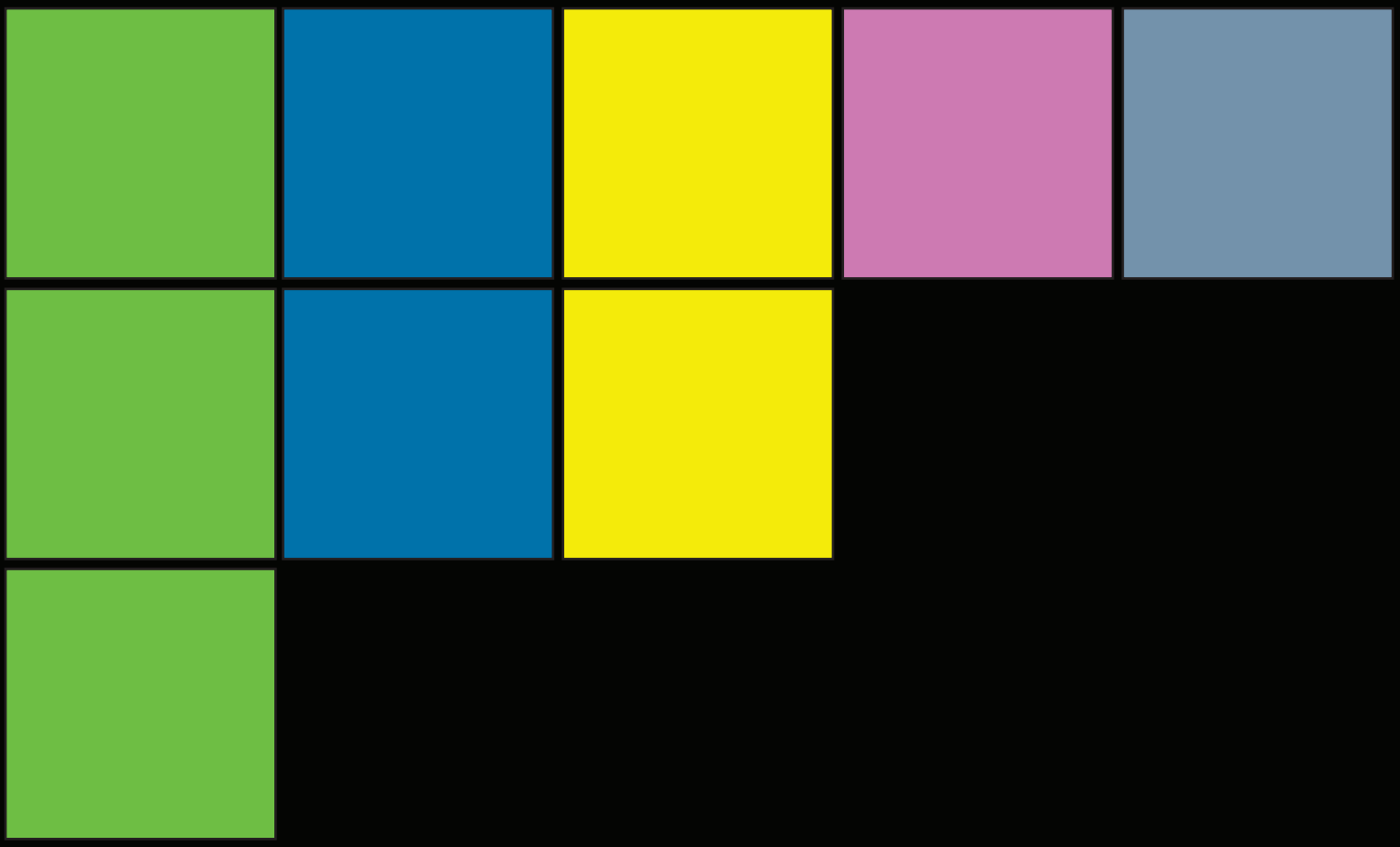


Prácticas de Estadística en R

Ingeniería Técnica en Informática de Sistemas

Manuel Febrero Bande
Pedro Galeano San Miguel
Julio González Díaz
Beatriz Pateiro López

Prácticas de Estadística en R
Ingeniería Técnica en Informática de Sistemas

Manuel Febrero Bande
Pedro Galeano San Miguel
Julio González Díaz
Beatriz Pateiro López

Práctica de Estadística en R

ISBN-13: 978-84-691-0975-1
DL: C 350-2008

Departamento de Estadística e Investigación Operativa
Universidad de Santiago de Compostela

Índice general

Capítulo 1

Introducción

1.1. ¿Qué puedes encontrar en este libro?

Este libro es el resultado de la recopilación del material utilizado en las clases prácticas de la asignatura de Estadística de la titulación de Ingeniería Técnica en Informática de Sistemas de la Universidad de Santiago de Compostela. Estas prácticas se entienden como un apoyo para la comprensión de la materia, además de introducir al alumno en un nuevo lenguaje de programación, R, de gran utilidad en estadística.

Los Capítulos 2, 3 y 4 están dedicados fundamentalmente al manejo de R como herramienta informática, sin profundizar todavía en conceptos estadísticos. Se ha incluido, además de las prácticas, una colección de ejercicios de boletines y exámenes con los que se pretende repasar los comandos básicos que se necesitarán a lo largo del curso: consulta de ayuda en R, importación de archivos, representaciones gráficas, sintaxis, *etc.*

A partir del Capítulo 5, una vez que el alumno ya está más familiarizado con el lenguaje, se desarrollan las prácticas relacionadas con los contenidos de la materia: estadística descriptiva, variables aleatorias, inferencia estadística, *etc.*

El objetivo fundamental de esta recopilación es proporcionar un material completo y estructurado a todos aquellos que deseen introducirse en la estadística a través de la herramienta R. Pretendemos que sirva de ayuda para entender, tanto el lenguaje de programación como los contenidos de la materia y que los diferentes ejemplos propuestos contribuyan a hacer más amena la lectura y comprensión.

1.2. Los conjuntos de datos

A lo largo de las prácticas se ha trabajado con diferentes conjuntos de datos con los que pretendíamos ilustrar los contenidos de la materia. Algunos son conjuntos de datos clásicos, como los *Fisher's iris data*, disponibles en R y otros han sido creados específicamente para la asignatura. Todos ellos se encuentran disponibles en la página web de la materia http://eio.usc.es/eipc1/ASIG.php. De esta manera se podrán reproducir las prácticas y ejercicios incluidos a lo largo de los siguientes capítulos.

Capítulo 2

El software R

2.1. Introducción

R es un entorno especialmente diseñado para el tratamiento de datos, cálculo y desarrollo gráfico. Permite trabajar con facilidad con vectores y matrices y ofrece diversas herramientas para el análisis de datos.

El lenguaje de programación R forma parte del proyecto GNU[1] y puede verse como una implementación alternativa del lenguaje S, desarrollado en AT&T Bell Laboratories. Se presenta como software libre, donde el término software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Se trata de un lenguaje creado específicamente para la visualización y exploración de datos así como para su uso en modelización y programación estadística.

En la web http://www.r-project.org/index.html se encuentra disponible toda la información acerca de R. La instalación de R se realiza a través de la CRAN (Comprehensive R Archive Network). Además, R es un entorno en el que se han ido implementando diversas técnicas estadísticas. Algunas de ellas se encuentran en la base de R pero otras muchas están disponibles como *paquetes* (packages). Estos paquetes están disponibles en la web http://cran.au.r-project.org/.

En resumen, R proporciona un entorno de trabajo especialmente preparado para el análisis estadístico de datos. Sus principales características son las siguientes:

- R proporciona un lenguaje de programación propio. Basado en el lenguaje S, que a su vez tiene muchos elementos del lenguaje C. Sin embargo, la semántica es muy distinta a la de este último. Esto es porque R permite ejecuciones de comandos en línea (compilación y ejecución unidos en un mismo paso), lo cual hace que su semántica esté más próxima a la de un lenguaje de programación funcional.

- Objetos y funciones específicas para el tratamiento de datos.

[1] El proyecto GNU se inició en 1984 con el propósito de desarrollar un sistema operativo compatible con Unix que fuera de software libre. Para más información sobre el proyecto GNU consultar http://www.gnu.org/

- R es software libre. Permite la descarga de librerías con implementaciones concretas de funciones gráficas, métodos estadísticos, algoritmos...

- En su momento el lenguaje S evolucionó por un lado hacia R, software libre con licencia GNU, y por otro hacia S-plus, software comercial y con un entorno gráfico mucho más "amigable". Esto hace que pasar de R a S-plus o al revés sea relativamente sencillo (bajo los dos subyace el mismo lenguaje de programación).

Estas notas pretenden ser una guía esquemática para una introducción superficial a R y su entorno, para profundizar en cualquier punto de ellas se pueden consultar tanto los manuales específicos como la propia ayuda de R (a todo ello se puede acceder desde el menú help).

2.2. Comandos y conceptos básicos

Aquí describimos únicamente la utilidad específica de cada comando, para consultar la sintaxis precisa de cada uno de ellos habrá de echar mano de la ayuda o de los manuales.

Obteniendo ayuda: Para obtener ayuda acerca de, por ejemplo, la función *plot*, se podrían utilizar los siguientes comandos *?plot*, *help(plot)*, *help.search(plot)*. Los dos primeros son equivalentes y el tercero permite realizar búsquedas más avanzadas. Además, también se pueden utilizar los correspondientes elementos del menú help.

Case sensitivity: El lenguaje R es "case sensitive", es decir, distingue entre mayúsculas y minúsculas.

Robustez: El lenguaje R es un lenguaje robusto. En otras palabras intenta, en la medida de lo posible, no dar mensajes de error. Esto en general es cómodo, pero hay que tener cuidado, es posible que al ejecutar un comando no obtengamos ningún error y sin embargo R no esté haciendo lo que nosotros pretendíamos.

Importar código: El comando *source("comandos.R")* permite introducir comandos procedentes de archivos. Útil para cargar funciones elaboradas por el usuario o bien para ejecutar macros y scripts.

Exportar: El comando *sink("archivo")*, nos permite que la salida de los comandos posteriores se almacene en "archivo". Para devolver la salida a la pantalla se usa nuevamente el comando *sink()*. Usado principalmente para guardar salidas de datos en archivos de texto.

Funciones: Para consultar el código de una función en R, ya sea una función propia de R o bien una del usuario, basta con teclear el nombre en la línea de comandos (sin paréntesis ni ningún tipo de argumentos).

Workspace: El comando *getwd()* nos devuelve el directorio de trabajo y el comando *setwd()* nos permite modificarlo. Al iniciar el programa R se abre automáticamente un espacio de trabajo, en él se almacenan todos los datos, funciones... usadas durante esa sesión. En cualquier momento se pueden guardar todos los objetos del espacio de trabajo como un archivo con extensión ".RData" (menú file). Como norma general, las funciones y objetos de datos es mejor guardarlos en archivos propios; esto evita tener objetos innecesarios en el "workspace". La mejor forma de hacer esto será guardando uno o varios objetos en un archivo (de extensión .RData) con el comando *save(objectos, file="nombre.RData")*. Si en sesiones posteriores se necesitan se pueden cargar con la función *load()*.

Librerías: Al iniciar el programa R se cargan por defecto unas librerías básicas. A veces es necesario cargar otras librerías para realizar ciertos análisis, o llamar a algunas funciones "no básicas". Esto se hace a través del comando *library(nombre)*.

Listados: El comando *ls()* nos proporciona un listado de los objetos que hay actualmente en el espacio de trabajo. Para hacer búsquedas de objetos en el espacio de trabajo se pueden utilizar los comandos *apropos()* y *find()*. Además, el comando *search()* nos da una lista de las librerías cargadas (más adelante veremos que no es lo único que lista este comando).

Borrado: Para eliminar uno o varios objetos del espacio de trabajo se usa el comando *rm(objectos)*.

Historial de comandos: Mediante el teclado (flechas) se puede acceder a los últimos comandos ejecutados. Además, el comando *history()* nos devuelve un archivo de texto con los últimos comandos ejecutados.

2.3. Objetos y operaciones básicas

En el lenguaje R se puede trabajar con varias clases de objetos; algunos de ellos son estándar en cualquier lenguaje de programación y otros son objetos específicos de R, objetos pensados para ser manejados con propósitos estadísticos.

2.3.1. Vectores

Son la estructura de datos más sencilla con la que trabaja R. A continuación mostramos varios ejemplos de asignaciones para obtener vectores. R utiliza el operador de asignación "`<-`" (también se puede poner en sentido contrario), en general no se hacen asignaciones con "=".

- **IMPORTANTE:** Hay que acostumbrarse a no usar "=" como operador de asignación en R. Normalmente su uso no devuelve ningún mensaje de error, pero tampoco realiza la asignación deseada. Esto da lugar a los errores más difíciles de depurar (errores lógicos).

`x<-c(2,4,6,8,10)`

Esto nos proporciona un vector formado por los 5 primeros números pares. La función *c()* se usa para concatenar objetos, aunque resulta especialmente cómoda para crear vectores. En el ejemplo anterior `x[4]` nos devolvería el valor 8.

`x<-numeric(50)`

Crearía un vector de 50 componentes, todas ellas 0. También se podrían definir vectores usando otros modos distintos: character, double...

`x<-1:5`

`y<-5:1`

Estas asignaciones devolverían vectores x e y con los 5 primeros números naturales en orden creciente y decreciente respectivamente.

`seq(1,9,by=2)` Nos devolvería los 5 primeros números impares (ir de 1 a 9 con paso igual a 2).

`seq(1,9,lenght=6)` Nos devolvería los números 1.0 2.6 4.2 5.8 7.4 9.0 (ir de 1 a 9 en 6 pasos).

`rep(3,10)` Proporcionaría un vector de 10 coordenadas, todas ellas con el valor 3.

`x<-c("a","e","i","o","u")` Nos devolvería un vector de caracteres con las 5 vocales.

Todos los elementos de un vector han de ser del mismo tipo, es decir todo números, caracteres...Esto en R admite dos excepciones, los valores NA (not avaliable) y NaN (not a number). Estos dos valores son muy importantes en el análisis estadístico: NA porque a veces hay valores perdidos en algunos campos de una muestra y NaN puede ser el resultado de alguna indeterminación. R nos permite hacer operaciones con vectores aunque en alguna de sus componentes tengan NA o NaN.

Operaciones básicas con vectores

Los operadores básicos son $+, -, *, /$, y `^` para la potencia; estos operadores funcionan componente a componente. Por ejemplo, dados los vectores x e y, $x + y$ nos daría el vector obtenido al sumar x e y componente a componente $1/x$ nos devolvería el vector que tiene en cada componente el inverso de la componente de x. Otras funciones básicas implementadas en R son *log*, *exp*, *sin*, *cos*, *tan*, *sqrt*... Además, $sum(x)$ nos devuelve la suma de los elementos de x, $prod(x)$ su producto, $mean(x)$ su media, $var(x)$ su quasivarianza...

En R tenemos los dos valores lógicos: TRUE y FALSE. Los operadores lógicos son `<, <=, >, >=, ==` para igual y `!=`para distinto. Además '`&`' se usa para indicar intersección ("y"), '`|`' indica disyunción ("o") y '`!`' indica la negación. Hay veces que dado un vector nos interesa hacerle algún tipo de filtro para quedarnos con los elementos de este que verifican una cierta propiedad. Por ejemplo, con los comandos:

`x<-c(1,-1,2,-2)`

`x>0`

Obtendríamos el vector: TRUE FALSE TRUE FALSE

`x<-c(1,-1,2,-2)`

`y<-x[x>0]`

El vector y estaría compuesto por los números 1 y 2 (los positivos de x). Otros posibles comandos serían:

`x[1:3]` para quedarnos con los tres primeros elementos de x,

`x[-(1:3)]` para quedarnos con todos los elementos de x salvo los 3 primeros.

`x[c(T,F)]` nos devolvería los elementos en coordenadas impares de x, mientras que el comando `x[c(F,T)]` devolvería los de las coordenadas pares (independientemente de la longitud de x). Esto es debido a que R cuando tiene que operar con dos vectores de distinto tamaño intenta reciclar el pequeño tantas veces como sea necesario hasta igualar el tamaño del grande.

A veces interesa quitar los valores NA y NaN de un vector, esto se haría con el comando

`x<-x[!is.na(x)]` Este comando elimina del vector x todos aquellos elementos que sean NA o NaN, para quitar sólo los NaN se puede usar el comando `x<-x[!is.nan(x)]`. Del mismo modo, `x[is.nan(x)]<-0` reemplazaría todos los NaN de x por 0. La expresión `is.objeto(x)` funciona para la mayoría de los objetos de R.

`which(is.nan(x))` nos devolvería las posiciones de los elementos de x que toman el valor NaN.

2.3.2. Arrays y matrices

Son la extensión natural de los vectores. Un array o una matriz se podrían definir del siguiente modo:

`x <- array(1:20,dim=c(4,5))` genera un array de 4 filas y 5 columnas con los números del 1 al 20. En R el array se llena por columnas (al contrario que en el lenguaje C), es decir, la 1ª columna del array x serían los números del 1 al 4, la 2ª los números del 5 al 8 y así sucesivamente. Lo más importante a la hora de definir un array es especificar el vector de dimensiones.

En el ejemplo anterior, `x[3,2]` nos daría el segundo elemento de la tercera fila de x, `x[,1]` sería la 1ª columna y `x[3,]` la tercera fila.

Aunque la función *array* se puede usar para definir matrices, para este caso particular existe la función *matrix*, que nos permite decir simplemente el número de filas o de columnas de la matriz; no es necesario decir los dos ya que R adapta los datos de la única forma posible. Además, el parámetro opcional `byrow` permite decir si llenamos la matriz por filas o por columnas.

Operaciones básicas con arrays y matrices

Dada una matriz A, la función *t(A)* calcularía la traspuesta de la matriz A. Dado un array de dimensión cualquiera, la función *aperm()* permite obtener trasposiciones del mismo (una trasposición no es más que intercambiar los índices en la matriz, esto en general se hace permutando los índices de columnas, filas...).

Dada una matriz A, *nrow(A)* y *ncol(A)* nos dicen el número de filas y columnas de la matriz A. Dadas dos matrices A y B, el producto de ambas se hace mediante el operador `%*%`. Si A y B tienen la misma dimensión, `A*B` nos devolvería el producto componente a

componente de sus elementos, no el producto matricial. Análogamente *sin(A)* nos daría la matriz que tiene en cada componente el seno del correspondiente elemento de A. Por otro lado, la función *solve* aplicada a una matriz nos calcula su inversa.

Producto exterior de dos arrays

Dados dos arrays A y B, la función *outer(A,B,func)* evalúa *func* para cada posible par que se pueda formar con un elemento de A y otro de B. La dimensión de este nuevo array se obtiene concatenando las dimensiones de A y B (el orden es importante). Esta función tiene una aplicación muy importante a la hora de evaluar funciones bidimensionales y hacer sus correspondientes representaciones gráficas. Nótese que en este momento tenemos definidos tres productos entre arrays: `*` para el producto elemento a elemento, `%*%` para el producto matricial, y *outer* para el producto exterior que acabamos de comentar. Veamos un ejemplo del uso de outer:

```
A<-matrix(1:4,nrow=2)
A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
B<-c(-1,1)
outer(A,B,"*")
, , 1
     [,1] [,2]
[1,]   -1   -3
[2,]   -2   -4
, , 2
     [,1] [,2]
[1,]    1    3
[2,]    2    4
dim(outer(A,B,"*"))
2 2 2
```

Vemos que hemos obtenido un array $2 \times 2 \times 2$, es decir, dos matrices 2×2: una en la que multiplicamos los elementos de A por -1 y otra en la que los multiplicamos por 1. Por otro lado, con el producto elemento a elemento, $*$, tendríamos

```
 A*B
     [,1] [,2]
[1,]   -1   -3
[2,]    2    4
dim(A*B)
 2 2
```

Y con el producto matricial

```
A%*%B
     [,1]
[1,]    2
```

```
[2,]    2
dim(A%*%B)
2 1
```

Factores

A veces nos encontramos con que los datos pueden ser agrupados de acuerdo a un determinado criterio. Imaginemos que tenemos un vector *prov* que nos dice la provincia a la que pertenece cada individuo de un grupo:

```
prov<-c("lu","co","lu","po","co","or","co","lu",
+ "lu","po","co","or","co","co","lu","po")
```

El comando *factor(prov)* convierte este vector en un factor. Es decir un nuevo vector que además contiene la información relativa a los grupos que se pueden hacer con los elementos del vector *prov*.

```
provf<-factor(prov)
provf
[1] lu co lu po co or co lu lu po co or co co lu po
Levels: co lu or po
```

Las funciones *apply* y *tapply*

Las funciones *apply* y *tapply* son muy importantes a la hora de programar códigos eficientes para el manejo de grandes conjuntos de datos. Permiten ahorrarse el uso de muchos bucles, lo cual lleva a un código más legible y, en muchos casos, más eficiente.

La función *apply* permite realizar la misma operación en todas las filas, columnas... de un array simultáneamente. Sólo hay que indicarle la operación a realizar y el índice/índices sobre los cuales ha de realizarla:

```
A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]   -2   -5   -8
[3,]   -3   -6   -9
```

Suma de los elementos de cada una de las filas de A:

```
apply(A,1,sum)
[1]  12 -15 -18
```

Suma de los elementos de cada una de las columnas de A:

```
apply(A,2,sum)
 -4  -7 -10
```

También se puede usar *apply* con funciones más generales:

```
apply(A,1,is.vector)
TRUE TRUE TRUE
```

Pasemos ahora a la función *tapply*. Su función y su sintaxis es similar a la de *apply*. Mientras que *apply* permite hacer fácilmente operaciones sobre elementos agrupados por filas, columnas... la función *tapply* se usa cuando los elementos están agrupados en

factores. Como ejemplo, tomemos el factor *provf* definido anteriormente, en el que hemos agrupado unos individuos según su provincia de procedencia. Tomemos ahora un vector con el número de hijos de cada individuo, digamos

```
hijos<-c(1,3,2,2,3,5,3,2,1,2,3,1,2,4,1,0)
```

La función *tapply* nos permite calcular, por ejemplo, el número medio de hijos en cada provincia. A *tapply* le damos primero el vector *hijo*, después el factor *provf*, que describe cómo queremos agrupar el vector *hijos*, y, finalmente, le decimos la función a aplicar sobre cada grupo, *mean*:

```
tapply(hijos,provf,mean)
     co       lu       or       po
 3.000000 1.400000 3.000000 1.333333
```

Al igual que la función *apply*, la función *tapply* se puede usar con funciones mucho más generales que la función *mean*.

Listas y data frames

Por último, presentamos dos clases de datos todavía más generales: las listas y los data frames.

Una lista es un objeto cuyas componentes pueden ser arrays, listas, Data Frames, variables lógicas,... y cualquier combinación de éstos (ésto nos permite almacenar juntos datos de distinta naturaleza como nombre, edad, ingresos, trabajos anteriores...) Los distintos elementos de la lista no han de ser necesariamente del mismo tipo. Por ejemplo `Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))` devuelve una lista con formada por 4 objetos, dos variables de tipo carácter, un valor numérico y un vector de tres componentes. Para referenciar a cada objeto de la lista se hace con `Lst[[i]]`, de este modo `Lst[[1]]="Fred"` y `Lst[[4]][3]=9`. Al trabajar con listas puede ser más cómodo usar los nombres de cada objeto de la lista, de este modo `Lst$wife="Mary"`. Para saber los nombres de los objetos que componen la lista se usa `names(Lst)`.

Los data.frames (campos de datos) son el objeto más habitual para almacenar datos. La forma de pensar en un data.frame es considerar que cada fila representa a un individuo de una muestra y el correspondiente valor para cada columna se corresponde con la medición de alguna variable para ese individuo (fila-individuo, columna-variable). Si por ejemplo tenemos 100 alumnos y las notas numéricas de ellos en cada examen junto con la calificación final (tipo carácter), esto podría en un data.frame de 100 filas con una columna por cada examen y una columna más para la calificación final.

Por último, a veces resulta latoso trabajar con un data.frame o una lista con muchos campos anidados (muchos subniveles). Por ejemplo, en el caso de la lista antes definida `Lst$wife` se usaba para referirse al 2º objeto de la lista, `Lst$name` al primero y así con los demás. Esta notación se puede hacer menos pesada mediante el comando *attach*(Lst), este comando hará que `Lst` "suba un nivel en el workspace" y a partir de ese momento podremos utilizar, por ejemplo, `child.ages` en vez de `Lst$child.ages` para acceder al vector de edades. Al hacer esto hay que tener cuidado de que en el espacio de trabajo no existan ya objetos con el mismo nombre que alguno de los objetos/columnas de la

lista/data.frame. Para volver a la situación anterior se usa el comando *dettach(Lst)*. En su momento dijimos que el comando *search()* permitía ver el número de librerías cargadas en el espacio de trabajo, del mismo modo permite ver los objetos a los que se ha "subido un nivel" con *attach*.

Dado un objeto se puede consultar su tipo, su clase (numérico, carácter, lista, matriz...) a través de las funciones *mode*, *class* o bien *typeof*; si bien son muy parecidas, no son equivalentes; dependiendo de la situación puede interesar más usar una que otra. La clase de un objeto es algo más específico que su modo, si un objeto no tiene clase, *class()* nos devolverá lo mismo que *mode()*. Además, también existe la función *attributes*, que nos proporciona características de un objeto tales como dimensión, clase, nombres de columnas...

Dado un objeto de un cierto tipo, es posible forzarlo a otro distinto (siempre que esto tenga sentido para el objeto en cuestión). Esto es posible mediante la familia de funciones *as.objeto()*. Por ejemplo dado un data.frame, Data, la función *as.matrix(Data)* lo convierte en una matriz si esto es posible. Un uso indebido de estas imposiciones de tipo puede llevar a resultados no deseados (el que no haya habido mensaje de error no quiere decir que la conversión se haya hecho como nosotros queríamos).

Para realizar operaciones numéricas con data.frames o listas, lo más conveniente es convertir a matriz, array o vector la parte con la que se desea operar y después hacer las operaciones.

En su momento vimos como definir un vector a través de la función *c()*, es importante tener en cuenta que esta función se puede utilizar para concatenar cualquier tipo de objetos, no sólo variables numéricas, enteras... De la misma familia son las funciones *cbind* y *rbind* que permiten combinar una serie de objetos bien por columnas o por filas. Habitualmente se usan para construir una matrices a partir de vectores.

2.4. Procedimientos gráficos

El lenguaje R dispone de varias funciones preparadas para la representación gráfica de datos. Estas funciones se dividen en dos grandes grupos:

Gráficos de alto nivel: Crean un nuevo gráfico en la ventana de gráficos.

Gráficos de bajo nivel: Permiten añadir líneas, puntos, etiquetas... a un gráfico ya existente.

Gráficos de alto nivel

De entre todos los gráficos de este tipo destaca la función *plot*, que tiene muchas variantes y dependiendo del tipo de datos que se le pasen como argumento actuará de modos distintos. Lo más común es *plot(x,y)* para representar un diagrama de puntos de y frente a x. Otras funciones de alto nivel importantes son, por ejemplo, *hist* para dibujar histogramas, o *persp* para representaciones tridimensionales (superficies).

Parámetros más importantes comunes a la mayoría de gráficos de alto nivel:

add=TRUE Fuerza a la función a actuar como si fuese de bajo nivel (intenta superponerse a un gráfico ya existente). Hay que tener cuidado, no sirve para todas las funciones.

type Indica el tipo de gráfico a realizar, cabe destacar `type="p"` para representar puntos (opción por defecto), `type="l"` para representar líneas, `type="b"` para representar los puntos unidos por líneas.

Gráficos de bajo nivel

Son de gran utilidad para completar un gráfico, compararlo con otro superponiendo ambos... Destacan los siguientes:

lines: Permite superponer nuevas funciones en una gráfica ya existente.

points: Permite añadir puntos.

legend: Para añadir una nueva leyenda.

text: Añade texto.

Parámetros más importantes comunes a la mayoría de gráficos, tanto de alto como de bajo nivel (para una descripción completa de estos y otros parámetros *help(par)*):

pch: Indica la forma en que se dibujaran los puntos.

lty: Indica la forma en que se dibujaran las líneas.

lwd: Ancho de las líneas.

col: Color usado para el gráfico (ya sea para puntos, líneas...)

font: Fuente a usar en el texto.

Una vez que en la ventana de gráficos tenemos algo representado, existen dos funciones que nos permiten trabajar interactivamente sobre dicha ventana

- La función *locator()* sitúa el cursor en la ventana de gráficos, cada vez que pulsemos el botón izquierdo del mouse nos devolverá las coordenadas del punto en el que hayamos marcado.

- La función *identify()* nos permite marcar uno de los puntos del gráfico (con el ratón, igual que antes), y nos devuelve la componente de los vectores representados que dio lugar a ese punto. Muy útil en estadística para identificar outliers.

2.5. Programando en R

La principal ventaja de R (y S-plus) sobre la mayoría del software estadístico es que permite combinar sus librerías especializadas con un lenguaje de programación propio. Esto resulta mucho más potente que, por ejemplo, las macros en SPSS. A continuación veremos las principales características de la programación en R.

Agrupando comandos: R permite escribir varios comandos en la misma línea. Esto se hace poniendo un marcador de final de comando antes de empezar el siguiente. El marcador utilizado en R es ";".

Expresiones: Una serie de comandos escritos entre llaves forman una expresión, el resultado de la misma será el de su último comando; la estructura general de una expresión es `{cmd_1; cmd_2; ... cmd_n}`. Puede haber expresiones anidadas, que se van evaluando de dentro hacia fuera. Son el componente principal de funciones, bucles...

Ejecución condicional: Como en todos los lenguajes de programación, esto se hace con las sentencias *if*. Estas sentencias en R tienen la siguiente sintaxis:

```
if (expr_1) {expr_2}
else if (expr_3) {expr_4}
else {expr_5}
```

donde `expr_1` y `expr_3` deben devolver un valor lógico. En R también se pueden construir sentencias *if* anidadas (no es más que anidar expresiones). La sentencia condicional *switch* también está definida en R.

Bucles R admite los bucles *for*, *repeat* y *while*, su sintaxis es la siguiente:

***for*:** `for (name in expr_1) expr_2`. Por ejemplo, la sintaxis para un bucle que mueva un índice i desde 1 hasta n es:

```
for (i in 1:n) {expr}
```

***repeat*:** `repeat expr`. En este caso hay que asegurarse de que en algún momento de `expr` se llega a un `break/return` que nos saca del bucle.

***while*:** `while(cond) expr`.

Depurando el código: Las funciones *print()* y *cat()* nos permiten imprimir datos en pantalla durante la ejecución de un programa. Útil tanto para presentar la salida de resultados como para depurar el código.

Funciones

El lenguaje R permite al usuario definir sus propias funciones. El esquema de definición de una función es muy sencillo, aunque no por ello deja de ser potente

`nombre_func <- function(arg_1, arg_2, ...) expr`

Si queremos que por ejemplo `arg_2` tome por defecto el valor '`val`' bastaría escribir

`nombre_func <- function(arg_1, arg_2=val, ...) expr`

Todo argumento que no tenga un valor por defecto será obligatorio. Una función tomará el valor de la expresión, es decir, el valor del último comando ejecutado en `expr` (que no tiene por qué ser el último comando de `expr`, la sentencia *return* puede ser utilizada para devolver un valor por el medio de una expresión y terminar la ejecución de la función).

Las funciones en R pueden ser recursivas; una función puede llamarse a si misma.

En una función en R se distinguen los siguientes tipos de variables:

Parámetros formales: Son los argumentos de la función, cualquier modificación que se haga sobre los mismos se pierde al salir de la función. Si se quiere que un cambio permanezca, la asignación debe hacerse con el operador `<<-`.

Variables locales: Aparecen en la función al serles asignado algún valor, desaparecen al salir de la función.

Variables libres: Son variables que aparecen en una función sin que sean parámetros ni se les haya asignado previamente ningún valor. R busca de dentro de la función hacia fuera (podemos tener funciones anidadas) hasta que encuentra alguna variable que con su nombre. Esto se conoce como "alcance lexicográfico" y es una de las principales diferencias entre R y S-plus (este último buscaría directamente una variable global). Hay que ser muy cuidadoso con el uso de este tipo de variables.

2.6. Librerías

Al iniciar una sesión en R se cargan una serie de librerías básicas, pero en muchas ocasiones es necesario cargar librerías específicas que tengan implementadas algunas funciones concretas. A continuación listamos las librerías que podremos llegar a usar a lo largo de la asignatura:

Librerías básicas (se cargan por defecto)

base: Contiene las bases de R.

ctest: Librería para realizar test estadísticos clásicos.

methods: Librería para trabajar en R con métodos y clases (no la usaremos).

modreg: Trae implementadas técnicas para regresiones.

nls: Librería para llevar a cabo regresiones no lineales.

mva: Análisis multivariante.

ts: Series de tiempo.

Librerías específicas

MASS: Librería bastante variada, tiene desde funciones avanzadas para la realización de histogramas, hasta transformaciones box-cox y funciones para el análisis discriminante.

lattice: Librería con gran cantidad de funciones gráficas implementadas.

scatterplot3D: Librería para hacer gráficos de dispersión en 3D.

KernSmooth: Funciones para inferencia no paramétrica a través de estimadores tipo núcleo.

foreign: Librería para importar datos, principalmente de SPSS.

RODBC: Importar datos de bases usando ODBC.

stepfun: Librería para representar funciones escalonadas (funciones de distribución empíricas).

gregmisc: Herramientas estadísticas varias.

DAAG: Herramientas para el análisis gráfico de datos.

2.7. Importando datos

Ya hemos resaltado anteriormente que R es un entorno pensado para el análisis de datos, por tanto es importante ser capaces de importar datos externos, a continuación vamos a ver como importar datos de SPSS, Excel y archivos de texto:

Teclado: Además de las fomas vistas hasta ahora para introducir datos por teclado existen otras dos. La primera es la función *scan()*; después de ejecutar la función hay que introducir los valores del vector separados por espacios. La segunda (mucho más cómoda), consiste en crear un vector, matriz... inicializado a 0 y después aplicarle la función *data.entry()* para abrir una hoja que nos permite editar fácilmente el conjunto de datos.

R: El programa R trae con su instalación múltiples conjuntos de datos. Para ver un listado de los mismos basta teclear *data()*, para cargar uno concreto se ha de usar el comando *data(name)*.

Excel: Hay que cargar la librería *RODBC*, se guarda el archivo excel en el directorio de trabajo de R y se abre la conexión a la base de datos:

```
conex<-odbcConnectExcel("archivo.xls")
```

Esto almacena en `conex` las referencias a las hojas del archivo excel. El comando `sqlTables(conex)` nos proporcionaría las tablas disponibles, para cargar una de ellas bastaría escribir el comando

```
tabla <- sqlFetch(conex, "Nombre_tabla")
```

SPSS: Se necesita cargar la librería *foreign*. Después obtenemos una lista con los datos del archivo spss con la ayuda del comando `read.spss("archivo.sav")` . El argumento `to.data.frame=TRUE` nos permitiría obtener un data.frame en vez de una lista.

.txt: Si un conjunto de datos viene guardado en un archivo .txt se pueden leer con la función *read.table* que nos devuelve un data.frame. Entre sus argumentos se puede decir cuál ha de ser el separador de campo, si tiene encabezado o no, así como alguna otra especificación más.

Capítulo 3

Un ejemplo para aprender R. El Hitori

3.1. Introducción

El Hitori es un rompecabezas matemático del mismo tipo que el Sudoku. El juego consiste en una matriz 6×6 (existen versiones con matrices de mayor dimensión) con las cifras del 1 al 6. A continuación se muestra un ejemplo.

2	2	4	3	4	1
6	5	5	1	1	3
3	5	6	1	4	3
5	1	1	1	6	2
3	4	3	2	5	1
4	3	6	6	2	5

Figura 3.1: Ejemplo de Hitori.

El objetivo del juego es tachar casillas de forma que no aparezcan dos números repetidos en la misma fila o columna. Además, no puede haber dos casillas tachadas pegadas. Por lo tanto el juego se reduce a las siguientes dos reglas:

- **Regla 1:** No puede haber dos números iguales en una misma fila o columna.
- **Regla 2:** No puede haber dos casillas tachadas juntas (si una casilla está tachada no se podrán tachar las casillas situadas a su izquierda, derecha, arriba o abajo).

3.2. Práctica en R

La práctica consiste en programar con R el juego del Hitori. El programa debe:

1. Leer el tablero de juego de un fichero de datos.
2. Pedir al jugador el valor de la fila y la columna en la que desea jugar. El programa avisará si la casilla no es válida.
3. En cada jugada el programa permitirá elegir entre:
   - Tachar una casilla.
   - Recuperar una casilla tachada.
4. Se mostrará en cada jugada un gráfico con el tablero, señalando las casillas que están tachadas en cada jugada.

3.3. El código

El programa consta de una función principal *juega.hitori* , en la que se establecen las reglas del juego y una función *dibujatablero* que permite dibujar en cada jugada el tablero con las casillas tachadas. La programación se ha hecho para una matriz 6×6 pero se puede generalizar para cualquier dimensión.

3.3.1. Función para dibujar el tablero

```
# Esta funcion dibuja el tablero del Hitori.
# Se marcan con una cruz las casillas tachadas.
dibujatablero<-function(hitori,tachada){
  plot(1,1,col=0,xlim=c(0,6),ylim=c(0,6),axes=F,xlab="",ylab="")
  for(i in 0:6){
    lines(c(0,6),c(i,i),lwd=2)
    lines(c(i,i),c(0,6),lwd=2)
  }
  for (i in 1:6){
    for(j in 1:6){
      text(j-0.5,7-i-0.5,hitori[i,j],col=1,cex=2)
      if(tachada[i,j]==1){
        text(j-0.5,7-i-0.5,"x",cex=5,col=2)
      }
    }
  }
}
```

3.3.2. Función principal

```
juega.hitori<-function(){
  hitori<-read.table("hitori.txt")  # Devuelve data.frame
  hitori<-as.matrix(hitori)
```

```
  tachada<-matrix(0,nr=6,nc=6)
  dibujatablero(hitori,tachada)
  juega=T
  while(juega){
    jugamos<-menu(c("Tachar casilla","Recuperar casilla"))
    cat("Elija fila y columna\n")
    pos<-scan(n=2)
# Si la posicion no existe paramos
    if (sum(pos<c(1,1)| pos>c(6,6))>0){
      cat("La posición no es válida\n")
    }
    else{
# SI QUEREMOS TACHAR CASILLA
      if (jugamos==1){
# Cómo sabemos que casillas ya estan tachadas?
        if (tachada[pos[1],pos[2]]==1){
          cat("La posición elegida ya estaba tachada\n")
        }
# Cómo sabemos que la casilla elegida se puede tachar?
# Tenemos que ver que no tiene  ninguna casilla tachada pegada
        else{
# Determino en que filas miro
          if (pos[1]==1){
            miro.filas=2
          }
          else if (pos[1]==6){
            miro.filas=5
          }
          else{
            miro.filas=c(pos[1]-1,pos[1]+1)
          }
# Determino en que columnas miro
          if (pos[2]==1){
            miro.col=2
          }
          else if (pos[2]==6){
            miro.col=5
          }
          else{
            miro.col=c(pos[2]-1,pos[2]+1)
          }
# Compruebo que las casillas adyacentes no estan tachadas
          if (sum(tachada[pos[1],miro.col])>0|sum(tachada[miro.filas,pos[2]])>0){
```

```
            cat("No se puede tachar la casilla seleccionada\n")
          }
          else{
            tachada[pos[1],pos[2]]=1
          }
        }
      }
# SI QUEREMOS RECUPERAR CASILLA
      else if (jugamos==2){
        if (tachada[pos[1],pos[2]]==0){
          cat("La posición elegida no estaba tachada\n")
        }
        else {
          tachada[pos[1],pos[2]]=0
        }
      }
      dibujatablero(hitori,tachada)
# SE ACABO EL JUEGO?
      fila<-numeric()
      col<-numeric()
# Miro en todas las filas y en todas las columnas si hay numeros
# que se repiten
      for (i in 1:6){
        fila[i]=sum(table(hitori[i,tachada[i,]==0])>1)
        col[i]=sum(table(hitori[tachada[,i]==0,i])>1)
      }
      if (sum(fila,col)>0){
        cat("Se repiten numeros\n")
      }
      else{
        cat("ENHORABUENA!!!\n")
        juega=F
      }
    }
  }
}
```

Capítulo 4

Ejercicios para practicar con R

4.1. Obteniendo ayuda en R

Recuerda que existen dos maneras de obtener ayuda en R. Lo puedes hacer a través del menú o de la ventana de comandos.

1. La función *mean* calcula la media aritmética de una serie de valores. ¿Cuáles son los argumentos de dicha función?

2. Dado un conjunto de valores, sabemos que existe una función en R que calcula el máximo, pero no recordamos su nombre. ¿Serías capaz de encontrarla a través de la ayuda? *Recuerda que la ayuda está en inglés.*

3. Queremos utilizar el comando *help.search* para obtener información sobre la función *plot*. El resultado es el siguiente:

```
> help.search(plot)
Error in help.search(plot) : argument 'pattern' must be
a single character string
```

   ¿Por qué nos da error? ¿Cómo lo escribirías de forma correcta?

4. Hemos visto que la función *apropos* nos permite realizar búsquedas de objetos. En la ayuda de dicha función aparecen diferentes ejemplos de cómo usarla. Siguiendo esos ejemplos:

   a) Busca los objetos cuyo nombre empiece por *var*.

   b) Busca los objetos cuyo nombre tenga de 5 a 7 caracteres.

5. ¿Cuáles son las librerías cargadas por defecto en R?

4.2. Directorio de trabajo ("workspace")

1. ¿Cuál es el directorio de trabajo por defecto de R?

2. Crea una carpeta *EjerciciosR* en la ubicación que prefieras de tu ordenador y establécela como directorio de trabajo de R. Comprueba que efectivamente lo has hecho bien con el comando *getwd()*.

3. Crea un archivo *ejemplo.R* con el siguiente código:

   ```
   area.rectangulo<-function(base,altura){
     return(base*altura)
   }
   ```

   La función *area.rectangulo* simplemente devuelve el área de un rectángulo dadas su base y altura. Ejecuta la función en R y calcula con ella el área de un rectángulo de base 8 cm y altura 3 cm. Guarda el archivo en el directorio de trabajo que has creado en el ejercicio anterior. Supongamos que la próxima vez que ejecutamos R queremos volver a utilizar la función *area.rectangulo*. ¿Cómo lo harías? Verás que si simplemente tecleas

   ```
   > area.rectangulo(4,5)
   ```

   lo que obtienes será un mensaje de error del tipo

   ```
   Error: couldn't find function "area.rectangulo"
   ```

4. Las edades de un grupo de amigos son 27, 23, 29, 24 y 31 años. Crea un vector *edades* con estos datos y calcula su media, de forma que la salida se guarde en un fichero llamado *amigos*.

   Vuelve a calcularla pero de manera que ahora el resultado salga por pantalla.

5. ¿Cómo obtienes el archivo de texto con los últimos comandos ejecutados?

4.3. Objetos y operaciones básicas

4.3.1. Operaciones con vectores

1. Hemos visto diferentes formas de definir vectores con R. Supongamos que queremos definir el vector $x = (1, 2, 3, 4, 5)$. Comprueba que las siguientes formas son equivalentes.

   ```
   > x<-c(1,2,3,4,5)
   > x<-1:5
   > x<-seq(1,5)
   ```

2. Define el vector $y = (1, 3, 5, 7)$ utilizando la función *c()*. ¿Cómo lo harías con la función *seq()* ?. *Recuerda que si tienes alguna duda sobre cómo se definen las funciones siempre puedes consultar la ayuda.*

3. Define los siguientes vectores. Intenta hacerlo de diferentes formas.

   a) $x = (8, 7, 6, 5)$

   b) $y = (3, 3, 3, 3, 3, 3, 3, 3, 2, 2)$

   c) $z = (1,\ 1.75,\ 2.5,\ 3.25,\ 4)$

4. Aunque pensamos en vectores como conjuntos de números, un vector en R no es más que celdas contiguas conteniendo datos. Estos datos deben ser del mismo tipo, pero no necesariamente números. Podemos construir así vectores de tipo *logical* o vectores de tipo *character* entre otros. Por ejemplo, hemos creado el vector *chica*. El resultado es

   ```
   > chica
   [1] "Ana"      "Maria"    "Natalia"  "Almudena"
   ```

   ¿Cómo ha sido definido dicho vector?

5. En muchas ocasiones nos interesa hacer referencia a determinadas componentes de un vector. Hemos visto que para ello utilizaremos los corchetes [ ] . Crea el vector $x = (2, -5, 4, 6, -2, 8)$. A partir de dicho vector define:

   a) $y = (2, 4, 6, 8)$. Así definido y es el vector formado por las componentes positivas de x.

   b) $z = (-5, -2)$. Así definido z es el vector formado por las componentes negativas de x.

   c) $v = (-5, 4, 6, -2, 8)$. Así definido v es el vector x eliminada la primera componente.

   d) $w = (2, 4, -2)$. Así definido w es el vector x tomando las componentes impares.

6. La función logaritmo sólo está definida para valores positivos. Construye el vector $x = (3, log(-15), 5)$. ¿Qué ocurre? Utiliza la función *is.nan()* y recicla x de forma que sea un vector sin componentes NaN.

7. Sabemos que para sumar vectores éstos deben tener la misma longitud. Sin embargo R trabaja de manera distinta. Define los vectores $x = (1, 2, 3, 4, 5, 6)$, $y = (7, 8)$, $z = (9, 10, 11, 12)$. Calcula:

   a) $x + x$

   b) $x + y$. ¿Qué ha hecho R?

   c) $x + z$. Ahora R da un warning pero aun así nos da un resultado. ¿Cómo lo ha calculado?

4.3.2. Arrays y matrices

1. Define el vector $x = (1, 2, 3, 4, 5, 6)$. A partir de dicho vector se han construido las matrices $m1$, $m2$, $m3$, $m4$

```
> m1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> m2
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> m3
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

> m4
     [,1] [,2] [,3]
[1,]    1    4    1
[2,]    2    5    2
[3,]    3    6    3
```

   Todas las matrices se han definido a partir de *matrix(x,...)*. Intenta reproducir el código necesario para obtener cada una de ellas. *Recuerda que puedes consultar la ayuda.*

2. ¿Qué ocurre cuando definimos una matriz en R y sólo especificamos el número de filas o el número de columnas? ¿Qué ocurre cuándo los datos no se corresponden con la dimensión de la matriz que queremos definir? Compruébalo ejecutando los siguientes comandos:

```
>matrix(1:6,nrow=2)
>matrix(1:6,nrow=4)
>matrix(1:6,nrow=4,ncol=4)
```

3. ¿Cuál es la diferencia entre `*`, `%*%` y *outer()* ? Compruébalo con las matrices

$$A = \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} \text{ y } B = \begin{pmatrix} 3 \\ 8 \end{pmatrix}$$

4. Sean A una matriz 2×3, B una matriz 3×4 y C una matriz 2×3. ¿De qué tipo y dimensión serán los objetos obtenidos de los siguientes comandos de R? ¿Alguno de los comandos produce mensajes de error? ¿Por qué?

   a) `A*B`

   b) `outer(A,B)`

   c) `A+2`

   d) `A%*%B`

   e) `exp(B)`. Nota: *exp()* es la función exponencial.

   f) `A*C`

   g) `A%*%C`

4.3.3. Listas, data frames y factores

1. Un grupo de amigos está formado por Ana de 23 años, Luis de 24 años, Pedro de 22, Juan de 24, Eva de 21 y Jorge de 22 años. Crea los vectores correspondientes a *nombre*, *edad* y *sexo*. (Usa la codificación M=mujer, H=hombre). Convierte el vector *sexo* en un factor *sexf*. ¿Cuáles son los niveles de dicho factor?

2. Con los datos anteriores hemos creado el dataframe *amigos*. El resultado es:

```
> amigos
  nombre edad sexo
1    Ana   23    M
2   Luis   24    H
3  Pedro   22    H
4   Juan   24    H
5    Eva   21    M
6  Jorge   22    H
```

   ¿Cuál es el código de R da como resultado esta salida?

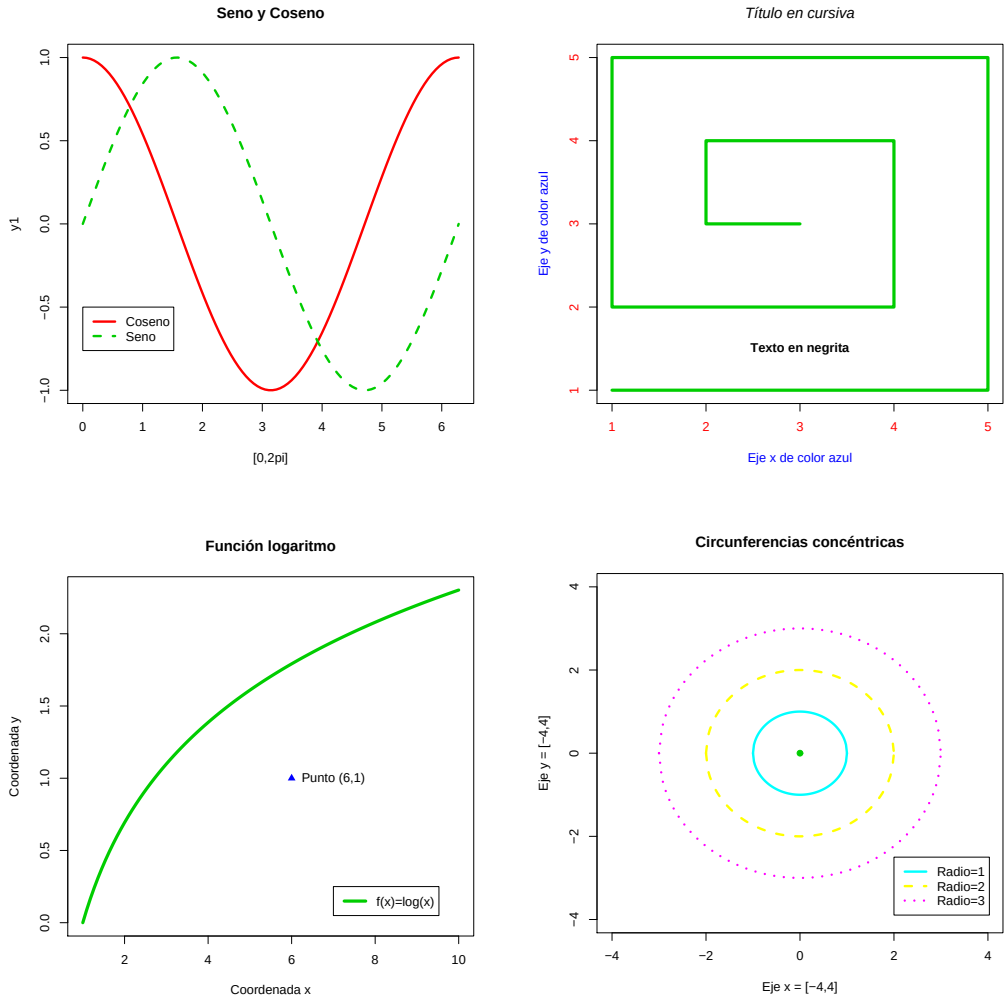
3. ¿Cuál es la edad media de las chicas del grupo? ¿Y la de los chicos? Intenta responder a ambas preguntas con un único comando.

4.4. Gráficos

1. Queremos representar gráficamente la función coseno en el intervalo $[0, 2\pi]$. Para ello creamos el vector x de la siguiente forma: `x<-seq(0,2*pi,length=100)`. ¿Cuál es la diferencia entre las gráficas obtenidas por los siguientes comandos?

   `>plot(cos(x))` y `>plot(x,cos(x))`

2. Intenta reproducir en lo posible las siguientes gráficas. Si tienes cualquier duda consulta la ayuda de las funciones *plot* y *par*.



4.5. Programando en R

1. Diseña funciones en R en las que utilices los distintos tipos de bucles (*for, while...*) así como sentencias condicionales.

2. Escribe una función *practicas* en R que, dado el grupo (A,B,C,D) te devuelva el horario de la clase de prácticas y el aula. La salida será similar a la siguiente:

```
> practicas("D")
La clase de prácticas del grupo D es el miercoles
a las 17:00h en el aula I4
> practicas("d")
La clase de prácticas del grupo d es el miercoles
```

```
a las 17:00h en el aula I4
> practicas("F")
Error in practicas("F") : El grupo introducido no es correcto
```

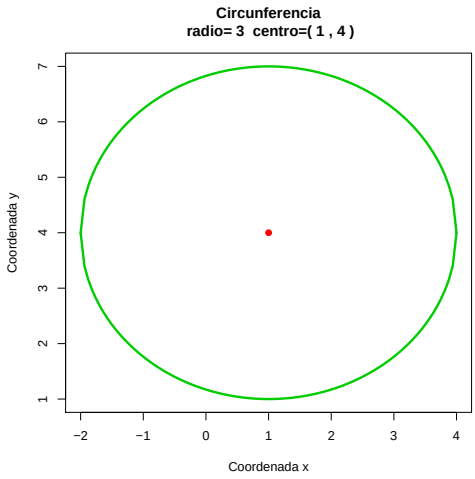
3. Escribe una función que calcule el factorial de un número. Recuerda que el factorial está definido para valores enteros no negativos como:

$$n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$$

   La función debe dar mensaje de error en caso de introducir un valor que no sea entero no negativo.

4. Crea una función *circ* en R que dibuje una circunferencia de radio r y centro c. Debe dar un mensaje de error si se introduce un valor negativo para el radio y la salida debe ser parecida a la que se muestra en la figura.

```
>circ(r=3,centro=c(1,4)).
```



5. Escribe una función que realice la suma de todos los números pares entre 2 y 1000.

6. Escribe una función que realice el producto de todos los números pares y también de todos los números impares comprendidos entre 1 y 200.

7. Escribe una función *producto* que lea dos matrices desde un fichero de texto y ofrezca al usuario la posibilidad de elegir entre realizar el producto matricial o el producto componente a componente de ambas, siempre que las dimensiones de las matrices lo permitan.

8. Escribe una función *edades* que, dado el día mes y año del nacimiento de una persona, nos devuelva la edad que tiene en el momento de la ejecución de la función.

9. **Sudoku:** El Sudoku es un rompecabezas matemático de colocación que se popularizó en Japón en 1986. El objetivo es rellenar una cuadrícula de 9×9 celdas (81 casillas) dividida en subcuadrículas de 3×3 con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas de las celdas. No se debe repetir ninguna cifra en una misma fila, columna o subcuadrícula.

0	6	0	1	0	4	0	5	0
0	0	8	3	0	5	6	0	0
2	0	0	0	0	0	0	0	1
8	0	0	4	0	7	0	0	6
0	0	6	0	0	0	3	0	0
7	0	0	9	0	1	0	0	4
5	0	0	0	0	0	0	0	2
0	0	7	2	0	6	9	0	0
0	4	0	5	0	8	0	7	0

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Figura 4.1: Ejemplo de Sudoku y solución.

Escribe un programa en R para jugar al Sudoku. El programa debe:

a) Leer el tablero de juego de un fichero de datos.

b) Pedir al jugador el valor de la fila y la columna que desea rellenar y el número con el que desea rellenarla. El programa avisará si la casilla no es válida y si el número elegido no es válido.

c) En cada jugada el programa permitirá elegir entre:

- Seguir jugando.
- Borrar una casilla (siempre que no sea una de las casillas fijas).
- Salir del juego.

d) Mostrar en cada jugada un gráfico con el tablero lo más parecido posible al que se muestra en el ejercicio.

10. Utiliza la función *apply* para comprobar que en una matriz que sea solución de un sudoku como los del ejercicio anterior todas las filas y columnas suman lo mismo.

4.6. Ejercicios de exámenes

1. **(Examen 05)** Escribe una función que dados dos vectores numéricos realice el siguiente cálculo: Contar cuántos números estrictamente positivos aparecerían como resultado de multiplicar (uno por uno) todos los elementos del primer vector por todos los elementos del segundo.

   Dado un vector x, length(x) nos devuelve el número de elementos de x.

2. **(Examen 05)** Escribe una función que, dados un vector numérico x y un número natural M, realice el siguiente cálculo: Sumar una a una las componentes del vector x y detenerse una vez que pasemos de M. En este momento, devolver al usuario lo siguiente:

   a) La suma obtenida.

   b) Un vector que contenga los elementos de x que hemos usado para obtener dicha suma.

   La forma más cómoda de devolver varios valores es a través de una lista:

   ```
   return(list(suma=variable_suma,vector=vector_elementos_usados))
   ```

3. **(Examen 05).** Importa de la página http://onlae.terra.es/indexp.htm los datos correspondientes a los resultados de la primitiva en el 2005 (hasta el 5 de marzo). Importar sólo desde la columna de recaudaciones en adelante. Después calcula lo siguiente:

   a) Media de acertantes por categoría. (media por columnas)

   b) Media de acertantes por jornada. (media por filas)

   c) Media de acertantes por categoría los sorteos con bote.

   d) Total de dinero recaudado.

   e) Total repartido en premios.

4. **(Examen 05).** Escribe una función que dado un número compruebe si es natural o no y además:

   - Si el número no es natural deberá mostrarse un mensaje de error.
   - Si el número es natural deberá devolverse un vector con sus divisores.

   Será de ayuda usar la función as.integer.

5. **(Examen 05).** Cargar los objetos del archivo *plantas.RData* y calcular, para el data.frame plantas, lo siguiente:

   a) Media de la anchura de sépalo.

   b) Media de la anchura de sépalo agrupada por especie.

   c) Media de la anchura de sépalo cuando la anchura del pétalo es superior a 1.4.

   d) Media de la anchura de sépalo agrupada por especie cuando la anchura del pétalo es superior a 1.4.

   e) Media de la longitud del sépalo cuando la anchura del sépalo es inferior a 3.5.

   f) Media de la longitud del sépalo cuando la anchura del sépalo es inferior a 3.5 y la anchura del pétalo es superior a 0.2.

g) Crear un nuevo vector conjunto de datos en los que los elementos de la especie virginica hayan sido eliminados. Calcular la diferencia entre la media de la anchura de pétalo de este conjunto de datos y el original.

Para cargar los objetos del archivo simplemente utiliza:

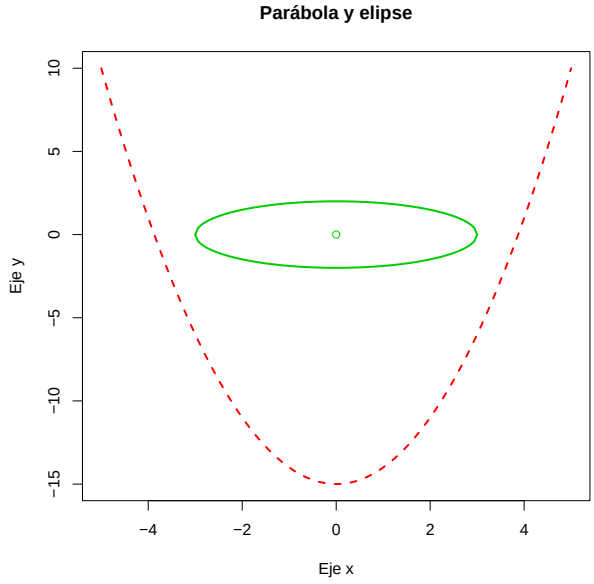
```
load("plantas.RData")
attach(plantas)
```

6. **(Examen 06).** Realiza una gráfica como la que se muestra a continuación. En ella aparece:

   a) La parábola $y = x^2 - 15$, para $x \in [-5, 5]$.

   b) Una elipse de centro $(0, 0)$ y radios $a = 3$, $b = 2$.
   La ecuación de una elipse de centro $(0, 0)$ y radios a y b viene dada por:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$



Notas:

- Asegúrate de que los límites de los ejes de tu gráfica coinciden con los de la gráfica que se muestra.
- El grosor de línea elegido para las funciones es 2. Fíjate además en los colores y en el tipo de línea utilizado para cada una de las funciones que aparecen en la gráfica.

- Añade el título y el nombre de los ejes.
- Marca el centro de la elipse con un punto.

7. **(Examen 06).** El archivo *grupo.RData* contiene el data.frame *grupo*, con información referente a las edades y sexo de 47 personas. Carga el archivo y calcula lo siguiente:

   a) Edad media de todo el grupo. (guarda el resultado en *media.grupo*)

   b) Edad media según sexo. (guarda el resultado en *media.sexo*)

   Para cargar los objetos del archivo simplemente utiliza (cuidado con el path):

   ```
   load("grupo.RData")
   attach(grupo)
   ```

8. **(Examen 06).** En el fichero *matrices.txt* están guardadas dos matrices A y B, que podrían ser la solución de dos juegos de sudoku.

   a) Importa los datos del fichero y guarda las matrices en dos objetos de R.

   b) Calcula el producto matricial de las dos matrices y el producto componente a componente.

   c) Utiliza la función *apply* para comprobar que en una matriz que sea solución de un sudoku como los del fichero todas las filas y columnas suman lo mismo.

   Nota: Si trabajas con objetos data.frame puede que sea de utilidad la función *as.matrix*.

9. **(Examen 06).** Escribe una función *expendedora* en R que simule el comportamiento de una máquina expendedora de bebidas. La máquina debe ofrecer al usuario la posibilidad de elegir entre:

   - Agua (1 euro)
   - Cola (1,50 euros)
   - Zumo natural (2.5 euros)

   a) El usuario debe seleccionar una bebida. El programa le pedirá que introduzca una cantidad de dinero por teclado.

   b) Si la cantidad de dinero introducida no es suficiente para la bebida seleccionada se le indicará cuánto falta.

   c) Si la cantidad de dinero introducida coincide con el precio de la bebida seleccionada se le indicará que la operación realizada es correcta.

   d) Si la cantidad de dinero introducida es mayor que el precio de la bebida seleccionada se le indicará la cantidad de dinero a devolver.

10. **(Examen 06).** La función *strsplit* separa los elementos de una variable character en subcadenas, según el elemento que elijamos para hacer la separación. Escribe una función en R llamada *separa* que, utilizando *strsplit*, devuelva el número de palabras de una frase. Se muestra a continuación el resultado de la ejecución de dicha función.

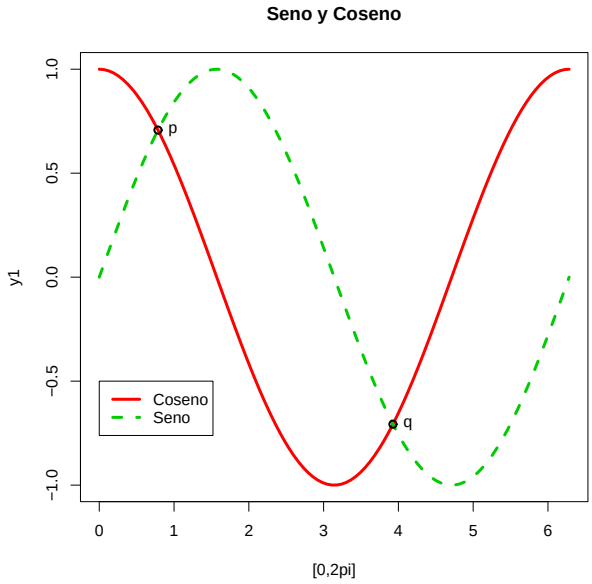
```
> separa("La madre de Juan trabaja en una farmacia")
El numero de palabras de la frase introducida es: 8
```

Nota: Fíjate en que el objeto que devuelve la función *strsplit* es una lista. Si tenemos una lista de nombre *Lst* , la forma de hacer referencia al objeto i-ésimo de la lista es mediante `Lst[[i]]` .

Puede que sea de utilidad la función *length*.

11. **(Examen 06).** Reproduce con R la gráfica que se muestra a continuación:

- Fíjate en que el rango de valores de los ejes.
- El grosor de línea elegido es 2. Fíjate además en los colores y en el tipo de línea utilizado para cada una de las funciones.
- Añade el título, nombre de los ejes y leyenda como se hace en el ejemplo.
- Marca los puntos de corte p y q.



12. **(Examen 06).** En el fichero *alumnos0506.xls*, en la hoja *datos*, se guarda información referente a los alumnos de la asignatura de Estadística de este curso.

a) Importa los datos de y guárdalos en un data.frame *alumnos*.

b) ¿Cuál es la altura media de todos los alumnos? Guarda el resultado en *media.grupo*.

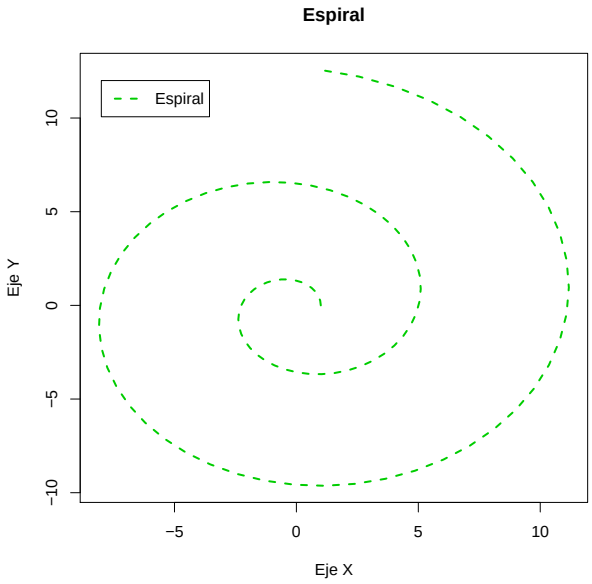
c) ¿Cuál es la altura media por sexo? Guarda el resultado en *media.sexo*.

d) ¿Cuánto aficionados del Barcelona (BAR) hay? Guarda el resultado en *barca*.

13. **(Examen 06).** Escribe una función en R, llamada *cinco*, que pida al usuario 5 números por teclado. Si alguno de los números introducidos es mayor o igual que 100, el programa devolverá un mensaje avisando de esta situación. Si no, devuelve la suma de los números introducidos que sean menores que 10.

14. **(Examen 06).** Realiza una gráfica como la que se muestra a continuación. En ella aparece una espiral cuyas ecuaciones en coordenadas polares son:

$$x = \cos(t) - t\,\mathrm{sen}(t)$$
$$y = \sin(t) + t\cos(t)$$



Notas:

- Para la espiral que aparece en la figura se ha utilizado $t \in [0, 4\pi]$.
- El grosor de línea elegido para la función es 2. Fíjate además en los colores y en el tipo de línea utilizado.
- Añade el título y el nombre de los ejes.
- Añade una leyenda como se muestra en la figura.

15. **(Examen 06).** En el fichero *gastos.xls* se guarda información referente a los gastos en material realizados por 10 empresas distintas. En la hoja *consumo* aparece el número de unidades de bolígrafos, paquetes de folios, agendas y sobres solicitados (cada fila hace referencia a una empresa). En la hoja *precio* aparece el precio por unidad.

    a) Importa los datos de consumo y guárdalos en un data.frame *consumo*.

    b) Importa los datos de precio y guárdalos en un data.frame *precio*.

    c) Utiliza la función *as.matrix* para convertir los data.frame anteriores en matrices. Llámalas *consumo.matriz* y *precio.matriz*

    d) ¿Cuál es el gasto en material de cada empresa?

    e) ¿Cuánto dinero han gastado en total las 10 empresas?

16. **(Examen 06).** Escribe una función *par.impar* que realice la suma de todos los números pares y también de todos los número impares comprendidos entre 1 y 200. La salida se muestra a continuación:

```
> par.impar()
La suma de los pares es: 10100
La suma de los impares es: 10000
```

17. **(Examen 06).** El archivo *grupo.RData* contiene el data.frame *grupo*, con información referente a las edades y sexo de 47 personas. Carga el archivo y calcula lo siguiente:

    a) Edad media de todo el grupo. (guarda el resultado en *media.grupo*)

    b) Edad media según sexo. (guarda el resultado en *media.sexo*)

    Para cargar los objetos del archivo simplemente utiliza (cuidado con el path):

```
load("grupo.RData")
attach(grupo)
```

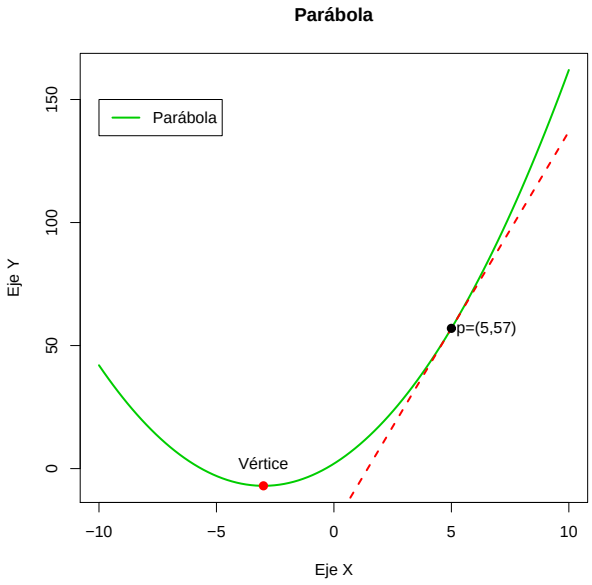
18. **(Examen 06).** Realiza una gráfica como la que se muestra a continuación. En ella aparece:

    a) La parábola $y = x^2 + 6x + 2$, para $x \in [-10, 10]$.

    b) La recta tangente a la parábola por el punto $p = (5, 57)$, cuya ecuación viene dada por $y = 16x - 23$.

    Notas:

    - Añade el vértice de la parábola como se muestra en la figura así como el punto $p = (5, 57)$.



- El grosor de línea elegido para las funciones es 2. Fíjate además en los colores y en el tipo de línea utilizado.
- Añade el título y el nombre de los ejes.
- Añade una leyenda como se muestra en la figura y el texto correspondiente al vértice y al punto de tangencia.

19. **(Examen 06).** El archivo *alumnos0506.RData* contiene el data.frame *alumnos*, donde se guarda información referente a los alumnos de la asignatura de Estadística.

    a) ¿Cuál es la altura media de todos los alumnos? Guarda el resultado en *media.grupo*.

    b) ¿Cuál es la altura media por sexo? Guarda el resultado en *media.sexo*.

    c) ¿Cuánto aficionados del Deportivo (DEP) hay? Guarda el resultado en *depor*.

    d) ¿Cuánto pesan en total todos los aficionados del Celta (CEL)? Guarda el resultado en *peso.celta*

    Para cargar los objetos del archivo simplemente utiliza (cuidado con el path):

    ```
    load("alumnos0506.RData")
    attach(alumnos)
    ```

20. **(Examen 06).** Escribe una función *suma.producto* que pida al usuario que introduzca cinco números por teclado. Además, el programa deberá ofrecer al usuario

la posibilidad de elegir entre sumar los números o multiplicarlos. Si se elige suma, se devolverá un mensaje con el resultado de la suma de los cinco números y si se elige producto, se devolverá un mensaje con el resultado del producto de los cinco números. (Puede que sean de ayuda las funciones *sum* y *prod*)

21. **(Examen 06).** Crea en R el vector $x = (1, 2, 3, 4, 5, 6)$ y define a partir de él las siguientes matrices:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \quad D = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

Capítulo 5

Estadística descriptiva

5.1. Introducción

Este capítulo está dedicado a la estadística descriptiva. En primer lugar, vamos a obtener tablas de frecuencia para variables cualitativas y cuantitativas (discretas o contínuas). A continuación veremos como resumir la información de una variable mediante el uso de medidas estadísticas, que pueden ser de centralización, de dispersión y de forma. Por último, veremos como resumir la información mediante el uso de gráficos.

5.2. Importando datos

Empezamos realizando un análisis descriptivo de datos. Lo primero que vamos a hacer es crear un script en R que lleve el nombre *Descriptiva.R*, en el que vamos a ir guardando todo el trabajo que vamos haciendo. Para ello, vamos al menú principal de R y marcamos:

$$\text{Archivo} \Longrightarrow \text{Nuevo Script}$$

y le damos un nombre en el menú principal del script:

$$\text{Archivo} \Longrightarrow \text{Guardar Como: Descriptiva.R}$$

A continuación vamos a importar un fichero de datos. Para ello, debemos escribir:

```
> datos.alumnos=read.csv2("c:/temp/alumnos0607.csv",dec=",")
```

Alternativamente, podemos importar los datos en fichero txt

```
> datos.alumnos=read.table("c:/temp/alumnos0607.txt",dec=",",header=TRUE)
> datos.alumnos
```

Para comprobar cuál es la estructura del fichero de datos una vez que lo hemos importado podemos escribir:

```
> class(datos.alumnos)
```

y obtenemos así un data.frame. Para saber qué variables contiene podemos escribir:

```
> names(datos.alumnos)
```

Vemos que tenemos 7 variables, que son: *Altura, Peso, Hermanos, Equipo, Zapato, Sexo e IMC*. Para conocer la dimensión de los datos escribimos:

```
> dim.datos<-dim(datos.alumnos)
```

Vemos que tenemos 44 valores. Definimos las siguientes variables:

```
> n.ind = dim.datos[1]
> n.var = dim.datos[2]
```

Veamos ahora diversas maneras de hacer estadística descriptiva con un conjunto de datos: (1) mediante tablas de frecuencias, (2) mediante gráficos, (3) mediante el uso de medidas de centralización, dispersión y forma.

5.3. Tablas de frecuencia y gráficos para variables cualitativas

Empezamos con las variables cualitativas. Vemos en primer lugar como obtener las frecuencias absolutas de la variable Equipo. Para ello vamos a utilizar las variables del conjunto de datos utilizando el comando *attach*:

```
> attach(datos.alumnos)
> Altura
> Peso
> Hermanos
```

A continuación, para obtener las frecuencias absolutas utilizamos la función *table*, como sigue:

```
> fabs<-table(Equipo)
> fabs
  Equipo
  BAR CEL DEP NIN RMA VAL
    3   4  13  15   8   1
> class(fabs)
```

Las frecuencias relativas las podemos obtener simplemente dividiendo la variable *fabs* por el número de individuos de la muestra *n.ind*:

```
> frel<-fabs/n.ind
  Equipo
         BAR         CEL         DEP         NIN         RMA         VAL
  0.06818182 0.09090909 0.29545455 0.34090909 0.18181818 0.02272727
```

Obtenemos así las frecuencias relativas de cada uno de los equipos de la muestra. Una vez que tenemos estas frecuencias podemos empezar a hacer resumenes gráficos para variables cualitativas. Por ejemplo, para hacer un diagrama de barras, podemos escribir:

```
> barplot(fabs,ylab="Frecuencias absolutas",main="Diagrama de barras
  de Equipo")
```

y aparece el diagrama de barras de la variable *Equipo*. El gráfico aparece en una nueva ventana con su propio menú. En el menú principal del gráfico podemos seleccionar:

$$\text{Archivo} \Longrightarrow \text{Guardar Como:}$$

que permite guardar el gráfico en diversos formatos. Las opciones del gráfico son múltiples y las podemos comprobar utilizando la ayuda:

```
> help("barplot")
```

Ver que también podemos hacer un diagrama de barras de la variable *Equipo* utilizando las frecuencias relativas:

```
> barplot(frel,ylab="Frecuencias relativas",main="Diagrama de barras
  de Equipo")
```

Para hacer un polígono de frecuencias debemos utilizar el comando *plot* como sigue:

```
> plot(fabs,type="l",main=c("Poligono de frecuencias absolutas de
  Equipos"),ylab="Frecuencias absolutas")
```

Para las frecuencias relativas tenemos:

```
> plot(frel,type="l",main=c("Poligono de frecuencias relativas de
  Equipos"),ylab="Frecuencias relativas")
```

Una alternativa es el gráfico de puntos, que es como un polígono de frecuencias salvo que no se conectan las frecuencias con líneas. Se hace de la siguiente manera:

```
> dotchart(frel,labels=c("BAR","CEL","DEP","NIN","RMA","VAL"),
  main="Grafico de puntos de Equipos")
```

Vemos como aparece el gráfico de manera horizontal con los puntos pero no aparecen las líneas. El gráfico de tarta o diagrama de sectores se hace utilizando la función *pie*:

```
> pie(fabs,col=rainbow(6),main=c("Grafico de tarta de Hermanos"))
```

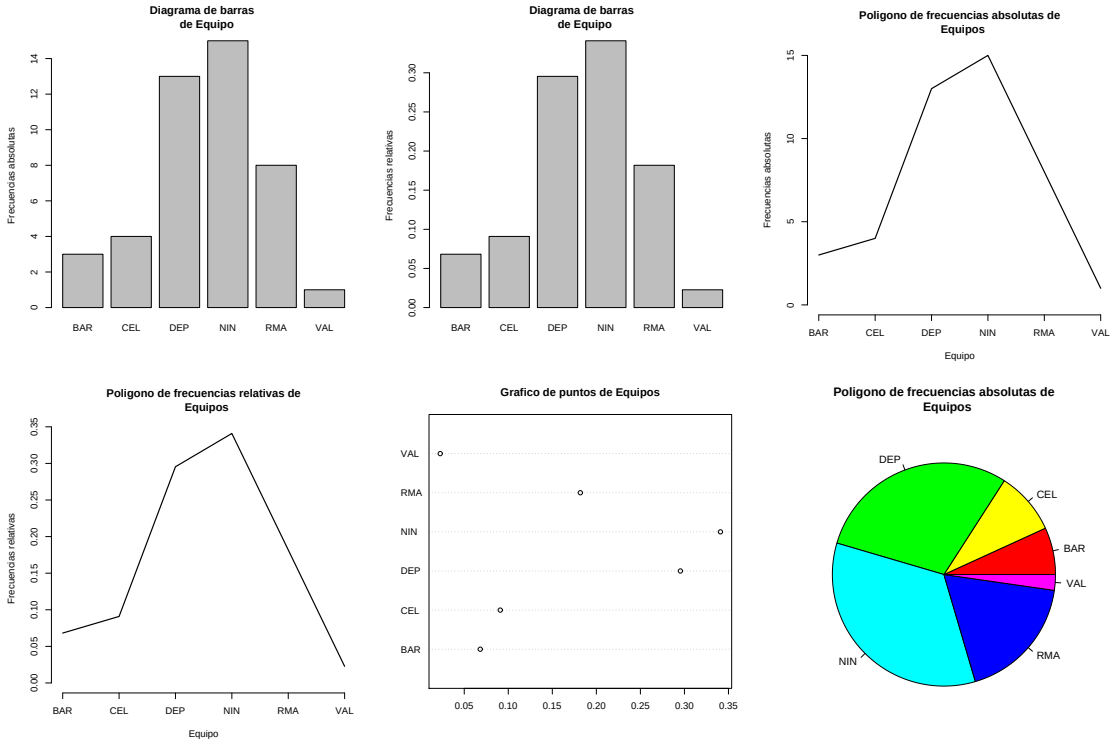


Figura 5.1: Ejemplos de gráficos para variables cualitativas.

5.4. Tablas de frecuencia y gráficos para variables cuantitativas

5.4.1. Variables discretas

Empezamos con las variables cuantitativas discretas. En primer lugar, obtenemos las frecuencias absolutas y relativas de la variable *Hermanos*, utilizando la función *table* como sigue:

```
> fabs<-table(Hermanos)
> fabs
  Hermanos
  1  2 3 4 6
  8 24 9 2 1
```

Las frecuencias relativas las podemos obtener simplemente dividiendo la variable *fabs* por el número de individuos de la muestra *n.ind*:

```
> frel<-fabs/n.ind
```

```
> frel
  Hermanos
          1          2          3          4          6
  0.18181818 0.54545455 0.20454545 0.04545455 0.02272727
```

Además ahora podemos obtener las frecuencias absoluta y relativa acumuladas. Para ello, podemos hacer lo siguiente:

```
> fabsacum<-as.table(cumsum(fabs))
> fabsacum
  1  2  3  4  6
  8 32 41 43 44
> frelacum<-as.table(cumsum(frel))
> frelacum
         1         2         3         4         6
  0.1818182 0.7272727 0.9318182 0.9772727 1.0000000
```

Con ellas podemos hacer resúmenes gráficos de todas las frecuencias, que se realizan de manera similar al caso de variables cualitativas. Por ejemplo, para hacer un diagrama de barras, podemos escribir:

```
> barplot(fabs,ylab="Frecuencias absolutas",main="Diagrama de barras de
  Hermanos")
> barplot(fabsacum,ylab="Frecuencias absolutas acumuladas",main="Diagrama
  de barras de Hermanos")
```

Para un polígono de frecuencias:

```
> plot(fabs,type="l",main="Poligono de frecuencias absolutas de
  Hermanos",ylab="Frecuencias absolutas")
> plot(fabsacum,type="l",main="Poligono de frecuencias absolutas
  acumuladas de Hermanos",ylab="Frecuencias absolutas acumuladas")
```

Para un gráfico de puntos:

```
> dotchart(fabs,main="Graficos de puntos de Hermanos")
> dotchart(fabsacum,main="Graficos de puntos de Hermanos")
```

Para el gráfico de tarta o diagrama de sectores se hace utilizando la función *pie*:

```
> pie(fabs,col=rainbow(6),main=c("Grafico de tarta de Hermanos"))
```

Hay que tener en cuenta que el diagrama de sectores no se puede hacer para frecuencias absolutas.

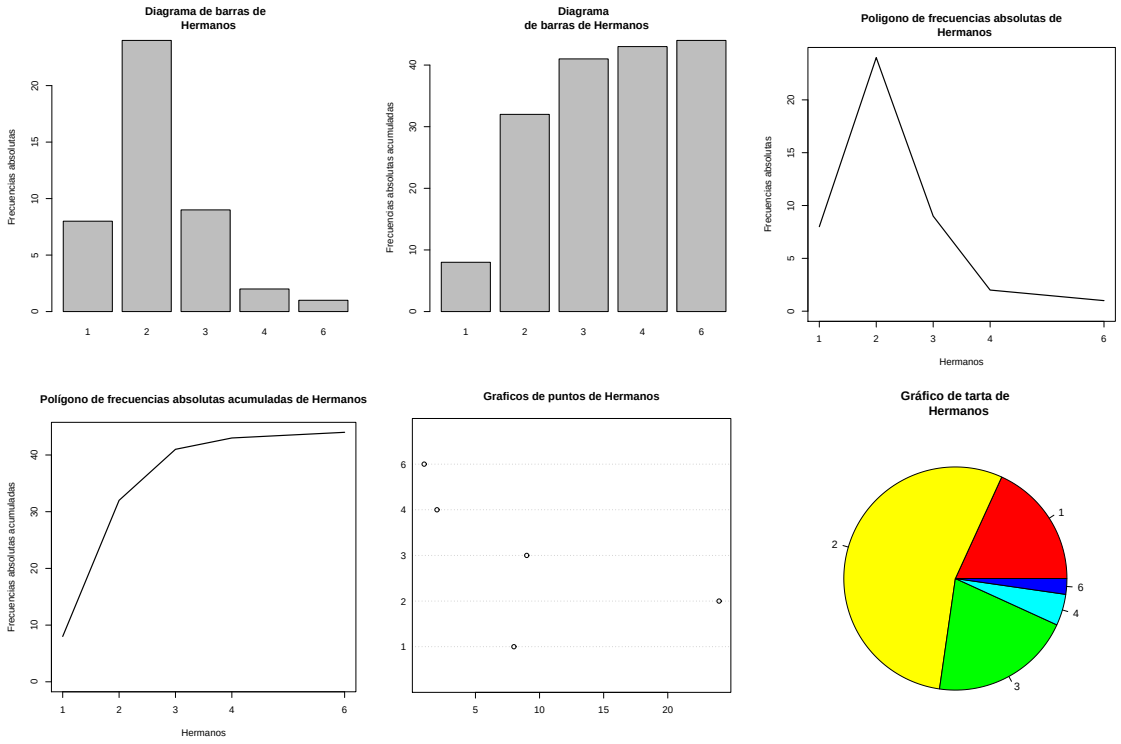


Figura 5.2: Ejemplos de gráficos para variables cuantitativas discretas.

5.4.2. Variables continuas

Para variables cuantitativas continuas las cosas se complican algo debido a que tenemos que agrupar los valores de las variables. En primer lugar, obtenemos las frecuencias absolutas y relativas de la variable *IMC*, utilizando la función table como sigue:

```
> fabs<-table(cut(IMC,breaks=7))
> frel<-table(cut(IMC,breaks=7))/n.ind
```

Mientras que las frecuencias acumuladas son:

```
> fabsacum<-cumsum(fabs)
> frelacum<-cumsum(frel)
```

Para hacer un histograma simplemente debemos de escribir la siguiente orden:

```
> hist(IMC)
```

El problema es que R por defecto selecciona el número de clases siguiendo un método interno llamado método de Sturges. Para poder utilizar el número de clases que a nosotros nos interesa más, tenemos que crear un vector con los puntos de corte de las clases. Esto se puede hacer fácilmente mediante:

```
> n.clases=7
> puntos=min(IMC)+(0:n.clases)*(max(IMC)-min(IMC))/n.clases
> hist(IMC, breaks=puntos)
```

Por ejemplo, para 9 clases sería:

```
> n.clases=9
> puntos=min(IMC)+(0:n.clases)*(max(IMC)-min(IMC))/n.clases
> hist(IMC, breaks=puntos)
```

Por último, un diagrama de tallo y hojas se puede obtener mediante:

```
> stem(IMC)
```

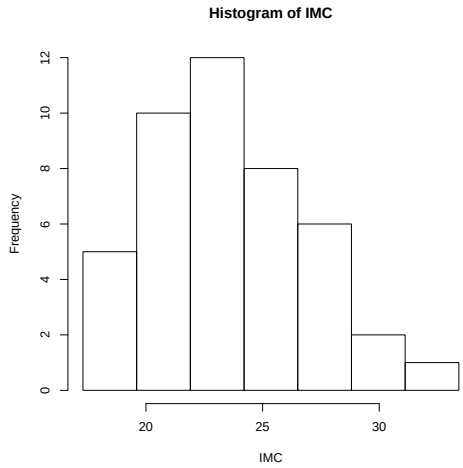


Figura 5.3: Ejemplo de gráfico para variables cuantitativas continuas: Histograma.

5.5. Medidas de centralización, dispersión y forma

A continuación vemos algunas de las medidas vistas en teoría. Empezamos con la media:

```
> mean(Altura)
  [1] 173.7045
```

Podemos utilizar la misma función para calcular la media truncada como sigue:

```
> mean(Altura,trim=0.10)
  [1] 173.8056
> mean(sort(Altura)[5:40])
  [1] 173.8056
```

Por lo tanto, la media truncada elimina el 10 % de las observaciones más grandes y más pequeñas. Para la variable Hermanos sería:

```
> mean(Hermanos)
  [1] 2.204545
> mean(Hermanos,trim=0.10)
  [1] 2.111111
```

Otra medida de centralización es la mediana, que se puede obtener como sigue:

```
> median(Altura)
  [1] 174
> median(Hermanos)
  [1] 2
```

Los cuantiles son igual de fáciles de obtener utilizando la función *quantile*. Debemos especificar que cuantiles queremos. Por ejemplo, para los cuantiles de orden 0.10, 0.25, 0.50, 0.70, 0.90 y 0.99, escribimos:

```
> quantile(Altura,probs=c(.1,.25,.5,.7,.9,.99))
     10%    25%    50%    70%    90%    99%
  165.00 170.00 174.00 177.00 182.00 186.71
```

Respecto a las medidas de dispersión, tenemos la varianza, la desviación típica, la meda, el rango intercuartílico y el rango:

```
> var(Altura)
  [1] 43.46882
> sd(Altura)
  [1] 6.593089
> sqrt(var(Altura))
  [1] 6.593089
> mad(Altura)
  [1] 5.9304
> IQR(Altura)
  [1] 8
> range(Altura)
  [1] 158 188
> diff(range(Altura))
  [1] 30
```

Veamos las mismas medidas para una variable asimétrica hacia la derecha:

```
> var(Hermanos)
  [1] 0.9106765
> sd(Hermanos)
  [1] 0.9542937
```

```
> mad(Hermanos)
  [1] 0
> IQR(Hermanos)
  [1] 1
> diff(range(Hermanos))
  [1] 5
```

Para obtener medidas de forma, necesitamos la librería *moments*:

```
> library(moments)
> skewness(Altura)
  [1] -0.1437376
> kurtosis(Altura)
  [1] 2.842092
```

Las medidas de forma para las variables asimétricas son:

```
> skewness(Hermanos)
  [1] 1.531962
> kurtosis(Hermanos)
  [1] 7.13868
```

Vemos como el coeficiente de asimetría es positivo, ya que la variable es asimétrica hacia la derecha, mientras que el coeficiente de curtosis es mayor que en el caso de la variable *Altura*. Con la función *moment* podemos obtener todo tipo de momentos, sean centrados o no. Por ejemplo, para obtener los momentos terceros centrados y no centrados podemos escribir:

```
> moment(Altura,order=3,central=TRUE)
  [1] -39.79797
> moment(Altura,order=3,central=FALSE)
  [1] 5263332
```

5.5.1. Gráficos basados en los cuartiles: el gráfico de caja

Por último, vemos el boxplot, que se puede obtener utilizando la función:

```
> boxplot(Peso)
> boxplot(Peso,notch = TRUE,col = "blue",main="Grafico de caja de
  Peso")
> boxplot(Hermanos)
> boxplot(Hermanos,notch = TRUE,col = "blue",main="Grafico de caja de
  Peso")
```

La primera línea es el mínimo entre el valor mínimo y Q1-1.5*IQR, la segunda es Q1, la tercera es la mediana y la cuarta es el máximo entre el valor máximo y Q3+1.5*IQR. Si hubiese "outliers" aparecerían como puntos antes o después de la primera y última línea.

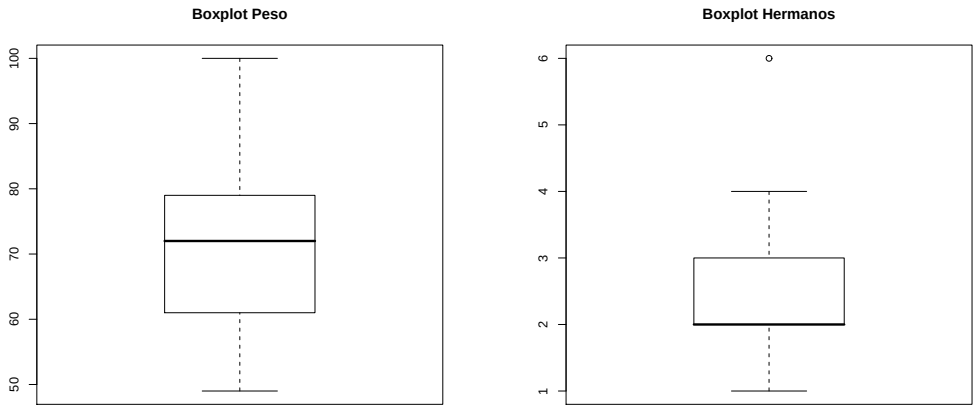


Figura 5.4: Diagramas de caja.

5.6. Gráficos para multiples variables

De la misma manera que hemos visto gráficos para variables unidimensionales, podemos hacer gráficos para múltiples variables. Cuando tenemos más de dos variables para analizar podemos utilizar gráficos alternativos que muestren la relación que existe entre estas variables. Evidentemente tenemos el problema de que podemos ver objetos en a lo sumo tres dimensiones.

En primer lugar vemos un gráfico de dispersión de la variable *Peso* frente a *Altura*:

```
> par(mfrow=c(1,2))
> plot(Peso,Altura,xlab="Peso",ylab="Altura",main="Grafico de dispersion")
```

Como se puede ver, parece existir la existencia de una relación lineal. También disponemos del histograma bidimensional. Para poder utilizar este diagrama en R necesitamos importar la librería *gplots*:

```
> library(gplots) # Alternativa es la libreria gregmisc.
> hist2d(Peso,Altura,nbins=c(7,7),xlab="Peso",ylab="Altura",
  main="Histograma bidimensional")
```

Comprobamos como la escala de colores de más oscuro a menos oscuro marca la presencia de un menor o un mayor número de datos. Si lo que queremos es cambiar los colores de este gráfico para que el significado sea exactamente el opuesto podemos hacer lo siguiente:

```
> par(mfrow=c(1,2))
> plot(Peso,Altura,xlab="Peso",ylab="Altura",main="Grafico de dispersion")
```

```
> hist2d(Peso,Altura,nbins=c(7,7),xlab="Peso",ylab="Altura",
  main="Histograma bidimensional",col=c("white",heat.colors(12)[12:1]))
```

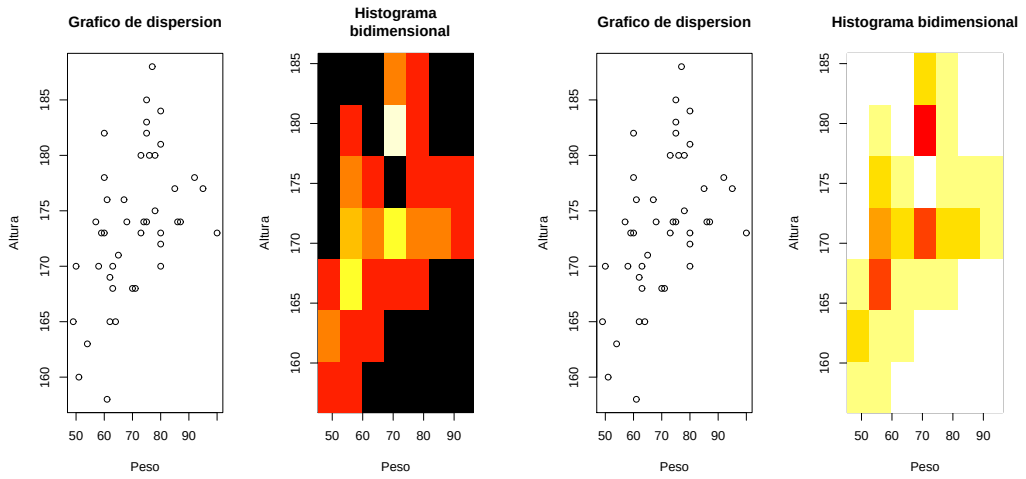


Figura 5.5: Gráficos para multiples variables (función hist2d).

Una manera de verlo en tres dimensiones es la siguiente. En primer lugar escribimos:

```
> h2d=hist2d(Peso,Altura,show=FALSE,nbins=c(7,7))
> h2d
```

Comprobamos como con esta función también podemos obtener una tabla de frecuencia bidimensional de dos variables. Además, podemos obtener las frecuencias relativas dividiendo por el número de datos.

```
> h2d$counts/44
```

A continuación, para poder ver un gráfico que si bien no es exactamente un histograma en tres dimensiones, es lo más parecido que podemos obtener:

```
> persp(h2d$x,h2d$y,h2d$counts,xlab="Peso",ylab="Altura",zlab="frecuencias
  absolutas",ticktype="detailed",theta=30,phi=30,col="lightblue",shade=0.75)
```

Aquí podemos ir cambiando el ángulo de vista de las variables (*theta*) y la inclinación del gráfico (*phi*). Por ejemplo,

```
> persp(h2d$x,h2d$y,h2d$counts,xlab="Peso",ylab="Altura",zlab="frecuencias
  absolutas",ticktype="detailed",theta=-30,phi=30,col="lightblue",shade=0.75)
```

También podemos incluir colores para distinguir grupos de datos. Por ejemplo, podemos hacer un gráfico de dispersión entre dos variables añadiendo colores dependiendo de si el alumno es hombre o mujer:

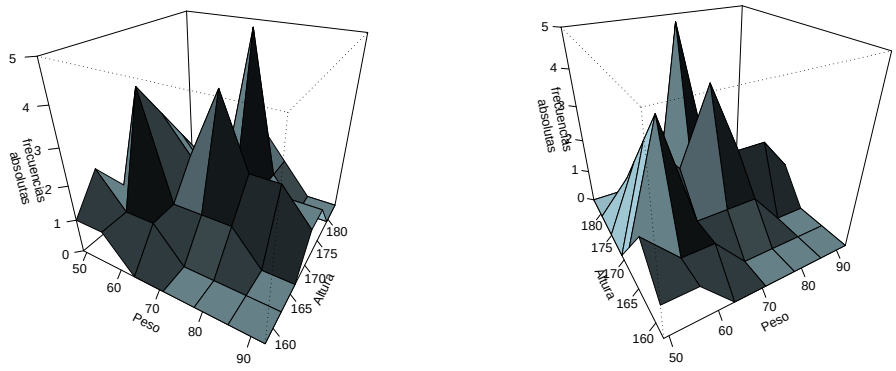


Figura 5.6: Gráficos para multiples variables (función persp).

```
> par(mfrow=c(1,1))
> plot(Peso,Altura,xlab="Peso",ylab="Altura",main="Grafico de
  dispersion")
> colores=c("red", "blue")[Sexo]
> plot(Peso,Altura,xlab="Peso",ylab="Altura",main="Grafico de
  dispersion",pch=21,bg=colores)
> plot(Zapato,Altura,xlab="Peso",ylab="Altura",main="Grafico de
  dispersion",pch=21,bg=colores)
> plot(Zapato,Peso,xlab="Peso",ylab="Altura",main="Grafico de
  dispersion",pch=21,bg=colores)
```

La explicación de las opciones *pch* y *bg* se obtinen utilizando:

```
> ?par
> ?points
```

De aquí obtenemos que `pch=21` se especifica para que dibuje círculos en cada uno de los puntos y que `bg` se utiliza para indicar los colores de los círculos. También podemos hacer "zooms" como sigue:

```
> plot(Peso,Altura,xlab="Peso",ylab="Altura",pch=21,bg=colores,
  xlim=c(55,75),ylim=c(165,180))
```

¿Qué ocurre cuando tenemos más de una variable? Podemos hacer un gráfico de dispersión para varias variables:

```
> pairs(datos.alumnos[,c(1,2,5)])
> pairs(datos.alumnos[,c(1,2,5)],pch=21,bg=colores)
```

Capítulo 6

Variables aleatorias

6.1. Introducción

Hasta ahora hemos supuesto que disponíamos de un conjunto de datos que nos venía dado, pero hemos reflexionado muy poco acerca de cómo se obtienen estos datos. Se denomina "experimento" al proceso por el que obtenemos observaciones. Notar que podemos distinguir entre dos tipos diferentes de experimentos: deterministas y aleatorios.

1. Los experimentos deterministas son aquellos tales que siempre que se repitan bajo condiciones análogas, se obtiene el mismo resultado. Es decir, son totalmente predecibles.

2. Los experimentos aleatorios son aquellos tales que siempre que se repitan bajo condiciones análogas, se obtienen resultados diferentes, pero que se conocen previamente. Es decir, dentro de los posibles resultados, el resultado del experimento es impredecible.

Los experimentos que nos interesan son los que producen resultados impredecibles, es decir, los experimentos aleatorios. Por ejemplo, ¿cúal es el tiempo de espera hasta acceder a una página web? Dicho tiempo depende de multitud de factores que ocasionan que tengamos un amplio rango de valores posibles, pero que antes de pulsar "enter" en el ordenador es imposible de determinar con exactitud. Una variable aleatoria se define entonces como el resultado de realizar un experimento aleatorio. En el ejemplo, podemos definir la variable aleatoria $X =$*"Tiempo de espera hasta acceder a una página web"*. Este experimento se puede repetir tantas veces como se quiera. Si realizamos este experimento 100 veces y tomamos los tiempos de espera, obtenemos una muestra de valores de la variable aleatoria de tamaño muestral 100. La población correspondería a todas las posibles veces que podemos intentar acceder a la página web que, en principio, son infinitas.

Podemos dividir las variables aleatorias en discretas y continuas:

1. Variables aleatorias continuas son las que toman valores de un conjunto de valores discretos. Por ejemplo, el número de zapato, el número de hermanos, el resultado

de lanzar un dado o el número de aciertos en una quiniela son variables aleatorias discretas.

2. Variables aleatorias discretas son las que toman valores de la recta real. Por ejemplo, el tiempo de espera hasta que te conectas a la página web o el índice de masa corporal son variables aleatorias continuas.

Pero gracias a nuestra experiencia sabemos que los valores de ciertos experimentos se repiten unos más que otros. Por ejemplo, sabemos que es más frecuente tener 7 aciertos en la quiniela que 14, o que el tiempo de espera en conectarse la página web principal de la Universidad de Santiago de Compostela no suele ser mayor que 1 segundo. Esto ya lo sabemos ya que hemos visto como obtener frecuencias absolutas y relativas. El concepto de probabilidad procede de estas frecuencias. Gracias a la probabilidad, podemos relacionar los conceptos de población y muestra e inferir si los resultados sobre una muestra pueden ser extrapolados al conjunto de la población. Vamos a ver esto con varios ejemplos.

6.2. Distribución de Bernoulli

Vamos a comprobar los resultados del lanzamiento de una moneda. Sólo tenemos dos posibles resultados para cada lanzamiento: cara o cruz. El ejercicio es el siguiente. Vamos a escribir una función que simule los lanzamientos de una moneda. Para ello, utilizamos la siguiente función, donde los valores "C" corresponden a caras y los valores "X" corresponden a cruces:

```
> moneda=function(n){mon=c(); u=runif(n); mon[u<0.5]="C";
  mon[u>=0.5]="X";mon}
```

Esta función proporciona n resultados del lanzamiento de una moneda al aire. Probamos los resultados para 5 lanzamientos.

```
> n=5
> moneda(n)
  [1] "X" "C" "X" "C" "C"
> moneda(n)
  [1] "C" "X" "C" "C" "C"
> moneda(n)
  [1] "X" "X" "X" "C" "C"
```

Vemos qué ocurre cuando el número de lanzamientos va creciendo:

```
> num.lan=10
> lanzamientos=moneda(num.lan)
> frec.abs=table(lanzamientos) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/num.lan # Corresponde a las frecuencias relativas
```

```
> par(mfrow=c(4,1))
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras de Lanzamientos")
> num.lan=100
> lanzamientos=moneda(num.lan)
> frec.abs=table(lanzamientos) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/num.lan # Corresponde a las frecuencias relativas
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras de Lanzamientos")
> num.lan=1000
> lanzamientos=moneda(num.lan)
> frec.abs=table(lanzamientos) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/num.lan # Corresponde a las frecuencias relativas
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras de Lanzamientos")
> num.lan=10000
> lanzamientos=moneda(num.lan)
> frec.abs=table(lanzamientos) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/num.lan # Corresponde a las frecuencias relativas
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras de Lanzamientos")
```

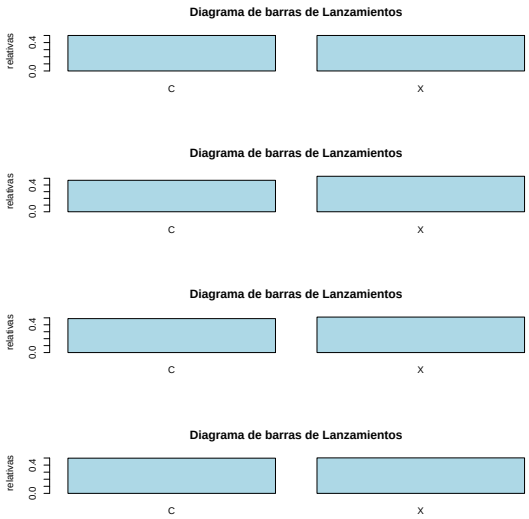


Figura 6.1: Frecuencias relativas para el lanzamiento de monedas con $n = 10, 100, 1000, 10000$.

Parece ser que, cuanto mayor es el número de intentos, más se acerca la frecuencia relativa del número de caras a 0.5. Este valor corresponde a lo que llamamos probabilidad de

obtener cara. Claro está, la probabilidad de obtener cruz es 0.5. Por lo tanto, podemos decir que la variable aleatoria X =*"Resultado de lanzar una moneda al aire"* toma el valor "C" (cara) con probabilidad 0.5 y el valor "X" (cruz) con probabilidad 0.5. Se define entonces la función de probabilidad de la variable X como $P(X=0) = 0{,}5$ y $P(X=1) = 0{,}5$. Se dice que las variables aleatorias que tienen función de probabilidad $P(X=1) = p$ y $P(X=0) = 1-p$, para un cierto valor de p entre 0 y 1, tienen distribución de Bernoulli de parámetro p.

6.3. Distribución binomial

Como hemos visto, una variable aleatoria Bernoulli toma dos posibles valores con probabilidades p y $1-p$, respectivamente. A continuación, consideramos la variable aleatoria binomial que se obtiene a partir de la variable aleatoria Bernoulli. Para ello, consideramos el siguiente experimento. Suponemos que tenemos una cierta conexión mediante un módem. Dicho módem recibe paquetes de 100 bits de manera que existe una probabilidad de 0,01 de sufrir un error en la lectura de cada uno de los bits. Parece interesante saber cuál es la probabilidad de recibir 0, 1, 2, 3, *etc...* bits erróneos en cada uno de los paquetes recibidos. Desde un punto de vista estadístico, el problema se puede plantear como sigue. Para cada paquete de 100 bits podemos asociar una variable aleatoria de 1's y 0's tales que dicha variable vale 0 si el bit recibido es correcto y vale 1 si el bit recibido es erróneo. Por lo tanto, para cada bit tenemos asociada una variable Bernoulli de probabilidad $p = 0{,}01$. El número de bits erróneos totales es la suma de las variables Bernoulli que valgan 0. Dicha variable aleatoria recibe el nombre de binomial de parámetros $n = 100$ y $p = 0{,}01$. Una variable aleatoria binomial de parámetros n y p tiene función de probabilidad:

$$P(X=x) = \binom{n}{x} p^x (1-p)^x, \ x = 0, 1, \ldots, n$$

El paquete R tiene una función que proporciona el valor de estas probabilidades para cualquier valor de los parámetros n y p. Vemos cuáles son estas probabilidades:

```
> n=100
> p=0.01
> dbinom(0,n,p)
  [1] 0.3660323 # Es la probabilidad de tener 0 bits erroneos
> dbinom(1,n,p)
  [1] 0.3660323 # Es la probabilidad de tener 1 bits erroneos
> dbinom(2,n,p)
  [1] 0.1848648 # Es la probabilidad de tener 2 bits erroneos
> dbinom(3,n,p)
  [1] 0.06099917 # Es la probabilidad de tener 3 bits erroneos
> dbinom(4,n,p)
  [1] 0.01494171 # Es la probabilidad de tener 4 bits erroneos
```

Podemos escribir todas las probabilidades obtenidas a partir de la función *dbinom*:

```
> for (i in 0:100) cat(i,"\t",dbinom(i,n,p),"\n")
```

Podemos hacer un gráfico de la función de probabilidad mediante:

```
> par(mfrow=c(1,2))
> plot(dbinom(0:100,n,p),xlab="Bits erroneos",type="h") # Funcion
  de probabilidad de una B(100,0.01)
```

Además, también podemos calcular la función de distribución que se define como:

$$P(X \leq x) = P(X = 0) + P(X = 1) + \cdots + P(X = x).$$

Para ello, utilizamos la función *pbinom* como sigue:

```
> for (i in 0:100) cat(i,"\t",pbinom(i,n,p),"\n")
```

Podemos hacer un gráfico de la función de distribución mediante:

```
> plot(pbinom(0:100,n,p),xlab="Bits erroneos",type="S") # Funcion
  de distribucion de una B(100,0.01)
```

El paquete R tiene una función que genera valores aleatorios de una variable con distribución binomial de estos parámetros. Por ejemplo, vamos a suponer que recibimos 100 paquetes a través de nuestro módem y contabilizamos el número de bits erróneos en cada uno de los paquetes. Para hacer esto:

```
> num.paq=100
> bits.err=rbinom(num.paq,n,p)
> frec.abs=table(bits.err) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/num.paq  # Corresponde a las frecuencias relativas
> par(mfrow=c(4,1))
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras del numero de bits erroneos")
```

Vemos qué ocurre cuando el número de intentos va creciendo:

```
> num.paq=1000
> bits.err=rbinom(num.paq,n,p)
> frec.abs=table(bits.err) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/num.paq # Corresponde a las frecuencias relativas
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras del numero de bits erroneos")
> num.paq=10000
> bits.err=rbinom(num.paq,n,p)
> frec.abs=table(bits.err) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/num.paq # Corresponde a las frecuencias relativas
```

```
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras del numero de bits erroneos")
> num.paq=100000
> bits.err=rbinom(num.paq,n,p)
> frec.abs=table(bits.err) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/num.paq # Corresponde a las frecuencias relativas
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras del numero de bits erroneos")
```

Podemos comparar las frecuencias relativas con los valores teóricos obtenidos anteriormente:

```
> dbinom(0:10,n,p)
```

Vemos como las frecuencias relativas tienden a las probabilidades obtenidas a partir de la función de probabilidad. Por último en este apartado, mencionar que dos distribuciones discretas relacionadas con la distribución de Bernoulli son la distribución geométrica y la distribución binomial negativa:

1. La distribución geométrica se utiliza para representar tiempos de espera y se define como el número de experimentos Bernoulli necesarios hasta el primer 1. Por ejemplo, para el caso de los bits defectuosos, el número de bits observados hasta que el primer error ocurre es una variable geométrica. La función de probabilidad está dada por:
$$P(X = x) = (1-p)^x p,\ x = 0, 1, \ldots$$
Funciones en R para la variable geométrica son *dgeom* (función de probabilidad), *pgeom* (función de distribución) y *rgeom* (valores aleatorios).

2. La distribución binomial negativa también se utiliza para representar tiempos de espera y se define como el número de experimentos Bernoulli necesarios hasta un cierto número de 1's. Por ejemplo, para el caso de los bits defectuosos, el número de bits observados hasta que el tercer error ocurre es una variable binomial negativa. La función de probabilidad está dada por:
$$P(X = x) = \binom{n+x-1}{x} p^n (1-p)^x,\ x = 0, 1, \ldots, n$$
donde n es el número total de experimentos realizados. Funciones en R para la variable binomial negativa son *dnbinom* (función de probabilidad), *pnbinom* (función de distribución) y *rnbinom* (valores aleatorios).

6.4. Distribución de Poisson

Supongamos que tenemos una página web y un patrocinador que nos paga en función del número de entradas en dicha página. Me interesa saber cual es el número de entradas

probables en la página para saber cuánto puedo ganar con este negocio. Mientras que, a mi patrocinador, también le interesa saber cuánto dinero se puede gastar. Si suponemos que existe "estabilidad", en el sentido de que el número de sucesos por unidad de tiempo (λ) permanece constante, entonces, la variable aleatoria número de entradas en mi página web es una variable aleatoria de Poisson y tiene función de probabilidad dada por:

$$P(X = x) = \frac{e^{-\lambda}\lambda^x}{x!},\ x = 0, 1, \ldots$$

donde λ es el número de entradas esperadas en el periodo de observación. Por ejemplo, si contabilizamos el número de entradas en una hora, λ es el número esperado de entradas en la página web y coincide con la esperanza de la variable aleatoria X.

Por ejemplo, supongamos que $\lambda = 20$, por lo que el número esperado de entradas en nuestra página web en una hora es 20. El paquete R tiene una función que proporciona el valor de las probabilidades de cualquier número para cualquier valor de λ. Vemos cúales son estas probabilidades:

```
> lam=20
> dpois(0,lam)
  [1] 2.061154e-09 # Probabilidad de que nadie entre
> dpois(1,lam)
  [1] 4.122307e-08 # Probabilidad de una entrada
> dpois(2,lam)
  [1] 4.122307e-07 # Probabilidad de dos entradas
> dpois(3,lam)
  [1] 2.748205e-06 # Probabilidad de tres entradas
```

Como se puede ver casi no hay probabilidad de números tan bajos. Podemos escribir varias probabilidades obtenidas a partir de la función *dpois*:

```
> for (i in 0:50) cat(i,"\t",dpois(i,lam),"\n")
```

Podemos hacer un gráfico de la función de probabilidad mediante:

```
> par(mfrow=c(1,2))
> plot(dpois(0:50,lam),xlab="Entradas en la pagina web",type="h")
  # Funcion de probabilidad de una Poisson(20)
```

Además, también podemos calcular la función de distribución que se define por:

$$P(X \leq x) = P(X = 0) + P(X = 1) + \cdots + P(X = x).$$

Para ello, utilizamos la función *ppois* como sigue:

```
> for (i in 0:50) cat(i,"\t",ppois(i,lam),"\n")
```

Podemos hacer un gráfico de la función de distribución mediante:

```
> plot(ppois(0:50,lam),xlab="Entradas en la pagina web",type="S")
  # Funcion de distribucion de una Poisson(20)
```

¿Qué pasa si contabilizamos el número de páginas web durante varias horas?

```
> lam=20
> horas=100
> num.ent=rpois(horas,lam)
> frec.abs=table(num.ent) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/horas # Corresponde a las frecuencias relativas
> par(mfrow=c(4,1))
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras del numero de entradas")
```

Vemos qué ocurre cuando el número de intentos va creciendo:

```
> horas=1000
> num.ent=rpois(horas,lam)
> frec.abs=table(num.ent) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/horas # Corresponde a las frecuencias relativas
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras del numero de entradas")
> horas=10000
> num.ent=rpois(horas,lam)
> frec.abs=table(num.ent) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/horas # Corresponde a las frecuencias relativas
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras del numero de entradas")
> horas=100000
> num.ent=rpois(horas,lam)
> frec.abs=table(num.ent) # Corresponde a las frecuencias absolutas
> frec.rel=frec.abs/horas # Corresponde a las frecuencias relativas
> barplot(frec.rel,col="lightblue",ylab="Frecuencias
  relativas",main="Diagrama de barras del numero de entradas")
```

Podemos comparar las frecuencias relativas con los valores teóricos obtenidos anteriormente:

```
> dpois(0:50,lam)
```

6.5. Distribución exponencial

La primera variable continua que vamos a analizar es la variable exponencial. Supongamos ahora que en lugar de contabilizar el número de entradas en la página web, lo que hacemos es medir el tiempo entre diferentes entradas. Si el proceso es de Poisson, la

variable aleatoria que mide el tiempo entre diferentes entradas tiene una distribución exponencial de parámetro λ. La función de densidad de una variable exponencial está dada por:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{si } x \geq 0, \\ 0 & \text{si } x < 0. \end{cases}$$

¿De dónde procede esta función de densidad? Veamos qué es lo que ocurre cuando medimos los tiempos entre entradas a la página web. En R podemos generar datos de una distribución exponencial utilizando la función *rexp*:

```
> lam=20
> num.ent=100
> tie.ent=rexp(num.ent,lam)
> par(mfrow=c(5,1))
> n.clases=10
> puntos=min(tie.ent)+(0:n.clases)*(max(tie.ent)-min(tie.ent))/n.clases
> hist(tie.ent,breaks=puntos,col="green",freq=FALSE)
> num.ent=1000
> tie.ent=rexp(num.ent,lam)
> n.clases=32
> puntos=min(tie.ent)+(0:n.clases)*(max(tie.ent)-min(tie.ent))/n.clases
> hist(tie.ent,breaks=puntos,col="green",freq=FALSE)
> num.ent=10000
> tie.ent=rexp(num.ent,lam)
> n.clases=317
> puntos=min(tie.ent)+(0:n.clases)*(max(tie.ent)-min(tie.ent))/n.clases
> hist(tie.ent,breaks=puntos,col="green",freq=FALSE)
> num.ent=100000
> tie.ent=rexp(num.ent,lam)
> n.clases=1000
> puntos=min(tie.ent)+(0:n.clases)*(max(tie.ent)-min(tie.ent))/n.clases
> hist(tie.ent,breaks=puntos,col="green",freq=FALSE)
```

Lo que comprobamos es que el límite del histograma es una curva que es la función de densidad. Podemos obtener probabilidades mediante la función de probabilidad que se define por:

$$P(X \leq a) = F(a) = \int_{-\infty}^{a} f(x)\, dx = \int_{0}^{a} \lambda e^{-\lambda x} dx = 1 - e^{-\lambda a}.$$

De otra manera, el área bajo la curva nos proporciona la probabilidad de un determinado suceso. Ahora podemos añadir al gráfico los valores de la función de densidad de la variable exponencial.

```
> plot(seq(0,.5,.001),dexp(seq(0,.5,.001),lam),type="l",main="Funcion
  de densidad de la exponencial")
```

También podemos ver la función de distribución:

```
> plot(seq(0,.5,.001),pexp(seq(0,.5,.001),lam),,type="l",main="Funcion
  de distribucion de la exponencial")
```

6.6. Distribución gamma

La distribución gamma es una generalización de la distribución exponencial. En lugar de esperar al primer evento, esperamos hasta el evento número r. De esta manera, tenemos la distribución Γ de parámetros λ y r, que tiene función de densidad dada por:

$$f(x) = \begin{cases} \frac{\lambda^r}{\Gamma(r)} e^{-\lambda x} x^{r-1} & \text{si } x \geq 0, \\ 0 & \text{si } x < 0, \end{cases}$$

donde $\Gamma(r)$ es la función gamma dada por $\Gamma(r) = \int_0^\infty x^{r-1} e^{-x} dx$. Generamos valores de dicha distribución y comparamos sus histogramas de frecuencias y de frecuencias acumuladas con la función de densidad y de distribución. Comprobamos como los dos histogramas tienden a las funciones de densidad y de distribución, respectivamente.

```
> r=10
> veces=100
> tiempos=rgamma(veces,lam,r)
> n.clases=10
> puntos=min(tiempos)+(0:n.clases)*(max(tiempos)-min(tiempos))/n.clases
> par(mfrow=c(2,2))
> histo=hist(tiempos,breaks=puntos,freq=FALSE)
> plot(seq(0,5,.001),dgamma(seq(0,5,.001),lam,r),,type="l",main="Funcion
  de densidad de la gamma") # Funcion de densidad de una Gamma(20,10)
> barplot(cumsum(histo$counts/veces),space=0)
> plot(seq(0,5,.001),pgamma(seq(0,5,.001),lam,r),,type="l",
  main="Funcion de distribucion de la gamma") # Funcion de distribucion
  de una Gamma(20,10)
> veces=10000
> tiempos=rgamma(veces,lam,r)
> n.clases=100
> puntos=min(tiempos)+(0:n.clases)*(max(tiempos)-min(tiempos))/n.clases
> par(mfrow=c(2,2))
> histo=hist(tiempos,breaks=puntos,freq=FALSE)
> plot(seq(0,5,.001),dgamma(seq(0,5,.001),lam,r),,type="l",main="Funcion
  de densidad de la gamma") # Funcion de densidad de una Gamma(20,10)
> barplot(cumsum(histo$counts/veces),space=0)
> plot(seq(0,5,.001),pgamma(seq(0,5,.001),lam,r),,type="l",main="Funcion
  de distribucion de la gamma") # Funcion de distribucion de
  una Gamma(20,10)
```

6.7. Distribución normal

La distribución normal es la más usada de todas las distribuciones de probabilidad. Gran cantidad de experimentos se ajustan a distribuciones de esta familia, por ejemplo el peso y la talla de una persona, los errores de medición... Además, si una variable es el resultado de la suma de variables aleatorias independientes, es bastante posible que pueda ser aproximada por una distribución normal. La distribución normal depende de dos parámetros, la esperanza μ y la desviación típica σ, y tiene función de densidad dada por:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad -\infty < x < \infty.$$

Generamos valores de dicha distribución para los valores $\mu = 0$ y $\sigma = 1$, y comparamos sus histogramas de frecuencias y de frecuencias acumuladas con la función de densidad y de distribución. Comprobamos como los dos histogramas tienden a las funciones de densidad y de distribución, respectivamente.

```
> mu=0
> sig=1
> num.datos=100
> datos=rnorm(num.datos,mu,sig)
> n.clases=10
> puntos=min(datos)+(0:n.clases)*(max(datos)-min(datos))/n.clases
> par(mfrow=c(2,2))
> histo=hist(datos,breaks=puntos,freq=FALSE)
> plot(seq(-5,5,.001),dnorm(seq(-5,5,.001),mu,sig),,type="l",
  main="Funcion de densidad de la normal") # Funcion de densidad
  de una N(0,1)
> barplot(cumsum(histo$counts/veces),space=0)
> plot(seq(-5,5,.001),pnorm(seq(-5,5,.001),mu,sig),type="l",
  main="Funcion de distribucion de la normal") # Funcion de
  distribucion de una N(0,1)
> num.datos=10000
> datos=rnorm(num.datos,mu,sig)
> n.clases=100
> puntos=min(datos)+(0:n.clases)*(max(datos)-min(datos))/n.clases
> par(mfrow=c(2,2))
> histo=hist(datos,breaks=puntos,freq=FALSE)
> plot(seq(-5,5,.001),dnorm(seq(-5,5,.001),mu,sig),type="l",
  main="Funcion de densidad de la normal") # Funcion de densidad
  de una N(0,1)
> barplot(cumsum(histo$counts/veces),space=0)
> plot(seq(-5,5,.001),pnorm(seq(-5,5,.001),mu,sig),type="l",
  main="Funcion de distribucion de la normal") # Funcion de
  distribucion de una N(0,1)
```

Otras distribuciones univariantes cotinuas de interés son la Erlang, la Weibull o la beta.

6.8. Variables aleatorias bidimensionales

A veces nos puede interesar obtener probabilidades conjuntas de varias variables aleatorias, es decir, buscamos:

$$F(x,y) = P(X \leq x, Y \leq y) = \int_{-\infty}^{x}\int_{-\infty}^{y} f(x,y)\,dydx.$$

El caso más simple es cuando las variables son independientes. En este caso,

$$P(X \leq x, Y \leq y) = P(X \leq x)\,P(Y \leq y)$$

es decir,

$$F(x,y) = F(x)\,F(y),$$

por lo que

$$F(x,y) = P(X \leq x, Y \leq y) = \int_{-\infty}^{x}\int_{-\infty}^{y} f(x,y)\,dydx = \int_{-\infty}^{x} f(x)\,dx \int_{-\infty}^{y} f(y)\,dy.$$

Veamos algún ejemplo. Tomamos como densidades la gamma de parámetros $\lambda = 20$ y $r = 10$ y la normal de parámetros $\mu = 0$ y $\sigma = 1$, podemos representar la función de densidad conjunta mediante: mediante:

```
> lam=20
> r=10
> x=seq(0,5,.05)
> mu=0
> sig=1
> y=seq(-5,5,.05)
> persp(x, y, outer(x,y,function(x,y){dgamma(x,lam,r)*dnorm(y,mu,sig)}),
  theta = 30,phi = 30, expand = 0.5, col = "lightblue",
  main="X: Gamma(20,10) Y: Normal(0,1)")
```

Si tomamos como densidades la exponencial de parámetros $\lambda = 20$ y la normal de parámetros $\mu = 0$ y $\sigma = 1$, podemos representar la función de densidad conjunta mediante:

```
> lam=20
> x=seq(0,0.5,.05)
> mu=0
> sig=1
```

```
> y=seq(-5,5,.05)
> persp(x, y, outer(x,y,function(x,y){dexp(x,lam)*dnorm(y,mu,sig)}),
  theta = 30,phi = 30, expand = 0.5, col = "lightblue",
  main="X: Exp(20) Y:Normal(0,1)")
```

Si tomamos como densidades dos normales, una de parámetros $\mu = 0$ y $\sigma = 1$ y otra de parámetros $\mu = 3$ y $\sigma = 2$. La función de densidad conjunta de ambas la podemos contemplar mediante:

```
> mu1=0
> sig1=1
> x=seq(-5,5,.05)
> mu2=3
> sig2=2
> y=seq(-5,10,.05)
> persp(x, y,outer(x,y,
  function(x,y){dnorm(x,mu1,sig1)*dnorm(y,mu2,sig2)}),
  theta =30,phi = 30, expand = 0.5, col = "lightblue",
  main="X: Normal(0,1) Y:Normal(3,2)")
```

6.9. Teorema Central del Límite

El Teorema Central del Límite (TCL) es uno de los resultados más importantes en estadística. Bajo ciertas condiciones nos permite usar la distribución normal para estudiar otras distribuciones más generales. Este teorema tiene muchas versiones, por lo que vemos una simple:

Teorema Central del Límite: Si $X_1, \ldots, X_n$ son variables aleatorias independientes e idénticamente distribuidas con media μ y desviación típica σ, entonces $\sum_{i=1}^{n} X_i$ tiene asintóticamente una distribución normal de media $n\mu$ y desviación típica $\sqrt{n}\sigma$. Por las propiedades de la normal, tendremos que:

$$\frac{\sum_{i=1}^{n} X_i - n\mu}{\sqrt{n}\sigma} \overset{n\to\infty}{\longrightarrow} N(0,1).$$

Este teorema nos dice que cuando los resultados de un experimento son debidos a un conjunto muy grande de causas independientes que actúan sumando sus efectos, entonces esos resultados siguen una distribución aproximadamente normal. Dicho de otra forma, aunque cada uno de los efectos sea raro o difícil de estudiar, si lo que queremos estudiar es la suma de los mismos sabemos que, bajo ciertas condiciones, esta se comportaría de modo normal. El Teorema Central del Límite sirve para dar una explicación al hecho constatado de que muchas distribuciones de variables observadas en la naturaleza o en experimentos físicos sean aproximadamente normales. Por ejemplo, las medidas del cuerpo humano, como la estatura, peso, longitud de los huesos, siguen aproximadamente una distribución normal dentro de una misma raza y sexo. En el caso de la estatura de un individuo afectan muchos factores, cada uno ejerciendo un efecto pequeño, de manera

que el efecto total sea la suma de esos efectos individuales. Por esa razón, un histograma de estaturas de individuos de un mismo sexo es aproximadamente normal.

Veamos el Teorema Central del Límite en el caso de la binomial. Como sabemos la binomial de parámetros n y p es la suma de n Bernoullis de parámetro p. Tomamos, $n = 40$ y $p = 0{,}4$. El valor esperado y la desviación típica de la binomial de parámetros $n = 40$ y $p = 0{,}4$ son $np = 16$ y $\sqrt{np\,(1-p)} = \sqrt{40 \times 0{,}4 \times 0{,}6} = 3{,}0983$, respectivamente. Comparamos ambas distribuciones.

```
> x=seq(0,40,by=1)
> plot(x,dbinom(x,40,0.4),pch=2,col=2)
> lines(x,dnorm(x,16,3.0983))
```

Como se puede comprobar la aproximación de la binomial por la normal es realmente buena. Podemos comparar numéricamente estos valores:

```
> for (i in 0:40) cat(i,"\t",dbinom(i,40,0.4),"\t",
  dnorm(i,16,3.0983),"\n")
```

En la práctica se utiliza la aproximación cuando $n \geq 30$, $np \geq 5$ y $n\,(1-p) \geq 5$. En nuestro caso se verifican todas las restricciones ya que $n = 40$, $np = 16$ y $n\,(1-p) = 24$. Para comprobar que estas restricciones son importantes, repetimos el experimento tomando valores que no las verifican. Por ejemplo, tomamos $n = 10$ y $p = 0{,}1$. Entonces, $np = 1$ y $n\,(1-p) = 9$. Vemos el efecto:

```
> x=seq(0,10,by=1)
> plot(x,dbinom(x,10,0.1),pch=2,col=2)
> lines(x,dnorm(x,1,0.9486))
```

Como se puede comprobar la aproximación de la binomial por la normal no es tan buena como en el caso anterior. Podemos comparar numéricamente estos valores:

```
> for (i in 0:10) cat(i,"\t",dbinom(i,10,0.1),"\t",
  dnorm(i,1,0.9486),"\n")
```

Veamos que pasa con la distribución de Poisson. Volvemos al ejemplo, en el que teníamos que el número de entradas en una página web por hora tenía una distribución Poisson de parámetro $\lambda = 20$. Esta variable puede considerarse como la suma de 60 variables que midan el número de entradas en una página web por minuto, que también son variables Poisson de parámetro $20/60 = 0{,}3333$. El TCL aplicado a variables con distribución Poisson dice que cuanto mayor sea el número de variables Poisson que sumemos, la variable resultante (que también es Poisson) se parecerá cada vez más a una normal. Se considera que si $\lambda > 5$, la aproximación a la normal es buena. En ese caso, se puede considerar que, aproximadamente $N\left(\lambda, \sqrt{\lambda}\right)$.

```
> x=seq(0,40,by=1)
> plot(x,dpois(x,20),pch=2,col=2)
> lines(x,dnorm(x,20,4.4721))
```

Como se puede comprobar la aproximación de la binomial por la normal es bastante buena. Podemos comparar numéricamente estos valores:

```
> for (i in 0:40) cat(i,"\t",dpois(i,20),"\t",dnorm(i,20,4.4721),"\n")
```

Capítulo 7

Transformaciones de variables aleatorias

7.1. Introducción

Tanto en variables discretas como en continuas a veces vamos a tener que realizar transformaciones por múltiples causas. En este breve capítulo vamos a ver las dos transformaciones más usuales: la tipificación y la transformación de Box-Cox.

7.2. Tipificación

La transformación más habitual de los datos es lo que llamamos la tipificación. La tipificación de una variable consiste en restar la media y dividir por la desviación típica. Es decir, si tengo la variable formada por los datos $x_1, \ldots, x_n$, entonces podemos definir los datos $z_1, \ldots, z_n$ como sigue:

$$z_i = \frac{x_i - \overline{x}}{s_x}.$$

De esta manera, $\overline{z} = 0$ y $s_z = 1$. Hacer la tipificación en R es muy simple. Importamos los datos:

```
> datos.alumnos=read.csv2("d:/temp/alumnos0607.csv",dec=",")
> datos.alumnos=read.table("d:/temp/alumnos0607.txt",dec=",",header=TRUE)
> attach(datos.alumnos)
> IMC
  [1] 27.68166 19.71332 19.69267 17.30104 17.99816 18.93700
      22.46003 20.04745 26.72993 22.53086 30.32334 24.07407
     .......
> IMC.tip=(IMC-mean(IMC))/sd(IMC)
> IMC.tip
  [1] 1.22657977 -1.03913994 -1.04501334 -1.72504926 -1.52682844
     -1.25987859 -0.25813890 -0.94413501  0.95596364 -0.23799912
```

```
     .......
> mean(IMC.tip)
  [1] -3.437685e-16
> sd(IMC.tip)
  [1] 1
```

En principio el hecho de tipificar no debe afectar en la forma del histograma siempre que seleccionemos el mismo número de clases. Por ejemplo:

```
> par(mfrow=c(1,2))

> n.clases=7
> puntos=min(IMC)+(0:n.clases)*(max(IMC)-min(IMC))/n.clases
> hist(IMC, breaks=puntos)\begin{verbatim}
> n.clases=7
> puntos=min(IMC.tip)+(0:n.clases)*(max(IMC.tip)-min(IMC.tip))/n.clases
> hist(IMC.tip, breaks=puntos)
```

7.3. Transformaciones de Box-Cox

Por lo general, la teoría estadística va a ser más sencilla cuanto mayor sea la simetría de la variable, es decir, si la asimetría es lo más próxima a cero. Para obtener una variable lo más simétrica posible disponemos de las transformaciones de Box-Cox que están definidas por:

$$z_i = \begin{cases} \frac{(x_i+m)^\lambda - 1}{\lambda} & \text{si } \lambda \neq 0, \\ log\,(x_i + m) & \text{si } \lambda = 0, \end{cases}$$

donde m es tal que $x_i + m > 0$, para todo valor de x_i. La transformación se elige de esta manera para que sea continua en 0, ya que:

$$\lim_{\lambda \to 0} \frac{(x_i+m)^\lambda - 1}{\lambda} = \lim_{\lambda \to 0} \frac{(x_i+m)^\lambda \log\,(x_i+m)}{1} = \log\,(x_i+m)\,.$$

Para valores de λ mayores que uno, se corrige la asimetría a la izquierda y para valores de λ menores que uno, se corrige asimetría a la derecha. Por ejemplo, la variable *IMC* es ligeramente asimétrica a la derecha. Podemos intentar corregir esta asimetría utilizando la transformación de Box-Cox. Utilizamos tres transformaciones:

```
> n.clases=7
> puntos=min(IMC)+(0:n.clases)*(max(IMC)-min(IMC))/n.clases
> hist(IMC, breaks=puntos)
> IMC.bc1=((IMC)^(1/2)-1)/(1/2)
> IMC.bc2=log(IMC)
> IMC.bc3=((IMC)^(-1/2)-1)/(-1/2)
> IMC.bc4=((IMC)^(-1)-1)/(-1)
```

```
> par(mfrow=c(4,1))
> puntos=min(IMC.bc1)+(0:n.clases)*(max(IMC.bc1)-min(IMC.bc1))/n.clases
> hist(IMC.bc1, breaks=puntos)
> puntos=min(IMC.bc2)+(0:n.clases)*(max(IMC.bc2)-min(IMC.bc2))/n.clases
> hist(IMC.bc2, breaks=puntos)
> puntos=min(IMC.bc3)+(0:n.clases)*(max(IMC.bc3)-min(IMC.bc3))/n.clases
> hist(IMC.bc3, breaks=puntos)
> puntos=min(IMC.bc4)+(0:n.clases)*(max(IMC.bc4)-min(IMC.bc4))/n.clases
> hist(IMC.bc4, breaks=puntos)
> library(moments)
> skewness(IMC)
  [1] 0.5902573
> skewness(IMC.bc1)
  [1] 0.3973666
> skewness(IMC.bc2)
  [1] 0.2107917
> skewness(IMC.bc3)
  [1] 0.03037595
> skewness(IMC.bc4)
  [1] -0.1440558
```

Si no podemos utilizar la librería *moments*, podemos calcular la asimetría mediante la siguiente fórmula:

```
> sum((IMC-mean(IMC))^3)/(44*sd(IMC)^3)
```

Notar que puede haber pequeñas discrepancias ya que R para calcular la desviación típica divide por $n-1$ en lugar de por n, por razones que entenderemos más adelante.

Capítulo 8

Gráficos dinámicos para el análisis de datos

8.1. Introducción

Esta práctica está dedicada a la utilización de R como herramienta para la realización de representaciones gráficas de datos bidimensionales, así como para el tratamiento visual de grandes muestras: "binned plots".

También veremos representaciones gráficas de datos multidimensionales, manipulaciones gráficas de datos bidimensionales: etiquetado (localización e identificación), eliminación y linkado de datos o grupos de datos de interés mediante técnicas de cepillado (brushing).

8.2. Los datos

A lo largo de esta práctica trabajaremos con el conjunto de datos conocido como *Fisher's iris data*. Este conjunto nos da la medida en cm. de las variables longitud y anchura de sépalo y longitud y anchura de pétalo para un total de 150 flores de tres especies diferentes de iris (iris setosa, versicolor y virginica).

Figura 8.1: Iris setosa, versicolor y virginica

Para acceder a los datos en R usaremos: *data(iris)*. Tendremos así:

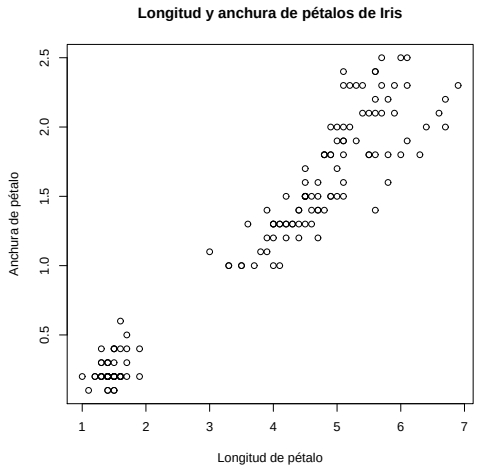
```
> data(iris)
> iris
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
...
```

El objeto *iris* es un data.frame con 150 filas y 5 variables (columnas) llamadas *Sepal.Length, Sepal.Width, Petal.Length, Petal.Width y Species*.

8.3. Representación gráfica de datos bidimensionales

Nos centraremos en las variables longitud y anchura de pétalo. Ya sabemos como representar gráficamente los datos correspondientes a estas variables. Simplemente haremos:

```
> plot(iris[,3],iris[,4],main="Longitud y anchura de pétalos de Iris",
xlab="Longitud de pétalo",ylab="Anchura de pétalo")
```



8.4. Localización e identificación de datos

A la vista de la gráfica anterior parece que existe al menos un grupo diferenciado de datos (aquellos que tienen una longitud y anchura de pétalo claramente inferior al resto). Puede ser de interés identificar este grupo de datos y ver si se corresponden con una especie de iris en particular.

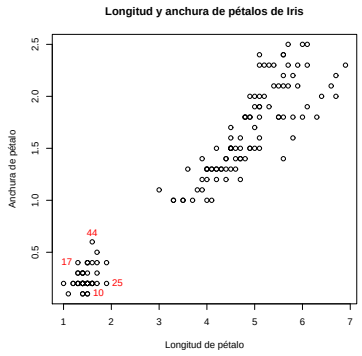
Las funciones *locator* e *identify* nos permiten localizar e identificar datos, respectivamente.

Supongamos entonces que queremos identificar los puntos en la gráfica anterior de ese grupo diferenciado para ver si pertenecen a una especie en particular. Si hacemos

```
> identify(iris[,3],iris[,4])
```

cada vez que marquemos un punto en la gráfica el programa nos devolverá el número de componente del punto dentro de los vectores `iris[,3]` e `iris[,4]`.

En el siguiente ejemplo hemos seleccionado cuatro puntos dentro de ese grupo que parece tener longitud y anchura de pétalo claramente inferior al resto. Una vez dadas los número de componente de los puntos marcados comprobamos cual es la especie a la que pertenecen dichos puntos. Todos ellos son datos de flores de la especie *setosa*.




```
> identify(iris[,3],iris[,4],col=2)
[1] 10 17 25 44
> iris[10,5]
[1] setosa
Levels: setosa versicolor virginica
> iris[17,5]
[1] setosa
Levels: setosa versicolor virginica
> iris[25,5]
[1] setosa
Levels: setosa versicolor virginica
> iris[44,5]
[1] setosa
Levels: setosa versicolor virginica
```

8.5. Reescalado y utilización de factores de visión ("zooms")

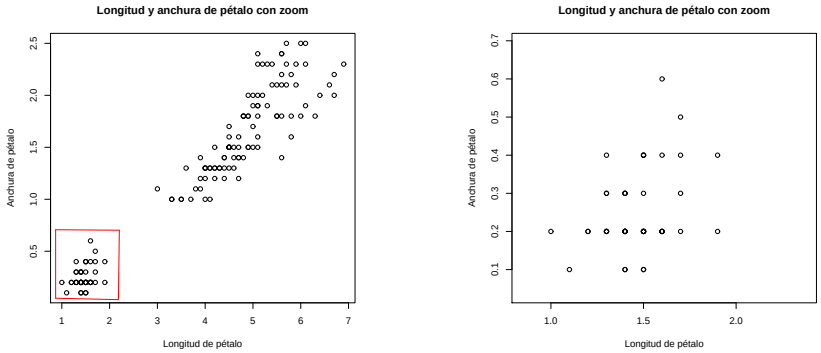
En ocasiones puede ser de utilidad ampliar una determinada zona de un gráfico para ver mejor los datos representados en dicha zona. En esta sección programaremos una función *plot.scaling* que nos permita realizar un zoom en un gráfico.

La idea es que dado un gráfico, el usuario marque la zona que desea ampliar. El programa devolverá la gráfica correspondiente al cuadrado marcado por el usuario. El código correspondiente a la función descrita es:

```
plot.scaling<-function(x,y=NULL,...){
  plot(x,y,...)
  z<-locator(5,type="l",col=2)
  lims=c(min(z$x),max(z$x),min(z$y),max(z$y))
  plot(x,y,xlim=lims[1:2],ylim=lims[3:4],...)
}
```

Se muestra a continuación el resultado de una ejecución de dicha función.

```
> plot.scaling(iris[,3],iris[,4],main="Longitud y anchura de pétalo
  con zoom",xlab="Longitud de pétalo",ylab="Anchura de pétalo")
```

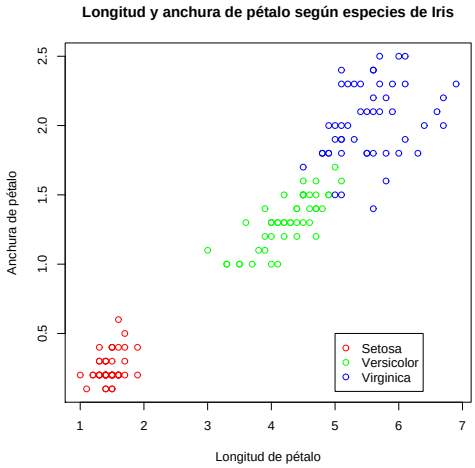


8.6. Representación gráfica de datos multidimensionales

Hasta ahora hemos visto como representar datos bidimensionales. En este caso la información relativa a los datos se puede visualizar fácilmente. Los diferentes tipos de gráficos (histogramas, boxplots, ...) nos ofrecen información relativa a los datos que estamos representando. Sin embargo, con datos multidimensionales la representación gráfica es más complicada. Veremos como tratar en R esta situación.

En primer lugar intentaremos repetir la primera gráfica pero de forma que ahora distingamos los datos correspondientes a cada especie. Para ello tendremos que indicarle al programa que represente cada especie mediante un color diferente.

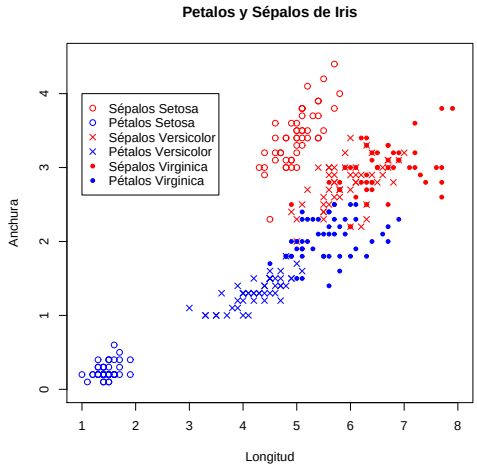
```
> iris.color<-c("red","green","blue")[iris$Species]
> plot(iris[,3],iris[,4],col=iris.color,main="Longitud y anchura
  de pétalo según especies de Iris",xlab="Longitud de pétalo",
  ylab="Anchura de pétalo")
> legend(5,0.5,c("Setosa","Versicolor","Virginica"),pch=1,
  col=c("red","green","blue"))
```



8.6.1. Las funciones matplot y matpoints

Cuando trabajamos con matrices de datos podemos utilizar las funciones *matplot* y *matpoints* de la misma forma que utilizábamos las funciones *plot* y *points* cuando teníamos vectores de datos. Por ejemplo:

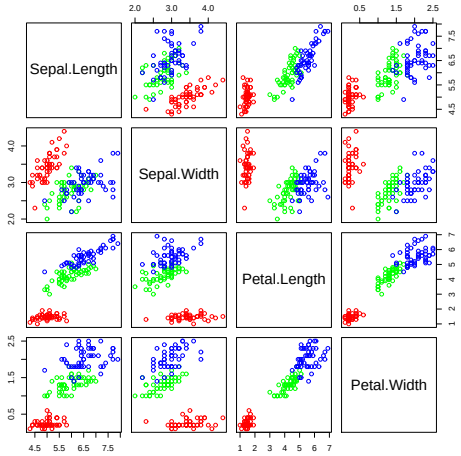
```
> iS <- iris$Species == "setosa"
> iV <- iris$Species == "versicolor"
> iVi<- iris$Species == "virginica"
> matplot(c(1,8),c(0,4.5),type= "n",xlab="Longitud",ylab ="Anchura",
  main = "Petalos y Sépalos de Iris")
> matpoints(iris[iS,c(1,3)],iris[iS,c(2,4)],pch =1, col=c(2,4))
> matpoints(iris[iV,c(1,3)],iris[iV,c(2,4)],pch =4, col=c(2,4))
> matpoints(iris[iVi,c(1,3)],iris[iVi,c(2,4)],pch=20, col=c(2,4))
> legend(1,4,c("Sépalos Setosa","Pétalos Setosa","Sépalos Versicolor",
  "Pétalos Versicolor","Sépalos Virginica","Pétalos Virginica"),
  pch =c(1,1,4,4,20,20),col=rep(c(2,4), 2))
```



8.6.2. La función pairs

Otra de las funciones utilizadas para la representación gráfica de datos multidimensionales es la función *pairs*. Se utiliza como se muestra a continuación.

```
> pairs(iris[1:4],col=iris.color)
```



8.7. Brushing

El *"brushing"* o *"cepillado"* es una técnica que nos permite rastrear e identificar datos o grupos de datos en diferentes gráficos.

Volvemos a la gráfica anterior. Se trata de seleccionar un punto o grupo de puntos en una de las gráficas y localizarlo en el resto. El código se muestra a continuación. Consta de una función *pairs.brushing* y una subrutina *submatriz* que sirve para elegir la gráfica que queremos ampliar.

```
pairs.brushing<-function(datos,color=1,color.extra=2,...){
  pairs(datos,col=color,...)
  nvar<-dim(datos)[2]
  z<-locator(1)
  sub<-submatriz(z,nvar)
  plot(datos[,sub[1]],datos[,sub[2]],col=color,...)
  z<-identify(datos[,sub[1]],datos[,sub[2]])
  newcolor<-numeric(dim(datos)[1])
  newcolor[]<-color
  newcolor[z]<-color.extra
  pairs(datos,col=newcolor,...)
}

submatriz<-function(z,nvar){
  plot.x<-as.integer(z$x*nvar)+1
  plot.y<-as.integer(nvar*(1-z$y))+1
  return(c(plot.x,plot.y))
}
```

Por ejemplo, hacemos

```
> pairs.brushing(iris[1:4],color=iris.color,color.extra=1)
```

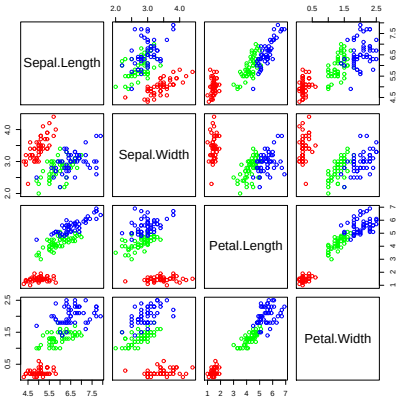


Figura 8.2: Dibujamos los datos

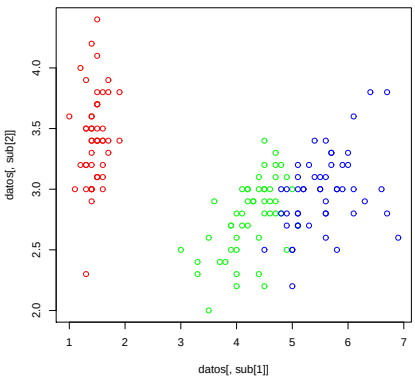


Figura 8.3: Elegimos una gráfica

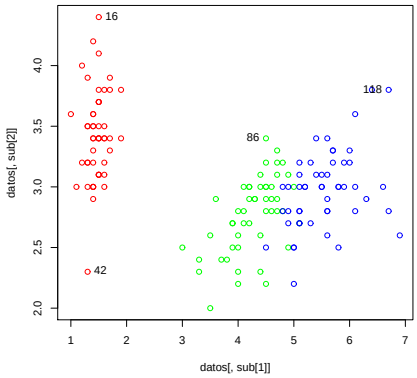


Figura 8.4: Seleccionamos puntos

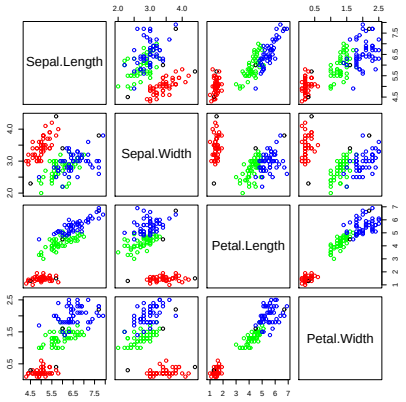


Figura 8.5: Los puntos elegidos aparecen marcados en negro

Podemos hacer lo mismo pero ahora, en lugar de seleccionar puntos sueltos, se trata de seleccionar un conjunto de puntos y localizar dichos puntos en todas las gráficas.

```
pairs.boxbrush<-function(datos,color=1,color.extra=2,...){
  pairs(datos,col=color,...)
  nvar<-dim(datos)[2]
  z<-locator(1)
  sub<-submatriz(z,nvar)
  plot(datos[,sub[1]],datos[,sub[2]],col=color,...)
  z<-locator(type="l")
  lims<-c(min(z$x),max(z$x),min(z$y),max(z$y))
```

```
  newcolor<-numeric(dim(datos)[1])
  newcolor[]<-color
  newcolor[(lims[1]<datos[,sub[1]]&datos[,sub[1]]<lims[2]&
  lims[3]<datos[,sub[2]]&datos[,sub[2]]<lims[4])]<-color.extra
  pairs(datos,col=newcolor,...)
}
```

Así, si hacemos

```
> pairs.boxbrush(iris[1:4],color=iris.color,color.extra=1)
```

obtenemos las siguientes representaciones.

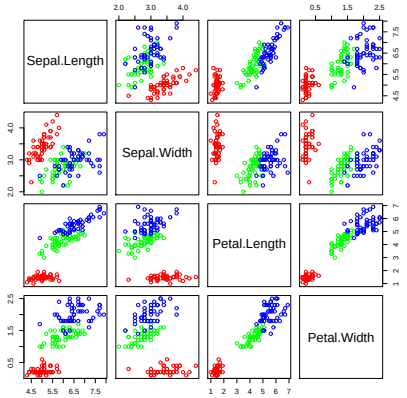


Figura 8.6: Dibujamos los datos

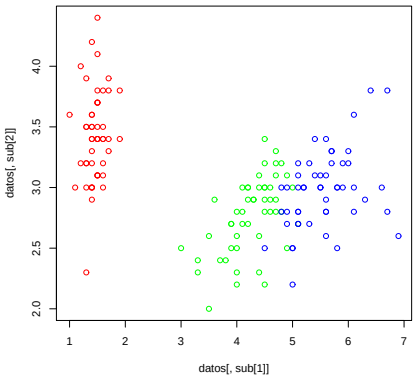


Figura 8.7: Elegimos una gráfica

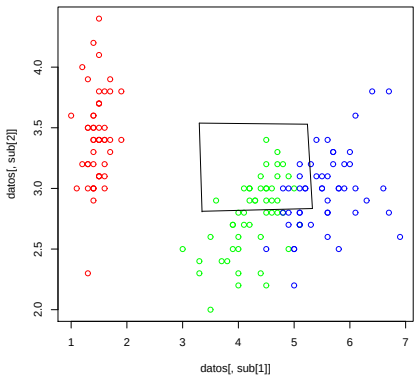


Figura 8.8: Seleccionamos un conjunto de puntos

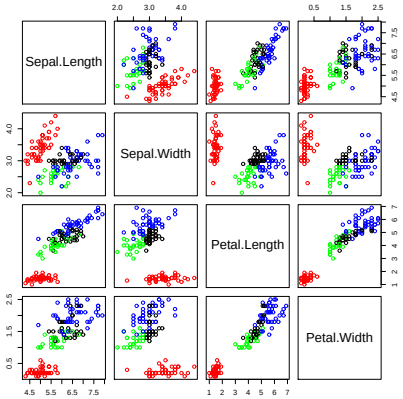


Figura 8.9: Los puntos elegidos aparecen marcados en negro

Capítulo 9

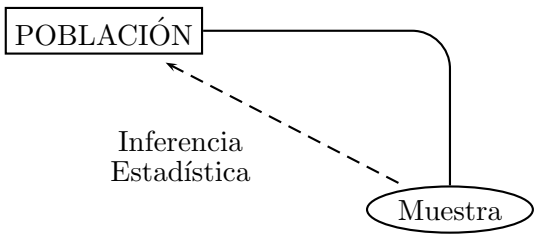
Inferencia paramétrica

9.1. Introducción

La inferencia estadística es la parte de la estadística matemática que se encarga del estudio de los métodos para la obtención del modelo de probabilidad (forma funcional y parámetros que determinan la función de distribución) que sigue una variable aleatoria de una determinada población, a través de una muestra (parte de la población) obtenida de la misma.

Los dos problemas fundamentales que estudia la inferencia estadística son el "Problema de la estimación" y el "Problema del contraste de hipótesis".

Cuando se conoce la forma funcional de la función de distribución que sigue la variable aleatoria objeto de estudio y sólo tenemos que estimar los parámetros que la determinan, estamos ante un problema de **inferencia estadística paramétrica** ; por el contrario cuando no se conoce la forma funcional de la distribución que sigue la variable aleatoria objeto de estudio, estamos ante un problema de **inferencia estadística no paramétrica**.



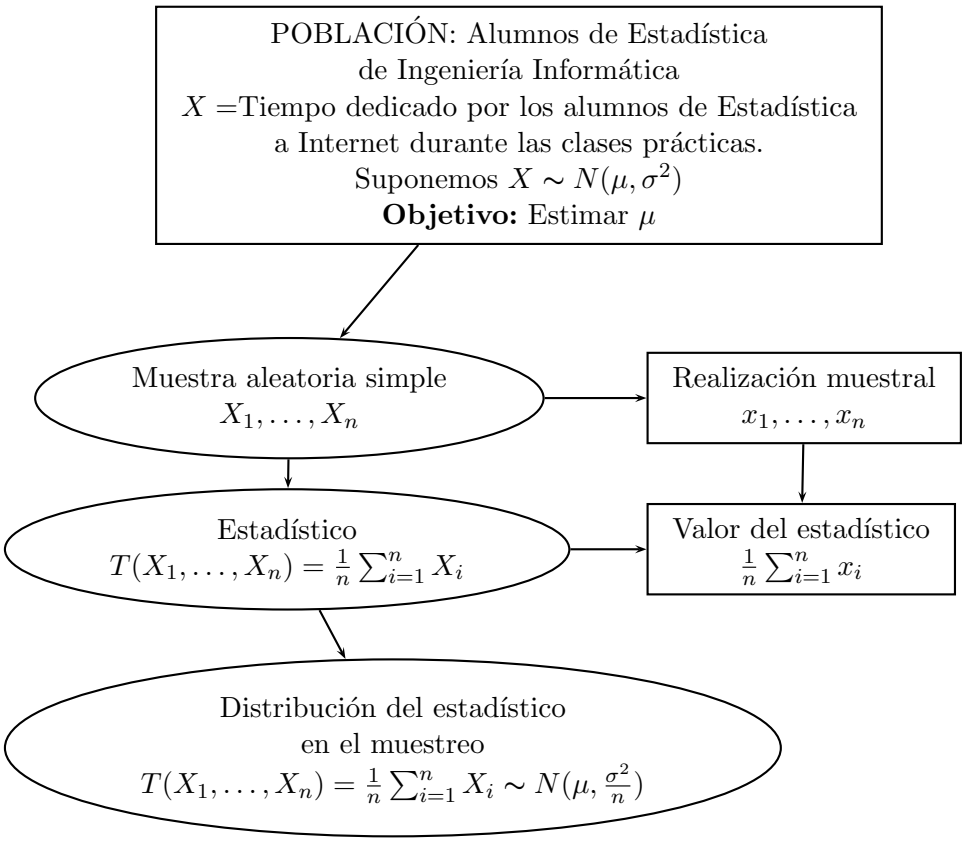
9.2. Planteamiento del problema

Objetivo: Estamos interesados en el estudio de una variable aleatoria X, cuya distribución F, es en mayor o menor grado desconocida.

Algunos de los conceptos que debemos tener claros a la hora de hacer inferencia estadística son:

- Población.
- Muestra aleatoria simple.
- Realización muestral.
- Estadístico.
- Valor del estadístico.
- Distribución del estadístico en el muestreo.

Ejemplo:



9.3. Un ejemplo: la paradoja de Mèré

Para entender en qué consiste la estimación puntual proponemos resolver un ejemplo clásico: la paradoja de Mèré. Aunque no está claro cuanto hay de cierto en la historia, se cree que el Caballero de Mèré era muy aficionado al juego y que, basándose en su propia experiencia, proponía la siguiente apuesta:

De Mèré gana si al tirar 4 veces un dado sale al menos un 6

1. ¿Crees que era un juego "rentable" para el Caballero de Mèré?
2. ¿Cómo estimarías la probabilidad de ganar el juego?
3. ¿Cómo generarías una muestra?
4. ¿Cuál es la distribución en el muestreo del estimador?
5. ¿Sabrías calcular la probabilidad de ganar el juego?

Lo primero que debemos tener en cuenta es que estamos ante un problema de inferencia paramétrica. El parámetro desconocido p no es más que

$$p = \textit{"Probabilidad de ganar el juego"}$$

Aunque como veremos al final de la práctica podemos calcular el valor exacto de p sin más que aplicar combinatoria, el objetivo es determinar un estimador $\hat{p}$. Para ello simulamos n partidas y tomamos

$$\hat{p} = \frac{n_G}{n} = \frac{\textit{"Número de partidas ganadas"}}{\textit{"Número de partidas jugadas"}}.$$

9.3.1. Programa R

En el siguiente programa se simulan mil partidas consistentes en lanzar un dado cuatro veces. Cada partida se cuenta como ganada si se ha sacado al menos un seis. Al final se calcula el estimador $\hat{p}$ como el cociente entre el número de partidas ganadas y el número de partidas jugadas.

```
> n.veces=4
> partidas=1000  #partidas=tamaño muestral
> dados=matrix(sample(1:6,n.veces*partidas,T),nc=n.veces)
> ganadas=sum(apply(dados==6,1,sum)>=1)
> prob.est=ganadas/partidas
```

Fíjate que el valor del estimador $\hat{p}$ depende de la muestra. Cada vez que generemos partidas obtendremos un valor distinto de $\hat{p}$.

9.3.2. Distribución del estadístico en el muestreo

Como acabamos de comentar el valor del estimador $\hat{p}$ depende de la muestra. ¿Cuál es la distribución en el muestreo del estadístico? Como sabemos por el Teorema Central del Límite,

$$\frac{\hat{p}-p}{\sqrt{\frac{p(1-p)}{n}}} \to N(0,1),$$

siendo p la probabilidad de ganar el juego.

En principio p es desconocido. Sin embargo, podemos calcular su valor aplicando combinatoria y comprobar que efectivamente la distribución del estadístico se aproxima a una normal con la media y varianza dadas en la expresión anterior.

$$p = 1 - \left(\frac{5}{6}\right)^4 = 0{,}5177.$$

```
> n.veces=4
> partidas=1000  #partidas=tamaño muestral
> n.muestras<-100
> prob.est<-numeric()
> for (i in 1:n.muestras){
  dados=matrix(sample(1:6,n.veces*partidas,T),nc=n.veces)
  ganadas=sum(apply(dados==6,1,sum)>=1)
  prob.est[i]=ganadas/partidas}

> plot(density(prob.est))
> p=1-625/1296
> desv=sqrt(p*(1-p)/partidas)
> x=seq(-0.6,0.6,length=1000)
> lines(x,dnorm(x,mean=p,sd=desv),col=2)
```

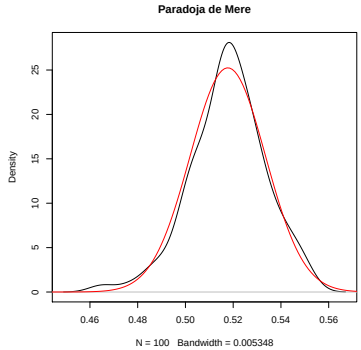


Figura 9.1: Se representan la estimación de la densidad de $\hat{p}$ obtenida a partir de 100 valores de $\hat{p}$ y la densidad de una normal de media p y varianza $p(1-p)/n$.

Capítulo 10

Tests de bondad de ajuste

10.1. Introducción

El objetivo de esta práctica es asignar un modelo de probabilidad a un conjunto de datos. De esta manera, el modelo elegido puede interpretarse como la distribución que ha generado dichos datos. Una vez que hemos verificado que no hay discrepancia entre que los datos y la distribución elegida es de interés: (1) estimar los parámetros de la distribución de probabilidad en función de los datos; (2) construir intervalos de confianza para los parámetros del modelo basados en dichas estimaciones y (3) predecir valores futuros.

Al proceso de búsqueda de un modelo de probabilidad a partir de la muestra de datos se le suele denominar ajuste de una distribución. Para que un modelo de probabilidad pueda considerarse razonable para explicar los datos, hemos de realizar algún tipo de pruebas estadísticas. Este proceso se denomina diagnosis o crítica del modelo. Diremos que un modelo explica bien nuestros datos si supera con éxito el proceso de diagnosis. La forma habitual de ajustar un modelo de probabilidad se basa en obtener las principales características de la muestra y seleccionar un conjunto de distribuciones candidatas (normal, exponencial, gamma, Weibull, log-normal, *etc.*) A continuación se realiza el ajuste del modelo y se evalúa si el ajuste es bueno mediante algún tipo de test estadístico, por ejemplo, el test χ^2 de Pearson o el test de Kolmogorov-Smirnov.

Utilizaremos un conjunto de datos que está en el fichero de nombre *Tiempoaccesoweb.txt*. La primera variable tiene 55 medidas del tiempo, en segundos, que se tarda en acceder a la página web de la Universidad de Santiago desde un ordenador de su biblioteca (intranet). La segunda variable tiene otras 55 medidas del tiempo, en segundos, que se tarda en acceder a la página web de la Universidad de Santiago desde un ordenador de una casa. Vamos a utilizar la segunda muestra, y como a partir de ella, podemos encontrar varias distribuciones candidatas para ajustar y que nos sirvan como modelo de distribución para saber el tiempo que tardamos cada vez que abrimos la página Web de la Universidad con ordenadores de su biblioteca.

10.2. Conjunto de datos: Tiempo de acesso a la web desde biblioteca

10.2.1. Análisis exploratorio de los datos

Lo primero que vamos a hacer es importar un fichero de datos que se encuentra en WebCT. Para ello, debemos escribir:

```
> tiem=read.table("c:/temp/Tiempoaccesoweb.txt",dec=",")
> tiempos=as.vector(tiem$V2,mode="numeric")
```

Vemos que tenemos una variable de 55 datos correspondientes a los tiempos de acesso a la página web. Representamos en primer lugar un histograma de los datos:

```
> n.cl=8
> puntos=min(tiempos)+(0:n.cl)*(max(tiempos)-min(tiempos))/n.cl
> hist(tiempos,breaks=puntos,freq=FALSE)
```

y observamos que la variable es bastante simétrica. Calculamos las principales medidas estadísticas:

```
> mean(tiempos)
  [1] 1.425455
> median(tiempos)
  [1] 1.42
```

Vemos que la media y la mediana son muy próximas indicando que la variable puede ser bastante simétrica. Veamos algunos cuantiles:

```
> c.tiempo=as.numeric(quantile(tiempos,probs=c(.1,.25,.5,.75,.9)))
> c.tiempo
  1.268 1.345 1.420 1.495 1.606
```

Las diferencias entre los cuantiles a la mediana son bastante parecidas para cuantiles emparejados lo que puede confirmar la simetría:

```
> c.tiempo[2]-c.tiempo[3]
  [1] -0.075
> c.tiempo[3]-c.tiempo[4]
  [1] -0.075
> c.tiempo[1]-c.tiempo[3]
  [1] -0.152
> c.tiempo[3]-c.tiempo[5]
  [1] -0.186
```

A continuación calculamos medidas de variabilidad. Los valores obtenidos parecen demostrar que la distribución puede ser bastante simétrica al no existir una gran diferencia entre la desviación típica y la desviación absoluta respecto a la media (mad).

```
> var(tiempos)
  [1] 0.01543636
> sd(tiempos)
  [1] 0.1242432
> mad(tiempos)
  [1] 0.118608
> IQR(tiempos)
  [1] 0.15
> range(tiempos)
  [1] 1.16 1.68
> diff(range(tiempos))
  [1] 0.52
```

Por último, veamos la asimetría y la curtosis:

```
> mean((tiempos-mean(tiempos))$^3$)/sd(tiempos)$^3$
  [1] 0.0812926
> mean((tiempos-mean(tiempos))$^4$)/var(tiempos)$^2$
  [1] 2.524455
```

Alternativamente, podemos importar las funciones de las librerías *moments* o *fBasics*.

Vemos que la variable es bastante simétrica, pero la curtosis es algo más pequeña de 3, que es la correspondiente a la normal. Esto puede ser muy común si sólo tenemos 55 datos, por lo que una candidata clara a ser la distribución generadora de los datos es la normal. Otras distribuciones como, por ejemplo, la Weibull, la log-normal o la gamma son distribuciones que también suelen ajustarse a tiempos de espera de este estilo. Además también son capaces de adecuarse a pequeñas asimetrías, como las que pueden presentar estos datos. Vemos a continuación cómo hacer los ajustes y la diagnosis del modelo.

10.2.2. Ajuste del modelo

En primer lugar, trabajamos con la distribución normal. Para ello, ajustamos la distribución normal a los datos, lo cual significa estimar los parámetros μ y σ de la distribución $N(\mu, \sigma)$. Para ello, importamos la librería *MASS* de R:

```
> library(MASS)
> ?fitdistr
> ajuste.normal=fitdistr(tiempos,"normal")
> ajuste.normal
       mean          sd
  1.42545455   0.12310850
 (0.01659995) (0.01173793)
```

Vemos cuáles son las estimaciones máximo verosimiles de los parámetros, que coinciden prácticamente con el resultado que da la función *mean* y la función *sd*. Esta última función da la cuasi-desviación típica, en lugar de la desviación típica y por eso existe esta pequeña diferencia. Guardamos los valores obtenidos en *mu.tiempos* y *sd.tiempos*:

```
> mu.tiempos=as.numeric(ajuste.normal$estimate[1])
> sd.tiempos=as.numeric(ajuste.normal$estimate[2])
```

Una vez que hemos hecho el ajuste, vamos a hacer tres gráficos, que nos van a permitir comparar: (1) el histograma de los tiempos con respecto a la densidad de la normal; (2) la función de distribución empírica de los datos con respecto a la función de distribución de una normal y (3) los cuantiles de los datos con respecto a los cuantiles de la distribución normal. Recordamos que la función de distribución empírica de una muestra está dada por:

$$F_n(x) = \begin{cases} 0 & \text{si } x < x_{(1)}, \\ \frac{r}{n} & \text{si } x_{(r)} \leq x < x_{(r+1)}, \\ 1 & \text{si } x_{(n)} \leq x. \end{cases}$$

Para hacer los gráficos recordamos que el valor mínino y el máximo de los datos eran 1.160 y 1.680, respectivamente. Entonces:

```
> par(mfrow=c(1,3))
> x=seq(1.16,1.68,by=.01)
> hist(tiempos,breaks=puntos,freq=FALSE)
> lines(x,dnorm(x,mu.tiempos,sd.tiempos))
> plot(ecdf(tiempos),do.points=F)
> lines(x,pnorm(x,mu.tiempos,sd.tiempos))
> tiempos.tip=(tiempos-mean(tiempos))/sd(tiempos)
> qqnorm(tiempos.tip)
> abline(0,1)
```

Todo ello parece indicar que la distribución normal puede ser una clara candidata para los datos. Pero no tiene porque ser la única que podemos usar para el ajuste. Como hemos dicho antes, también podemos intentar la Weibull o la log-normal. Empezamos con la Weibull. Recordamos que la distribución Weibull tiene dos parámetros, que aquí aparecen con los nombres de *shape* y *scale*, respectivamente.

```
> ?rweibull
> ajuste.weibull=fitdistr(tiempos,"weibull")
> ajuste.weibull
        shape          scale
  12.33899365    1.48241754
 ( 1.24243723) ( 0.01716987)
```

Guardamos esos valores en *shape.tiempos* y *scale.tiempos*:

```
> shape.tiempos=as.numeric(ajuste.weibull$estimate[1])
> scale.tiempos=as.numeric(ajuste.weibull$estimate[2])
```

Una vez que hemos hecho el ajuste, vamos a hacer los tres gráficos que hemos visto anteriormente, que nos permiten comparar: (1) el histograma de los tiempos con respecto a la densidad de la Weibull; (2) la función de distribución empírica de los datos con respecto a la función de distribución de una Weibull y (3) los cuantiles de los datos con respecto a los cuantiles de la distribución Weibull. Entonces:

```
> par(mfrow=c(1,3))
> hist(tiempos,breaks=puntos,freq=FALSE)
> lines(x,dweibull(x,shape.tiempos,scale.tiempos))
> plot(ecdf(tiempos),do.points=F)
> lines(x,pweibull(x,shape.tiempos,scale.tiempos))
> qqplot(tiempos,rweibull(1000,shape.tiempos,scale.tiempos))
```

También podemos probar la log-normal. Recordamos que la distribución log-normal tiene dos parámetros, que aquí aparecen con los nombre de *meanlog* y *sdlog*.

```
> ?rlnorm
> ajuste.lognormal=fitdistr(tiempos,"log-normal")
> ajuste.lognormal
      meanlog         sdlog
  0.350742387   0.086756565
 (0.011698253) (0.008271914)
```

Guardamos esos valores en *meanlog.tiempos* y *sdlog.tiempos*:

```
> meanlog.tiempos=as.numeric(ajuste.lognormal$estimate[1])
> sdlog.tiempos=as.numeric(ajuste.lognormal$estimate[2])
```

Una vez que hemos hecho el ajuste, vamos a hacer los tres gráficos que hemos visto anteriormente, que nos permiten comparar: (1) el histograma de los tiempos con respecto a la densidad de la log-normal; (2) la función de distribución empírica de los datos con respecto a la función de distribución de una log-normal; y (3) los cuantiles de los datos con respecto a los cuantiles de la distribución log-normal. Entonces:

```
> par(mfrow=c(1,3))
> hist(tiempos,breaks=puntos,freq=FALSE)
> lines(x,dlnorm(x,meanlog.tiempos,sdlog.tiempos))
> plot(ecdf(tiempos),do.points=F)
> lines(x,plnorm(x,meanlog.tiempos,sdlog.tiempos))
> qqplot(tiempos,rlnorm(1000,meanlog.tiempos,sdlog.tiempos))
```

10.2.3. Diagnosis del modelo

Vamos a utilizar dos contrastes de hipótesis sobre la distribución de la variable tiempos. El primero es el contraste χ^2 de Pearson. En primer lugar, este contraste especifica la hipótesis nula H_0 de que la distribución generadora de los datos es F_0 y una hipótesis alternativa H_1 de que la distribución generadora de los datos no es F_0. Este contraste está basado en la comparación de las frecuencias absolutas observadas con la muestra comparadas con las frecuencias absolutas esperadas con la muestra. El contraste se construye como sigue:

1. Crear k clases como hemos hecho para el histograma y calcular O_i, la frecuencia absoluta de datos en la clase i.

2. Calcular la probabilidad p_i que la distribución F_0 asigna a cada clase. Entonces, el número de datos esperados en la clase i es $E_i = np_i$.

3. Calcular la discrepancia entre lo observado y lo esperado mediante:

$$X^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

   que se distribuye como una χ^2 cuando la distribución especificada F_0 es correcta. Los grados de libertad de esta distribución son $k - r - 1$ donde r es el número de parámetros estimados de la distribución F_0.

4. La hipótesis nula H_0 (la distribución generadora de los datos es F_0) se rechaza si la probabilidad de obtener una discrepancia mayor o igual que la observada es lo suficientemente baja. En otras palabras, cuando $X^2 \geq \chi^2_{k-r-1,\alpha}$ para un cierto α pequeño.

Vamos a realizar este contraste para nuestros datos. Para ello, definimos el número de clases *n.cl* y a las frecuencias de cada clase O:

```
> n.cl=8
> puntos=min(tiempos)+(0:n.cl)*(max(tiempos)-min(tiempos))/n.cl
> O=table(cut(tiempos,breaks=puntos))
> O
  (1.16,1.22] (1.22,1.29] (1.29,1.35] (1.35,1.42] (1.42,1.48] (1.48,1.55]
  (1.55,1.61] (1.61,1.68]
   2 6 6 14 10 5 5 6
```

A continuación debemos calcular las frecuencias absolutas esperadas para la primera distribución candidata que es la $N(1{,}4254, 0{,}1231)$. Para ello, definimos el vector E:

```
> library(MASS)
> ajuste.normal=fitdistr(tiempos,"normal")
```

```
> mu.normal=as.numeric(ajuste.normal$estimate[1])
> sd.normal=as.numeric(ajuste.normal$estimate[2])
> E=vector()
> E[1]=pnorm(puntos[2],mu.normal,sd.normal)*55
> E[8]=(1-pnorm(puntos[8],mu.normal,sd.normal))*55
> for (i in 2:7){E[i]=(pnorm(puntos[i+1],
  mu.normal,sd.normal)-pnorm(puntos[i],mu.normal,sd.normal))*55}
> E
  [1] 2.845329 4.612880 8.137625 10.932312 11.185050 8.715198
  [7] 5.171429 3.400177
```

A continuación podemos obtener el valor del estadístico como sigue:

```
> X2=sum(((O-E)^2)/E)
> X2
  [1] 4.763598
> gdl=8-2-1
  [1] 5
> p.valor=1-pchisq(X2,gdl)
  [1] 0.4454068
```

Como el p-valor es 0.4454068, que es mayor que 0.05, no podemos rechazar la hipótesis nula. A continuación probamos con la gamma.

```
> ajuste.gamma=fitdistr(tiempos,"gamma")
> shape.gamma=as.numeric(ajuste.gamma$estimate[1])
> rate.gamma=as.numeric(ajuste.gamma$estimate[2])
> E=vector()
> E[1]=pgamma(puntos[2],shape.gamma,rate.gamma)*55
> E[8]=(1-pgamma(puntos[8],shape.gamma,rate.gamma))*55
> for (i in 2:7)\{E[i]=(pgamma(puntos[i+1],shape.gamma,rate.gamma)-
  pgamma(puntos[i],shape.gamma,rate.gamma))*55\}
> E
  [1] 2.561262 4.829285 8.586916 11.184181 10.964439 8.285439 4.928234
      3.660244
```

A continuación podemos obtener el valor del estadístico como sigue:

```
> X2=sum(((O-E)^2)/E)
> X2
  [1] 3.967707
> gdl=8-2-1
  [1] 5
```

```
> p.valor=1-pchisq(X2,gdl)
  [1] 0.5540747
```

Como el p-valor es 0.5540747, que es mayor que 0.05, no podemos rechazar la hipótesis nula.

El segundo test que podemos considerar es el de Kolmogorov-Smirnov. Al igual que el contraste χ^2, este contraste especifica la hipótesis nula H_0 de que la distribución generadora de los datos es F_0 y una hipótesis alternativa H_1 de que la distribución generadora de los datos no es F_0. El contraste está basado en la comparación de la función de distribución empírica y la función de distribución. El contraste se construye como sigue:

1. En primer lugar, se ordenan los datos de manera creciente, es decir, se obtienen $x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n)}$.
2. A continuación, se obtiene la función de distribución empírica:
$$F_n(x) = \begin{cases} 0 & \text{si } x < x_{(1)}, \\ \frac{r}{n} & \text{si } x_{(r)} \leq x < x_{(r+1)}, \\ 1 & \text{si } x_{(n)} \leq x. \end{cases}$$
3. Por último, se calcula la discrepancia máxima entre las funciones de distribución empírica, $F_n(x)$, y la teórica, $F_0(x)$, con el estadístico:
$$D_n = \text{máx} \left| F_n(x) - F_0(x) \right|.$$

La distribución de D_n no tiene forma estándar pero se conocen los valores numéricos. Este contraste es más sencillo de realizar que el contraste χ^2 de Pearson ya que no tenemos que fijar el número de clases. Por el contrario, es un contraste conservador ya que tiende a aceptar la hipótesis nula con bastante frecuencia. Vamos a ver qué ocurre con nuestros datos. Empezamos realizando el contraste para la distribución normal. Vemos la comparación entre las funciones de distribución empírica y la normal:

```
> x=seq(1.16,1.68,by=.01)
> plot(ecdf(tiempos),do.points=F)
> lines(x,pnorm(x,mu.tiempos,sd.tiempos))
```

A continuación, vemos el resultado del contraste:

```
> ks.test(tiempos,"pnorm",mu.normal,sd.normal)
  One-sample Kolmogorov-Smirnov test
  data: tiempos
  D = 0.0937, p-value = 0.7196
  alternative hypothesis: two-sided
  Warning message:
  cannot compute correct p-values with ties in: ks.test(tiempos,"pnorm",
  mu.normal, sd.normal)
```

Igual que antes, como el p-valor es 0.7196, que es mayor que 0.05, no podemos rechazar la hipótesis nula. A continuación probamos con la gamma. Vemos primero la comparación de las funciones de distribución empírica y la de la gamma:

```
> x=seq(1.16,1.68,by=.01)
> plot(ecdf(tiempos),do.points=F)
> lines(x,pgamma(x,shape.gamma,rate.gamma))
```

A continuación, hacemos el contraste:

```
> ks.test(tiempos,"pgamma",shape.gamma,rate.gamma)
  One-sample Kolmogorov-Smirnov test
  data: tiempos
  D = 0.0929, p-value = 0.7301
  alternative hypothesis: two-sided
  Warning message:
  cannot compute correct p-values with ties in: ks.test(tiempos,"pgamma",
  shape.gamma, rate.gamma)
```

Igual que antes, como el p-valor es 0.7301, que es mayor que 0.05, no podemos rechazar la hipótesis nula.

En resumen, ambos contrastes aceptan las dos distribuciones, con lo que con ambas distribuciones podemos obtener ajustes adecuados. Como regla, podemos decir que la gamma es ligeramente superior ya que los p-valores de los contrastes con esta distribución son ligeramente mayores que con la distribución normal. Podemos hacer un gráfico de las dos densidades juntas comparadas con el histograma de la muestra:

```
> x=seq(1.16,1.68,by=.01)
> hist(tiempos,breaks=puntos,freq=FALSE)
> lines(x,dnorm(x,mu.tiempos,sd.tiempos))
> lines(x,dgamma(x,shape.gamma,rate.gamma))
```

Este gráfico parece confirmar lo que decíamos anteriormente. Por último, con estas distribuciones podemos predecir la probabilidad del tiempo de espera hasta que se conecta un ordenador de la biblioteca a la página web de la Universidad de Santiago. Para ello, por ejemplo, podemos calcular cuál es la probabilidad de esperar como mucho 1.60 seg.:

```
> pnorm(1.60,mu.tiempos,sd.tiempos)
  [1] 0.921878
> pgamma(1.60,shape.gamma,rate.gamma)
  [1] 0.9176931
```

Como podemos ver los resultados son muy parecidos ya que la diferencia entre uno y otro es de:

```
> pnorm(1.60,mu.tiempos,sd.tiempos)-pgamma(1.60,shape.gamma,rate.gamma)
  [1] 0.004184921
```

Capítulo 11

Intervalos de confianza y contrastes de hipótesis

11.1. Introducción

El objetivo de esta práctica es utilizar R para construir intervalos de confianza y realizar contrastes de hipótesis sobre parámetros como la media o varianza en poblaciones normales. Como ya sabemos, uno de los objetivos de la inferencia paramétrica es estimar un determinado parámetro desconocido de la población que es objeto de estudio. Es lo que conocemos como estimación puntual. En algunas ocasiones, en lugar de proporcionar un estimador del parámetro, nos puede interesar dar un intervalo numérico que con cierta "seguridad" contiene al valor del parámetro en la población. Por otra parte, nos puede interesar comprobar empíricamente alguna hipótesis inicial sobre la población en estudio a partir de la información extraída de la muestra. Este es el objetivo de los contrastes de hipótesis.

11.2. Distribuciones asociadas al muestreo en poblaciones normales

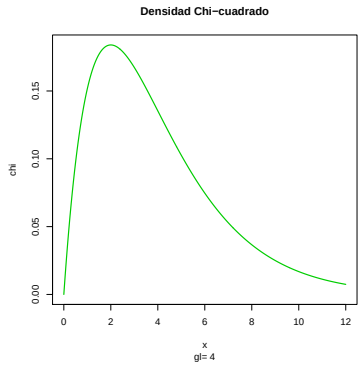
Dentro de la estimación en poblaciones normales aparecen una serie de distribuciones como la χ^2 o la t de Student que cobran gran importancia en la construcción de intervalos de confianza y en los contrastes de hipótesis. A continuación se explican dichas distribuciones y las funciones de R relacionadas con ellas.

11.2.1. Distribución χ^2

Sean $X_1, \ldots, X_n \in N(0,1)$ independientes. Diremos que la variable aleatoria

$$\chi^2_n = X_1^2 + \cdots + X_n^2$$

sigue una distribución χ^2 con n grados de libertad.



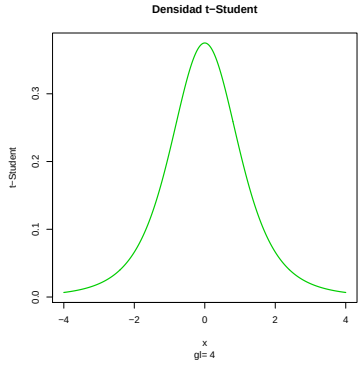

```
> ngl<-4
> x<-seq(0,ngl*3,length=5000)
> plot(x,dchisq(x,ngl),
  main=c("Densidad Chi-cuadrado"),
  sub=paste("gl=",ngl),
  ylab="chi",type="l",col=3)
```

11.2.2. Distribución t de Student

Sean $X, X_1, \ldots, X_n \in N(0,1)$ independientes. Diremos que la variable aleatoria

$$t_n = \frac{X}{\sqrt{\frac{X_1^2+\cdots+X_n^2}{n}}}$$

sigue una distribución t de Student con n grados de libertad.



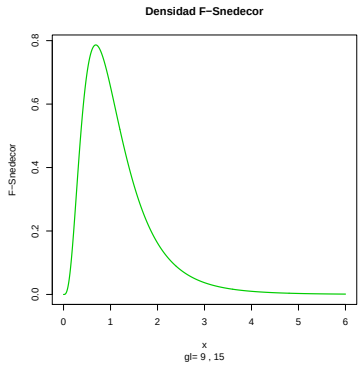

```
> ngl<-4
> x<-seq(-4,4,length=5000)
> plot(x,dt(x,ngl),
  main=c("Densidad t-Student"),
  sub=paste("gl=",ngl),
  ylab="t-Student",type="l",col=3)
```

11.2.3. Distribución F de Snedecor

Sean $X_1, \ldots, X_n, Y_1, \ldots, Y_m \in N(0,1)$ independientes. Diremos que la variable aleatoria

$$F_{n,m} = \frac{\frac{X_1^2+\cdots+X_n^2}{n}}{\frac{Y_1^2+\cdots+Y_m^2}{m}}$$

sigue una distribución F de Snedecor con n grados de libertad en el numerador y m grados de libertad en el denominador.

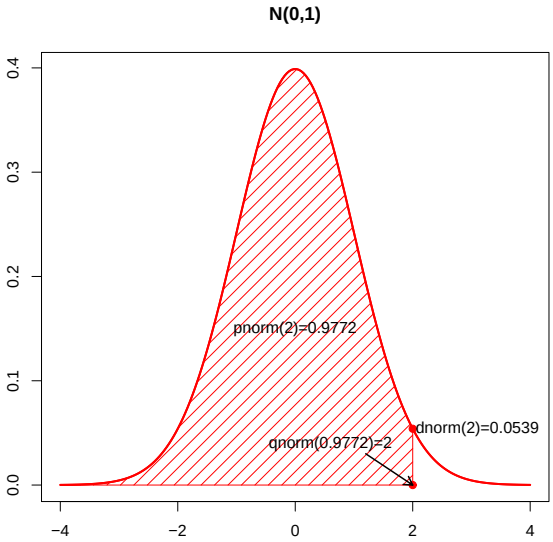



```
> gl.n<-9
> gl.d<-15
> x<-seq(0,6,length=5000)
> plot(x,df(x,gl.n,gl.d),
  main=c("Densidad F-Snedecor"),
  sub=paste("gl=",gl.n,",",gl.d),
  ylab="F-Snedecor",type="l",col=3)
```

11.2.4. Funciones en R

Las principales funciones que necesitamos conocer en R para trabajar con las distribuciones explicadas anteriormente son:

Distribución	Funciones en R
Normal	`rnorm`, `dnorm`, `pnorm`, `qnorm`
χ^2	`rchisq`, `dchisq`, `pchisq`, `qchisq`
t-Student	`rt`, `dt`, `pt`, `qt`
F-Snedecor	`rf`, `df`, `pf`, `qf`



11.3. Distribuciones de los estadísticos en el muestreo

Una vez vistas las principales distribuciones asociadas al muestreo de poblaciones normales, podemos empezar a trabajar con los distintos estadísticos que conocemos.

A partir de ahora, $X_1, \ldots, X_n$ denota una muestra aleatoria simple de $N(\mu, \sigma^2)$.

11.3.1. Estimadores de la media y varianza

La siguiente tabla resume las propiedades de los principales estimadores de media y varianza. Recuerda que las distribuciones de los estadísticos que se muestran son válidas sólo cuando la población de partida es normal.

Media	$\bar{X} = \frac{1}{n}\sum_{i=1}^n X_i$	$\mathbb{E}(\bar{X}) = \mu$	$\mathrm{Var}(\bar{X}) = \frac{\sigma^2}{n}$	$\bar{X} \sim N(\mu, \frac{\sigma^2}{n})$
Varianza	$S^2 = \frac{1}{n}\sum_{i=1}^n (X_i - \bar{X})^2$	$\mathbb{E}(S^2) = \frac{n-1}{n}\sigma^2$	$\mathrm{Var}(S^2) = \frac{2(n-1)}{n^2}\sigma^4$	$\frac{nS^2}{\sigma^2} \sim \chi^2_{n-1}$
	$\hat{S}^2 = \frac{1}{n-1}\sum_{i=1}^n (X_i - \bar{X})^2$	$\mathbb{E}(\hat{S}^2) = \sigma^2$	$\mathrm{Var}(\hat{S}^2) = \frac{2}{n-1}\sigma^4$	$\frac{(n-1)\hat{S}^2}{\sigma^2} \sim \chi^2_{n-1}$
	$S^2_\mu = \frac{1}{n}\sum_{i=1}^n (X_i - \mu)^2$	$\mathbb{E}(S^2_\mu) = \sigma^2$	$\mathrm{Var}(S^2_\mu) = \frac{2}{n}\sigma^4$	$\frac{nS^2_\mu}{\sigma^2} \sim \chi^2_n$

Tabla 11.1: Estimadores para la media y varianza. El estimador denotado por S^2_μ sólo puede ser calculado cuando la media μ es conocida.

11.3.2. Distribución de estadísticos en el muestreo

En la siguiente tabla se resumen los principales estadísticos y su distribución en el muestreo.

Problema de inferencia	**Estadístico**	**Distribución**
Una proporción	$\dfrac{\hat{p} - p}{\sqrt{\frac{p\,(1-p)}{n}}}$	$N(0,1)$
Media con varianza conocida	$\dfrac{\bar{X} - \mu}{\sigma/\sqrt{n}}$	$N(0,1)$
Media con varianza desconocida	$\dfrac{\bar{X} - \mu}{\hat{S}/\sqrt{n}}$	t_{n-1}

Varianza con media conocida	$\frac{n\,S_\mu^2}{\sigma^2}$	χ_n^2
Varianza con media desconocida	$\frac{(n-1)\,\hat{S}^2}{\sigma^2}$	χ_{n-1}^2
Diferencia de proporciones	$\frac{\hat{p}_X - \hat{p}_Y - (p_X - p_Y)}{\sqrt{\frac{\hat{p}_X(1-\hat{p}_X)}{n} + \frac{\hat{p}_Y(1-\hat{p}_Y)}{m}}}$	$N(0,1)$
Diferencia de medias con varianzas conocidas	$\frac{\bar{X} - \bar{Y} - (\mu_X - \mu_Y)}{\sqrt{\frac{\sigma_X^2}{n} + \frac{\sigma_Y^2}{m}}}$	$N(0,1)$
Diferencia de medias con varianzas desconocidas pero iguales	$\frac{\bar{X} - \bar{Y} - (\mu_X - \mu_Y)}{\hat{S_T}\sqrt{\frac{1}{n} + \frac{1}{m}}}$	t_{n+m-2}
Diferencia de medias con varianzas desconocidas (muestras grandes: $n > 30$ y $m > 30$)	$\frac{\bar{X} - \bar{Y} - (\mu_X - \mu_Y)}{\sqrt{\frac{\hat{S}_X^2}{n} + \frac{\hat{S}_Y^2}{m}}}$	$N(0,1)$
Cociente de varianzas con medias conocidas	$\frac{S_{\mu_X}^2}{S_{\mu_Y}^2} \cdot \frac{\sigma_Y^2}{\sigma_X^2}$	$F_{n\,,m}$
Cociente de varianzas con medias desconocidas	$\frac{\hat{S}_X^2}{\hat{S}_Y^2} \cdot \frac{\sigma_Y^2}{\sigma_X^2}$	$F_{(n-1)\,,(m-1)}$

11.4. Intervalos de confianza y contrastes de hipótesis

Este apartado está dedicado a la utilización de R para la construcción de intervalos de confianza y la realización de contrastes de hipótesis. El ejemplo que se muestra a continuación calcula los intervalos de confianza y realiza el contraste de hipótesis para la media, tanto para el caso en que se conoce la varianza de la población como para el caso en que no se conoce. También calcula intervalos de confianza y realiza el contraste de hipótesis para la diferencia de medias cuando se introducen 2 muestras.

```
# CONTRASTES SOBRE LA MEDIA
# if (is.null(y)) sólo tenemos una muestra
# if (is.null(var)) no conocemos la varianza
# var.equal=FALSE dos muestras con varianzas distintas
```

```
mean.test<-function(x,y=NULL,var.x=NULL,var.y=NULL,conf.level=0.95,
                    mu=0,var.equal=FALSE){
  alpha<-1-conf.level
# Una muestra
  if (is.null(y)){
    n<-length(x)
# Media con varianza desconocida
    if (is.null(var.x)){
      t.test(x,conf.level=conf.level,mu=mu)
    }
# Media con varianza conocida
    else{
      sigma<-sqrt(var.x)
      z<-qnorm(c(1-alpha/2,alpha/2))
      interval<-c(mean(x)-z[1]*sigma/sqrt(n),mean(x)-z[2]*sigma/sqrt(n))
      list(test="Contraste para la media con varianza conocida",
      length=n,conf.level=conf.level,interval=interval)
    }
  }
# Dos muestras independientes
  else{
# Varianzas desconocidas
    if (is.null(var.x)){
# Varianzas desconocidas pero iguales
    if (var.equal){
      t.test(x,y,conf.level=conf.level,var.equal=TRUE)
    }
# Varianzas deconocidas y distintas (muestras pequeñas)
    else if ((length(x)<30) || (length(y)<30)){
      t.test(x,y,conf.level=conf.level)}
    }
# Varianzas conocidas
    else{
      n<-length(x)
      m<-length(y)
      z<-qnorm(c(1-alpha/2,alpha/2))
      dev<-sqrt(var.x/n+var.y/m)
      interval<-c((mean(x)-mean(y))-z[1]*dev,(mean(x)-mean(y))-z[2]*dev)
      list(test="Contraste para la diferencia de medias con varianzas
      conocidas",length=c(n,m),conf.level=conf.level,interval=interval)
    }
  }
}
```

Capítulo 12

El modelo de regresión lineal simple

12.1. Introducción

En esta práctica aprenderemos a utilizar R como herramienta para la estimación y diagnosis de modelos de regresión. La situación general es la siguiente. Disponemos de una variable aleatoria respuesta Y, que supondremos relacionada con otra variable X, que llamaremos explicativa o independiente. A partir de una muestra de n individuos para los que se dispone de los valores de ambas variables, $\{(X_i, Y_i), i = 1, \ldots, n\}$, podemos visualizar gráficamente la relación existente entre ambas. Así, utilizando la función *plot* de R podemos realizar un *gráfico de dispersión*, en el que los valores de la variable X se disponen en el eje horizontal y los de Y en el vertical. En la Figura 12.1 se muestran ejemplos de gráficos de dispersión.

- ¿Qué conclusiones podrías sacar a partir de las gráficas sobre la relación entre las variables X e Y en cada ejemplo?
- Los puntos (X_i, Y_i) de la gráfica (a) han sido generados a partir del modelo lineal $Y_i = a + bX_i + \varepsilon_i$, con $\varepsilon_i \sim N(0, \sigma^2_\varepsilon)$. ¿Sabrías decir a que se debe que casi no se aprecie la relación lineal? Aunque los datos han sido generado a partir de la ecuación de una recta, el error ε que hemos introducido tiene una varianza muy grande. Eso hace que los datos aparezcan tan dispersos.
- ¿Existe relación lineal entre las variables X e Y representadas en la gráfica (b)? ¿Qué tipo de relación crees que existe?
- En la gráfica (c), los puntos (X_i, Y_i) han sido generados a partir del modelo lineal $Y_i = a + bX_i + \varepsilon_i$, pero con errores heterocedásticos, es decir, la varianza del error no es constante.
- En la gráficas (d) y (e), los puntos (X_i, Y_i) han sido generados a partir del modelo lineal $Y_i = a + bX_i + \varepsilon_i$, con $\varepsilon_i \sim N(0, \sigma^2_\varepsilon)$. ¿En que se diferencian ambos ejemplos?

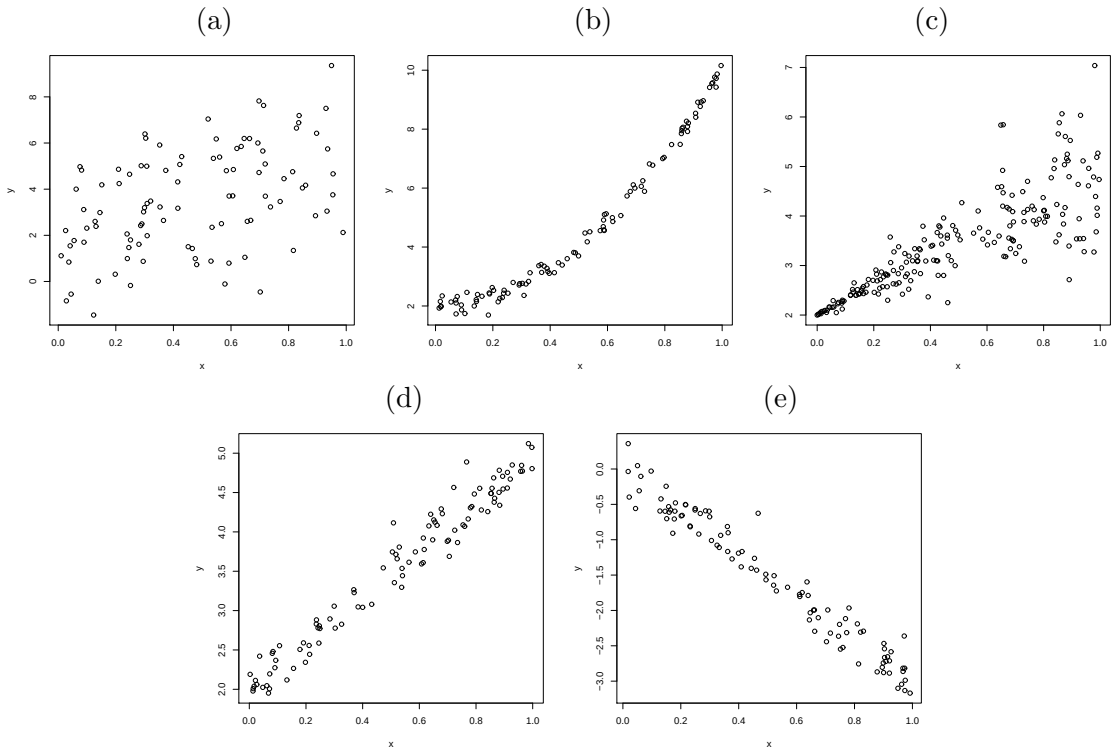


Figura 12.1: Ejemplos de gráficos de dispersión.

- ¿Podrías determinar ejemplos reales en los que la relación entre variables se ajuste a alguna de las gráficas mostradas?

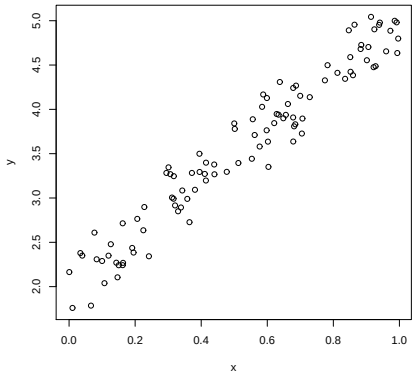
La práctica consistirá en:

1. Simular el modelo de regresión $Y = a + bX + \varepsilon$.
   - Generar las observaciones X_i, $i = 1, \ldots, n$.
   - Generar los errores $\varepsilon_i \sim N(0, \sigma^2_\varepsilon)$.
   - Generar las observaciones $Y_i = a + bX_i + \varepsilon_i$, $i = 1, \ldots, n$.
2. Dibujar Y frente a X.
3. Estimar el modelo de regresión: función *lm*.
4. Analizar el modelo estimado: función *summary*.

12.2. Simulación de observaciones

En la mayoría de situaciones prácticas toda la información de la que disponemos está en las observaciones muestrales. No tenemos ninguna información sobre el modelo subyacente y lo único que podemos hacer es asumir que los datos se ajustan a un determinado modelo y estimarlo (revisar la práctica dedicada a los test de bondad de ajuste). La ventaja al hacer simulación es que somos nosotros mismos los que generamos los datos y tenemos control sobre el modelo subyacente. Esto nos permite comparar los resultados obtenidos al estimarlo.

```
# Parametros del modelo
# y=ax+b+eps
n=100
a=2
b=3
desv=0.2
# Simulamos el modelo
eps=rnorm(n,sd=desv)
x=runif(n)
y=a+b*x+eps
# Grafico de dispersion
plot(x,y)
```



12.3. Estimación del modelo

La función de R que nos permite estimar un modelo de regresión lineal es la función *lm*. La forma de invocar a la función para estimar un modelo de regresión lineal simple es *lm(y∼x)*. Puedes consultar la ayuda de la función para ver todas las posibilidades que ofrece. Para nuestro ejemplo, obtenemos:

```
> lm(y~x)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      2.019        2.966
```

Como puedes comprobar, los valores obtenidos son las estimaciones de los coeficientes de la recta de regresión a y b.

$$\hat{b} = \frac{S_{X,Y}}{S_{X^2}} \qquad \hat{a} = \bar{Y} - \hat{b}\bar{X}$$

siendo

$$S_{X,Y} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})$$

$$S_X^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

Efectivamente,

```
> cov(x,y)/var(x)
[1] 2.965763
> mean(y)-cov(x,y)/var(x)*mean(x)
[1] 2.018518
```

que coinciden con los valores obtenidos por medio de la función *lm*. En lugar de invocar simplemente la función podemos guardar su resultado en una variable y veremos así que obtenemos más información.

```
> reg=lm(y~x)
> names(reg)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

Además de los coeficientes estimados, R devuelve también los residuos estimados $\hat{e}_i = Y_i - (\hat{a} + \hat{b}X_i)$, los grados de libertad de los residuos, los valores de $\hat{Y}_i = \hat{a} + \hat{b}X_i$,...

```
> reg$residuals
           1            2           3           4           5
 0.151171813 -0.042671979  0.310145653  0.339613218 -0.210517764
           6            7           8
 -0.08993393  0.210728129 -0.230327511
.....

> reg$fitted.values
       1        2        3        4        5        6        7        8
4.801958 3.942097 3.188916 3.501621 4.865023 3.079907 2.685361 4.941944
       9       10       11       12       13       14       15       16
4.030721 3.321547 4.038718 4.056115 4.113711 3.761337 2.049225 4.641683
.....

> reg$df.residual
[1] 98
```

12.4. Análisis del modelo

Para realizar el análisis del modelo estimado utilizaremos la función *summary*. Así

```
> summary(reg)

Call:
lm(formula = y ~ x)

Residuals:
      Min        1Q    Median        3Q       Max
-0.457422 -0.156148 -0.008925  0.153244  0.434384

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.01852    0.04559   44.27   <2e-16 ***
x            2.96576    0.07629   38.88   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2207 on 98 degrees of freedom
Multiple R-Squared: 0.9391,     Adjusted R-squared: 0.9385
F-statistic:  1511 on 1 and 98 DF,  p-value: < 2.2e-16
```

Obtenemos así información sobre los residuos estimados, los contrastes de regresión $H_0 : a = 0$, $H_0 : b = 0$, el coeficiente de determinación, ...

A la vista de los resultados:

- Podemos rechazar la hipótesis nula $H_0 : a = 0$. Asimismo rechazamos la hipótesis nula $H_0 : b = 0$. Es razonable si tenemos en cuenta que hemos generado la muestra a partir del modelo con $a = 2$ y $b = 3$. ¿Qué ocurre si repetimos la práctica generando los datos a partir del modelo con $a = 0$? ¿Cuál es el p-valor del estadístico de contraste en ese caso?

- El coeficiente de determinación en el modelo de regresión lineal simple coincide con el coeficiente de correlación al cuadrado, es decir

$$R^2 = \frac{S_{X,Y}^2}{S_X^2 S_Y^2}.$$

  Lo podemos comprobar mediante:

```
> cov(x,y)^2/(var(x)*var(y))
[1] 0.9391012
```

- De manera informal, el coeficiente de determinación nos sirve para decidir si el modelo de regresión lineal explica bien los datos o no. En nuestro caso es natural obtener un valor alto del coeficiente de determinación, pues los datos han sido generados a partir de un modelo lineal.
- ¿Cómo afecta el tamaño muestral a las estimaciones? ¿Y la varianza del error σ_ε^2?

También podemos hacer un gráfico con la recta de regresión estimada y la recta original a partir de la cual se ha generado la muestra.

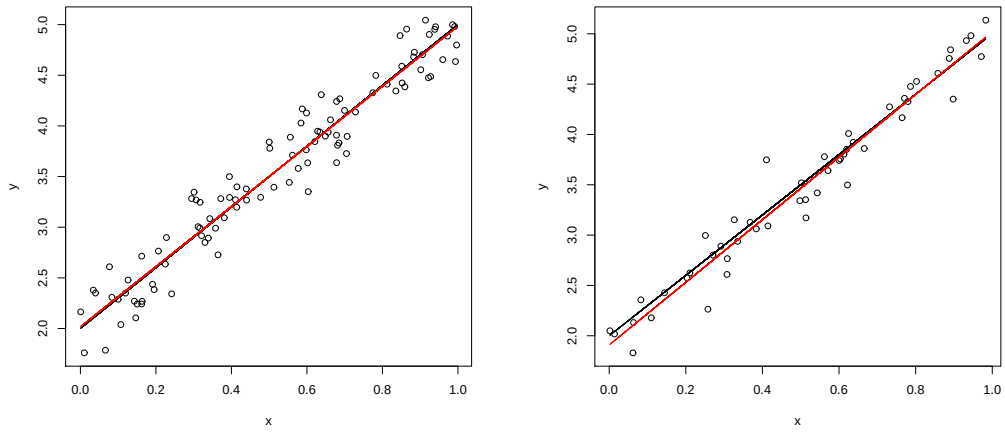


Figura 12.2: Se representan la recta $Y = 2 + 3X$ a partir de la que hemos generado los datos añadiendo el error y la recta de regresión estimada $Y = \hat{a} + \hat{b}X$. En la gráfica de la izquierda $n = 100$ y en la de la derecha $n = 50$.