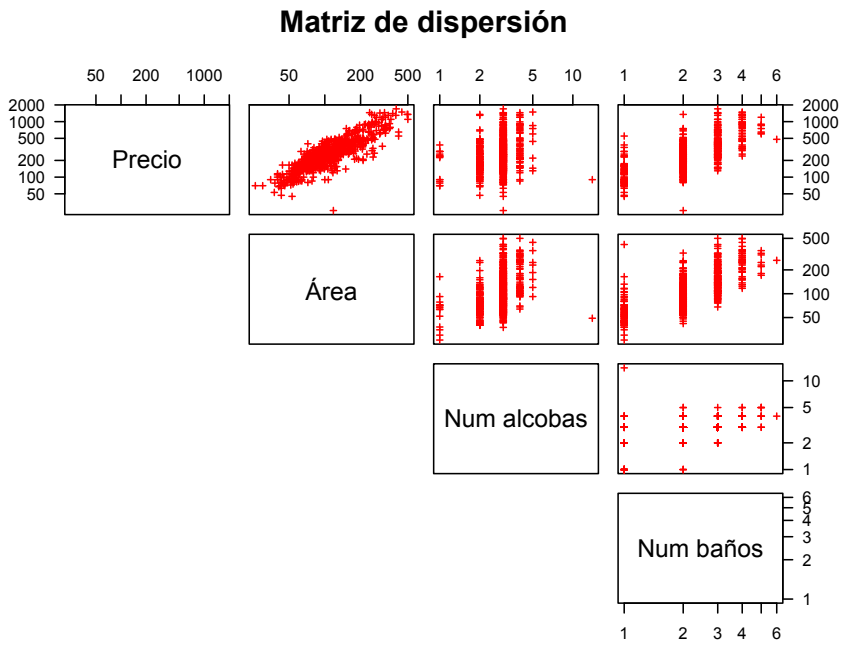


Figura 3.8: Matriz de dispersión modificando los parámetros adicionales de la función pairs.

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 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 < 150 mt2")
```

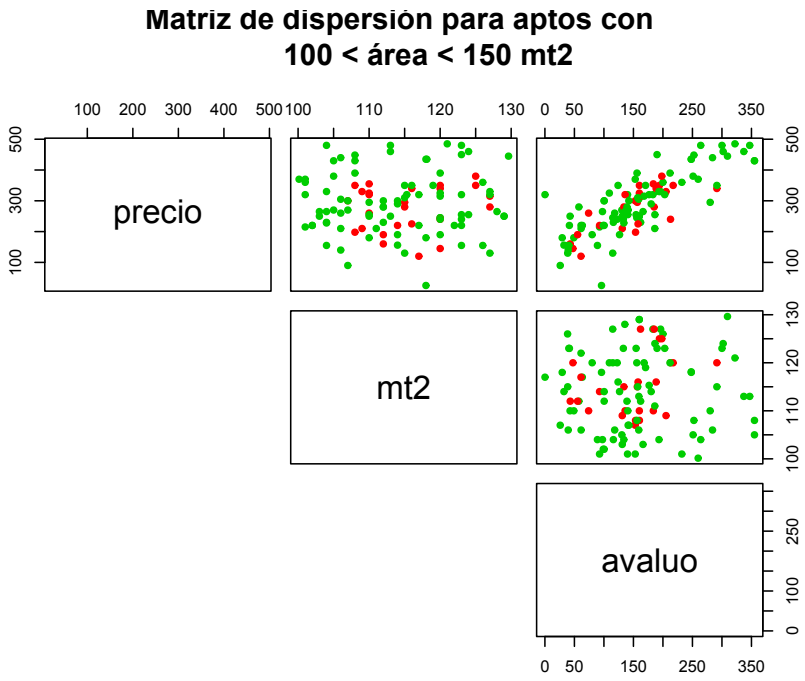


Figura 3.9: Matriz de dispersión con un subconjunto de los datos y con colores para identificar los puntos.

En la Figura 3.9 se presenta la matriz de dispersión solicitada, los puntos rojos 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 3.10 se presenta el resultado.

```
pairs(iris[1:4], main="Anderson's Iris Data -- 3 species",
      pch=21, bg=c("red", "green3", "blue")[iris$Species],
      oma=c(4, 4, 6, 12))
par(xpd=TRUE)
legend(0.85, 0.7, as.vector(unique(iris$Species)), bty='n',
       fill=c("red", "green3", "blue"))
```

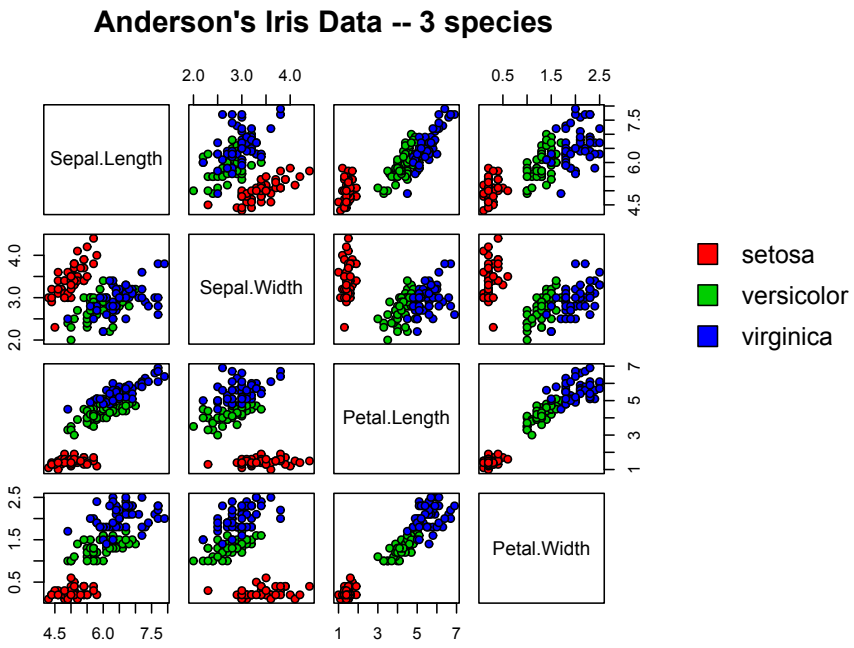


Figura 3.10: Matriz de dispersión con leyenda.

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

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)
```

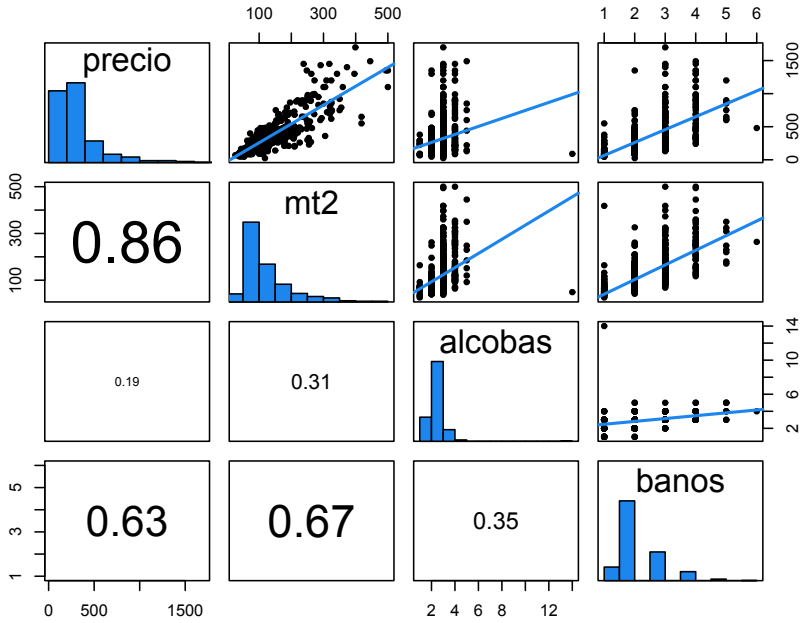


Figura 3.11: Matriz de dispersión con páneles modificados.

En la Figura 3.11 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.

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 3.12 se muestra gráfico obtenido.

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

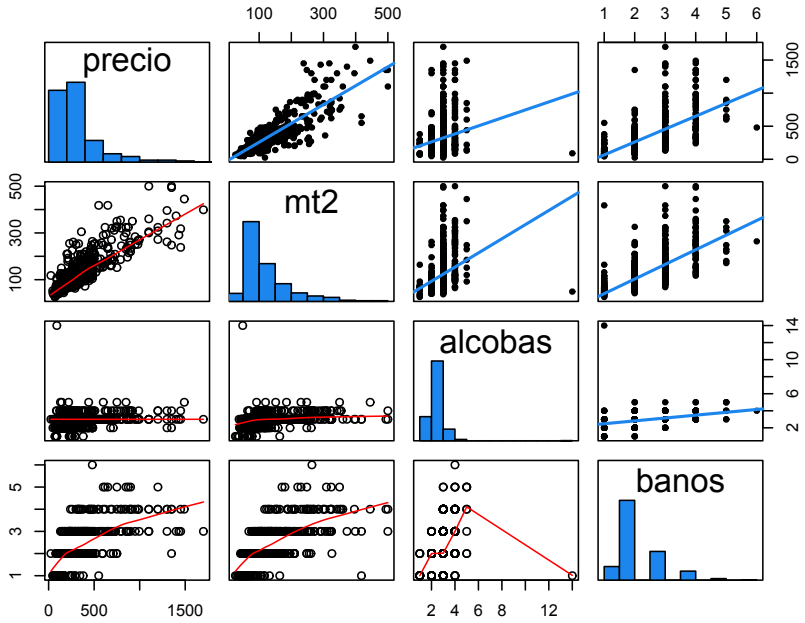


Figura 3.12: Matriz de dispersión usando la función panel.smooth.

3.4. Función `contour`

A

More to Say

Yeah! I have finished my book, but I have more to say about some topics. Let me explain them in this appendix.

To know more about **bookdown**, see https://bookdown.org.

Bibliografía

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