Trabajo Práctico 3: Introducción al lenguaje R - cont.

4. Introducción a los Gráficos

Uno de los poderes mayores de R es su capacidad gráfica. Veremos algunas de estas características en esta sección.

4.1 La ventana gráfica

Cuando las imágenes se crean en R, se presentan en un ventana gráfica activa (active graphical *device*). Si no está abierta el momento de ejecutar una fución gráfica, R abrirá una. Algunas de las características de la ventana gráfica son:

- Se puede imprimir directamente desde la ventana gráfica, o elegir copiar el gráfico al clipboard y pegarlo a un procesador de textos. Allí se puede modificar el tamaño del gráfico. También, un gráfico se puede guardar en distintos formatos como pdf, bitmap, metafile, jpeg, o postscript.

- Cada vez que se produce un gráfico nuevo, se pierde el anterior.

4.2 Dos funciones gráficas básicas

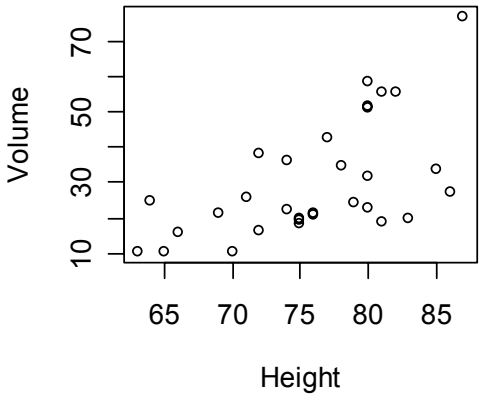
Hay muchas funciones en R que producen gráficos y van desde las muy básicas hasta las más avanzadas y complicadas. En esta sección describiremos dos funciones básicas y en las siguientes veremos como embellecer los gráficos. En el capítulo 5 veremos otras funciones gráficas.

4.2.1 La función **`plot()`**

La función más común para realizar un gráfico en R es la función **`plot()`**. Esta es una función genérica que se puede utilizar para realizar gráficos de dispersión (scatterplots), gráficos de series de tiempo, de funciones, etc.
Si se da como argumento un único vector a **`plot()`**, los valores son graficados sobre el eje vertical y (ordenadas) contra los índices del vector (*index*) en el eje horizontal x (abscisas). Si son dados dos objetos vecoriales (de la misma longitud), se produce un gráfico de disperión bivariado. Por ejemplo consideremos nuevamente el dataset **`trees`** de R. Para visualizar la relación entre la altura y el volumen (**`Height, Volume`**), podemos we can dibujar un gáfico de dispersión:

```
> attach(trees)
>  plot(Height, Volume) # el objeto trees está en el search path
```

Aparece el gráfico en una ventana aparte:



La primera variable es graficada sobre el eje horizontal y la segunda variable es graficada sobre el eje vertical. Por defecto los nombres de las variables aparecen encada uno de los ejes.

Este gráfico es extremadamente básico, pero la función **`plot()`** permite "vestirlo" simplemente cambiando los argumentos que la función tiene por defecto. Estos incluyen el agregado de títulos/subtítulos, el cambio de los colores de los caracteres gráficos (están disponibles más de 600 colores). Se puede consultar **`?par`** para obtener una abrumadora lista con estas opciones.

Esta función será utilizada en los próximos capítulos.

4.2.2 La función ***`curve()`***.

Puede utilizarse la función **`curve()`**para graficar una función continua sobre un rango especificado de valores (a pesar de ser interesante la función **`curve()`** en realidad llama a la función **`plot()`**). La forma básica para utilizar esta función es:

```
curve(expr, from, to, add = FALSE, ...)
```

Argumentos:

`expr`: una expresión escrita como función de **`'x'`**

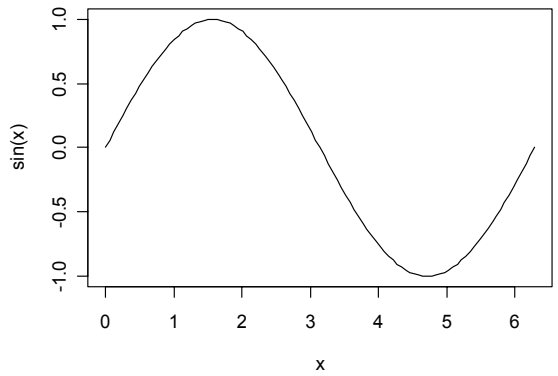
`from, to`: el rango sobre el cual la función será graficada.

`add`: argumento lógico (logical); si es **`'TRUE'`** el gráfico se agrega al existente.

Observe que es necesario que el argumento de **`expr`** esté escrito siempre como función de **`'x'`**. Si el argumento **`add`** se fija en **`TRUE`**, el gráfico de la función será superpuesto al gráfico que se encuentra en la ventana gráfica activa (ilustraremos esta útil propiedad en el Capítulo 6).

Por ejemplo la función **`curve()`**puede ser utilizada para graficar la función seno de 0 a 2π:

```
> curve(sin(x), from = 0, to = 2*pi)
```



4.3 Embelleciendo los Gráficos

4.3.1 *Argumentos de funciones gráficas*

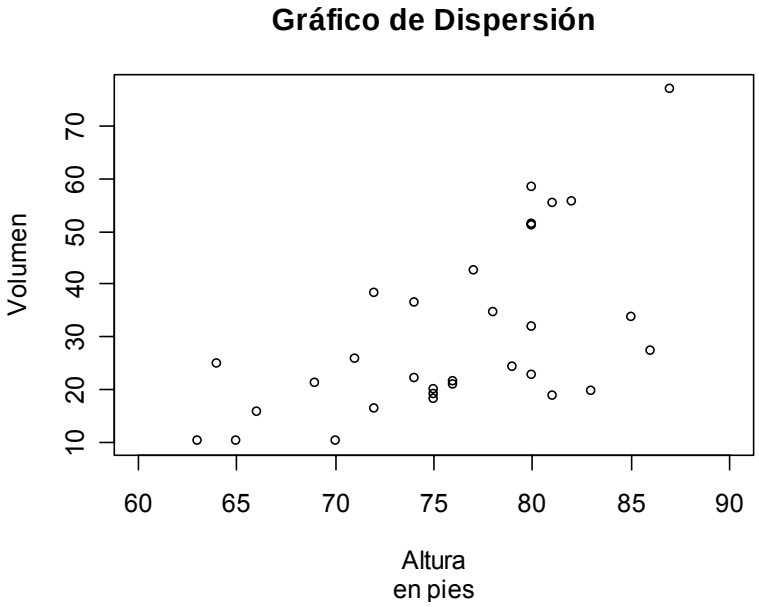
El aspecto de los gráficos generados por la función **`plot`** puede mejorarse modificando los valores por defecto de sus argumentos.Veamos los principales:

Argumento	**Función**
`xlim = , ylim =`	Especifica los límites superiores e inferiores de los ejes
`xlab = , ylab =`	Especifica las leyendas de los ejes
`main =`	título principal
`sub =`	subtítulo
`pch =`	

Mediante las siguientes instrucciones

```
> plot(Height, Volume, xlab="Altura", ylab="Volumen",
+             xlim = c(60,90), ylim=range(Volume),
+             main = "Gráfico de Dispersión", sub="en pies")
```

obtenemos



4.3.2 *Comandos gráficos de bajo nivel*

Sumadas a las funciones gráficas estándar, existe un ejército de funciones que se pueden *agregar* a un gráfico dibujado en una pantalla gráfica. Se denominan *comandos gráficos de bajo nivel* pues afectan un gráfico ya existente:

Incluyen (para obtener información adicional ver el help para cada una de las funciones):

Función	**Operación**
`abline()`	Agrega una línea con ordenada al origen y pendiente especificadas
`arrows()`	Agrega una flecha en coordenadas de comienzo y final especificadas
`lines()`	Agrega líneas entre coordenadas
`points()`	Agrega puntos en coordenadas especificadas
`rug()`	Agrega, sobre el eje x, los datos del eje x como pequeñas líneas verticales
`segments()`	Agrega un segmento lineal entre un par de puntos
`text()`	Agrega un texto en coordenadas establecidas, posiblemente dentro de la región graficada
`title()`	Agrega títulos, subtítulos, etc.

4.4 Modificación de Parámetros Gráficos

Es posible modificar las presentaciones gráficas aún más. Para lograr que la apariencia de todos los gráficos subsecuentes sigan un formato determinado pueden modificarse los parámetros gráficos que vienen por defecto utilizando la función **`par()`**. Hay más de 70 parámetros gráficos que pueden ser ajustados, mencionaremos solo algunos. Veamos algunos muy útiles:

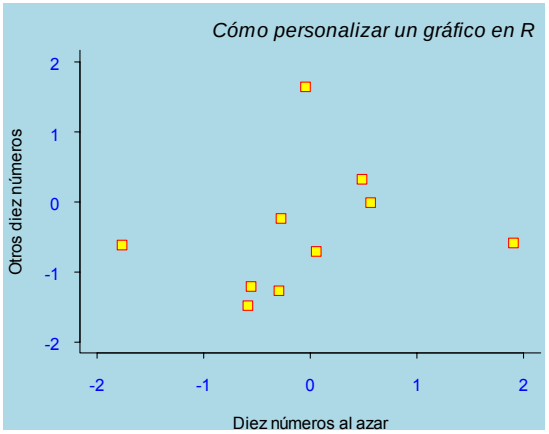
```
> par(mfrow = c(2, 2) # divide la zona del gráfico en 4
                      # y permite realizar 4 gráficos distribuidos
                      # en una matriz de 2x2
> par(bg = "cornsilk")# todos los gráficos con este fondo

> par(xlog = TRUE)    # siempre grafique el eje x axis
                      # en una escala logarítmica
```

Se puede modificar alguno o todos los parámetros en el comando **par()**y el efecto se mantendrá hasta que sean modificados nuevamente o hasta que se salga del programa. Se puede guardar una copia de los parámetros preestablecidos en **par()**, y luego después de haber realizado los cambios recuperar los parámentros originales. Para hacer esto:

```
> oldpar <- par(no.readonly = TRUE)
... luego, se realizan los cambios en par() ...
> par(oldpar) # default (originales) se restauran los parámetros
```

Veamos un ejemplo:

```
> oldpar <- par(no.readonly = TRUE)
> par(bg="lightblue", col.axis="blue", mar=c(4, 4, 2.5, 0.25))
> plot(runif(10,-2,2), runif(10,-2,2),
+ xlab="Diez números al azar", ylab="Otros diez números",
+ xlim=c(-2, 2), ylim=c(-2, 2), pch=22, col="red", bg="yellow",
+ bty="l", tcl=-.25, las=1, cex=1.5)
> title("Cómo personalizar un gráfico en R ", font.main=3, adj=1)
> par(oldpar) # default
```



4.5 Ejercicios

Veremos más gráficos en las próximas secciones. Ahora entre los siguientes comandos para ver algunos especiales.

```
> demo(graphics)
> demo(persp) # for 3-d plots
> demo(image)
```

5. Resumiendo Datos

Unas de las maneras más simples de describir qué está ocurriendo con un conjunto de datos es utilizar resúmenes numéricos o gráficos. Los resúmenes numéricos son por ejemplo medias, proporciones, desvíos estándar mientras que los resúmenes graáficos incluyen histogramas y gráficos caja (boxplots).

5.1 Resumenes numéricos

R incluye un ejército de funciones incorporadas para calcular estadísticos muestrales tanto para datos numéricos (tanto continuos como discretos) y datos categóricos. Para *datos numéricos*, estos incluyen

Nombre	Operación
mean()	Promedio aritmético
median()	Mediana muestral
fivenum()	Cinco números resumen
min(), max()	Valores menor / mayor
quantile()	Calcula cuantiles muestrales
var()	Varianza muestral
sd()	Desvío estándar muestral
cov(), cor()	Covarianza / correlación muestral

Estas funciones tomarán como argumentos para el cálculo, uno o más vectores. Además, en general, funcionarán correctamente cuando se les da un data frame como argumento.

Si los datos contienen *cantidades* (como la cantidad de mascotas que tiene cada familia de un grupo de 20 familias), o es de naturaleza *categorical* (como cuando se registra el color de ojos en una muestra de 50 moscas de la fruta), las medidas numéricas anteriores no son de mucha utilidad. Para datos categóricos o discretos, podemos utilizar la función **table()**para resumir un conjunto de datos.

Tomemos el conjunto de datos de R **mtcars** in R para ejemplificar estas funciones. Contiene mediciones sobre 11 aspectos del diseño y performance para 32 automoviles (modelos 1973-74):

```
> data(mtcars) # load in dataset
> attach(mtcars) # add mtcars to search path
> mtcars
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
. . .
```

```
>  help(mtcars) # Del help obtenemos las descripciones de las
variables
```

mpg Miles/(US) gallon – millas por galón

cyl Number of cylinders
disp Displacement (cu.in.)
hp Gross horsepower
drat Rear axle ratio
wt Weight (lb/1000) – peso
qsec 1/4 mile time
vs V/S
am Transmission (0 = automatic, 1 = manual)
gear Number of forward gears –cantidad de marchas hacia adelante
carb Number of carburetors

Este conjunto de datos contiene tanto variables continuas (por ej. **mpg**, **disp**, **wt**) como discretas (por ej. **gear**, **carb**, **cyl**) . Para las variables continuas, podemos calcular:

```
> mean(hp)
[1] 146.6875
> var(mpg)
[1] 36.3241
> quantile(qsec, probs = c(.20, .80)) # 20th and 80th percentiles
20% 80%
16.734 19.332
> cor(wt,mpg) # no sorprende este valor negativo
[1] -0.8676594
```

Para las variables discretas vemos las cantidades resumen:

```
> table(cyl)
cyl
 4  6  8
11  7 14
```

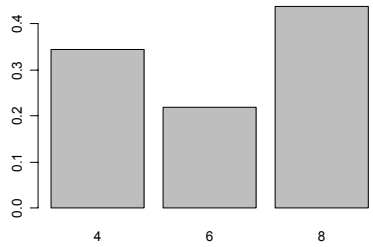
Vemos que hay once vehículos con 4 cilindros, siete con 6 cilindros y catorce tienen 8 cilindros. Podemos transformar estas cantidades en porcentajes (o *frecuencias relativas*) dividiendo por la cantidad total de observaciones.

```
> table(cyl)/length(cyl) # nota: length(cyl) = 32
cyl
      4       6       8
0.34375 0.21875 0.43750

> table(cyl)/length(cyl)*100 # porcentajes
cyl
     4      6      8
34.375 21.875 43.750
```

5.2 Resúmenes gráficos

- **barplot():**
  Para datos discretos o categóricos la función **barplot()** muestra la información dada en un comando "table". Esta función toma como argumento el objeto tabla creado utilizando el comando **table()** descripto anteriormente:

```
> barplot(table(cyl)/length(cyl)) # use frecuencias relativas en
                                  # el eje y
```



Vea ?**barplot** sobre como cambiar el color de relleno, agregar títulos, etc.

• **hist():**

Esta función graficará un histograma que generalmente se utiliza para mostrar datos continuos. Su formato, con las opciones más comunes, es:

```
hist(x,breaks="Sturges",
                  prob=FALSE,main=paste("Histograma de" ,xname))

Argumentos:

x: un vector, a cuyos valores quiere construirse el histograma.

breaks: uno de:

     * una cadena de caracteres(entre dobles comillas) nombrando
     el
        algoritmo para calcular la cantidad de clases

       El valor por defecto de 'breaks' es '"Sturges"': Otros
     nombres
       para los cuales se proveen algoritmos son '"Scott"' y
     '"FD"'

     * un único número indicando la cantidad de clases para el
     histograma

     * un vector indicando los puntos que determinan las clases

En los primeros tres casos el número es únicamente una sugerencia.

prob: variable lógica; si es FALSE, las alturas de los rectángulos
     de cada clase en el histograma representan las frecuencias de
     dichas clases (cantidades); si es TRUE, se grafican las
     frecuencias relativas (proporciones).

main: título del histograma
```

El argumento **breaks** especifica la cantidad de clases (“bins”) para el histograma. Pocas o muchas clases pueden dar una imagen pobre que no caracterice bien la

distribución de los datos. Por defecto, R utiliza la fórmula de Sturges para calcular la cantidad de clases:

$$\lceil 1 + \log_2(n) \rceil$$

donde *n* es el tamaño de la muestra y $\lceil\ \rceil$ es el operador parte entera. Esta función toma un argumento numérico 'x' y devuelve un vector con los menores enteros, no menores que los elementos correspondientes de 'x' (mirar **`ceilieng`** en el **`help`** para ver funciones relacionadas con la parte entera).

Existen otros métodos que consisten en hallar la *longitud de clase* óptima - optimal *bin width* - (la cantidad de clases -number of bins- requerida en este caso sería el rango muestral dividido por la longitud de la clase). La fórmula de Freedman-Diaconis (Freedman and Diaconis 1981) está basada en el rango inter cuartil (*riq*)

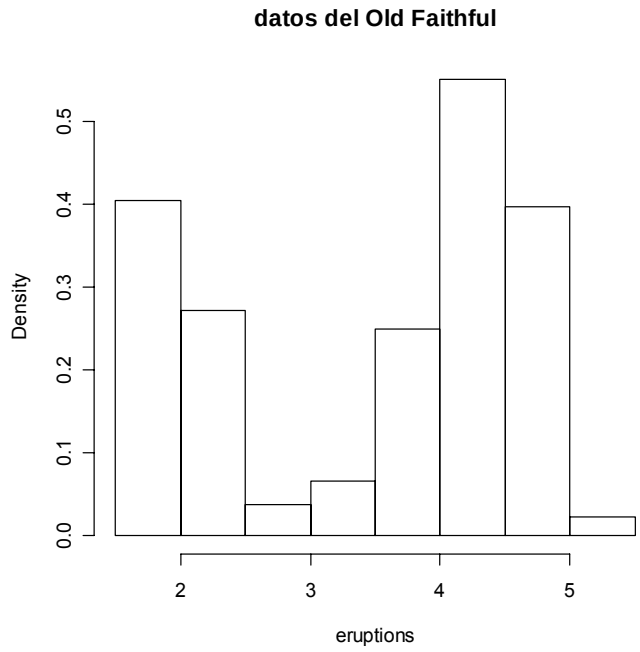
$$2\ riq\ n^{-1/3}$$

y la fórmula propuesta por Scott (1979) está basada en el desvío estándar (*s*)

$$3.49\ s\ n^{-1/3}$$

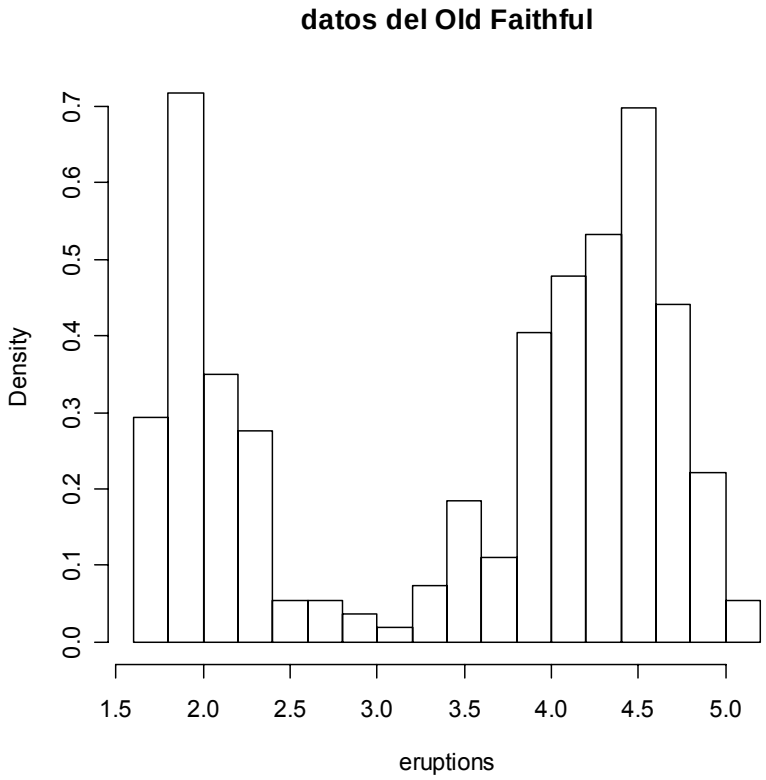
Para ver algunas diferencias consideremos el conjunto de datos de R **`faithful`**. Los datos corresponden al géiser "Old Faithful en el Yellowstone National Park, USA". Es un conjunto de datos famoso que presenta una bimodalidad natural. La variable **`eruptions`** da la duración de la erupción (en minutos) y **`waiting`** es el tiempo entre erupciones:

```
> data(faithful)
> attach(faithful)
>  hist(eruptions, main = "datos del Old Faithful", prob = T)
```



Podemos dar un aspecto un poco diferente al histograma cambiando la cantidad de clases:

```
> hist(eruptions, main = "datos del Old Faithful",
    prob = T, breaks = 20)
```



Vemos que con la opción **`breaks = 20,`** el histograma tiene 18 clases, es la misma cantidad de clases que se obtiene con **`breaks = 18`**.

- **`stem()`**

Esta función construye un diagrama tallo-hoja de texto directamente en la ventana de comandos (**`R Console`**). Se puede utilizar el argumento opcional **`scale`** para controlar la longitud del diagrama

```
> stem(eruptions)

  The decimal point is 1 digit(s) to the left of the |

  16 | 070355555588
  18 | 000022233333335577777777888822335777888
  20 | 00002223378800035778
  22 | 0002335578023578
  24 | 00228
  26 | 23
  28 | 080
  30 | 7
  32 | 2337
  34 | 250077
```

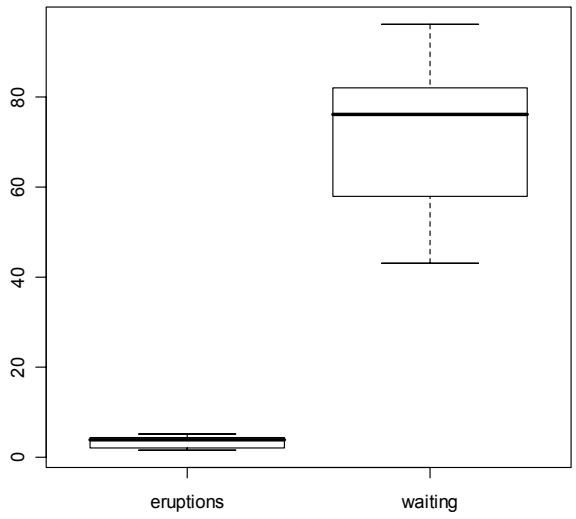
```
  36 | 0000823577
  38 | 2333335582225577
  40 | 000000335778888800233555577778
  42 | 03335555778800233333555577778
  44 | 022223355577800000000233333577788888
  46 | 0000233357700000023578
  48 | 00000022335800333
  50 | 0370
```

• **boxplot()**

Esta función construirá un único boxplot si el argumento que se le ha dado es un único vector, pero si el argumento contiene muchos vectores (o un data frame) se produce un boxplot para cada variable *en el mismo gráfico.*

Para los datos del Old Faithful:

```
    > boxplot(faithful) # casi lo mismo que boxplot(eruptions,
waiting)
```



Vemos que el tiempo de espera (**waiting**) es generalmente mucho mayor y tiene una variabilidad más alta que el tiempo de una erupción (**eruption**). Vea **?boxplot** para agregar títulos, color, orientación, etc.

• **ecdf():**

Esta función crea los valores de la función de distribución empírica (FDE) $F_n(x)$ (empirical distribution function (EDF)). Requiere de un único argumento un vector numérico. Para graficar la FDE de un conjunto de datos contenidos en **x** :

```
    > plot(ecdf(x))
```

• **qqnorm() y qqline()**

Estas funciones se utilizan para validar la normalidad en un conjunto de datos muestrales mediante la construcción de un gráfico de probabilidad normal (NPP) o *q-q* plot normal. La sintaxis es:

```
> qqnorm(x) # crea el NPP para valores guardados en x
> qqline(x) # agrega una línea de referencia al NPP
```

Hemos descripto sólo algunas de las tantas funciones gráficas. Aquí presentamos algunas más:

Nombre	**Operación**
pairs()	Grafica diagramas de dispersión de todos los pares de columnas posibles tomadas de un objeto matrix o dataframe
persp()	Produce gráficos tridimensionales con colores y perspectiva
pie()	Construye un gráfico de torta para datos categoricos o discretos
qqplot()	Gráfico cuantil-cuantil (quantile-quantile plot) para comparar la distribución de dos conjuntos de datos
ts.plot()	Time series plot

5.3 Ejercicios

Utilice el conjunto de datos **stackloss** incorporado al R para:

1. Calcular la media, varianza, y los 5 números resumen de la variable **stack.loss**.
2. Construya un histograma, boxplot y gráfico de probabilidad normal **stack.loss**. ¿Parece adecuado el supuesto de normalidad en esta muestra?