# Guía Rápida R

María Santos

## 1. R.I

### Tema 0. Logística

- library(*paquete*): para cargar un paquete ya instalado
- install.packages(*paquete*, dep=TRUE): para instalar un paquete

### Tema 1. La calculadora

- $+$ : suma
- - : resta
- $*$ : multiplicación
- $/$ : división
- ^ : potencia
- %/%: cociente entero
- %%: resto división entera
- pi: número $\pi$
- sqrt($x$): raíz cuadrada de $x$
- exp(1): número $e$
  - Si en vez de 1 ponemos otro número $x$, R calcula $e^x$
- log($x$): logaritmo neperiano de $x$
- log10($x$): logaritmo en base 10 de $x$
- log($n$,$a$): logaritmo en base $a$ de $n$
- factorial($x$): $x$ factorial, $x!$
- choose($n, m$): número combinatorio de $n$ sobre $m$, $\binom{n}{m}$
- sin($x$): seno de $x$
  - El resultado de todas las funciones trigonométricas las da en radianes. Si lo queremos en grados, multiplicamos dicho resultado por $\frac{\pi}{180}$
- cos($x$): coseno de $x$
- tan($x$): tangente de $x$
- asin($x$): arcoseno de $x$
- acos($x$): arcocoseno de $x$
- atan($x$): arcotangente de $x$
- abs($x$): valor absoluto de $x$

- print($x$,$n$): muestra las $n$ cifras significativa del número $x$
- round($x$,$n$): redondea a $n$ cifras significativas un resultado o vector numérico $x$
  - En caso de empate, R redondea al valor que termina en cifra par
- floor($x$): parte entera por defecto de $x$
- ceiling($x$): parte entera por exceso de $x$
- trunc($x$): parte entera de $x$, eliminando la parte decimal
- *nombre de la variable*=*valor*: para definir una variable
- *nombre de la función*=function(*variable*){*función*}: para definir una función
  - Entre paréntesis las variables
  - Entre llaves las instrucciones
- ls(): nos da la lista de los objetos definidos hasta el momento
- rm($x$): borra la definición del objeto $x$ u objetos a los que se aplica
- rm(list=ls()): borrar todas las definiciones anteriores
- $a$+$b$i: número complejo
  - No hay que poner el operador $*$ entre el coeficiente imaginario $b$ e $i$
  - Si el coeficiente de i es 1 o $-1$, hay que escribirlo
  - complex(real=...,imaginary=...): otra forma de definir un número complejo
  - complex(modulus=...,argument=...): otra forma de definir un número complejo
- sqrt(as.complex(-$x$)): para calcular la raíz cuadrada de un número negativo $x$
- Re($x$): muestra parte real de un número complejo $x$
- Im($x$): muestra la parte imaginaria de un número complejo $x$
- Mod($x$): calcula el módulo de un número complejo $x$
- Arg($x$): calcula el argumento de un número complejo $x$
- Conj($x$): calcula el conjugado de un número complejo $x$

## Tema 2. Introducción a la regresión lineal

- lm($y$~$x$, data=*data frame*): para calcular la recta de regresión de $y$ en función de $x$
  - lm(log10($y$)~$x$): recta de regresión semilogarítmica
  - lm(log10($y$)~log10($x$)): recta de regresión doble logarítmica
- summary(lm(...)): para saber todo lo que ha calculado R con la función lm
  - summary(lm(...))$r.squared: nos da el valor del coeficiente de determinación de $R^2$
- abline(lm(...)): añade la recta de regresión al gráfico (ya creado) inmediatamente anterior
- curve($f(x)$, add=TRUE): para añadir la curva $f(x)$ al gráfico (ya creado) inmediatamente anterior

- **Tema 3. Vectores**

  ✎ Tipos de datos:
  - ◇ logical: lógicos (TRUE o FALSE)
  - ◇ integer: números enteros
  - ◇ numeric: números reales
  - ◇ complex: números complejos
  - ◇ character: palabras

  - ○ Todos los objectos de un mismo vector han de ser del mismo tipo
  - ○ Cuando queramos usar vectores formados por objetos de diferentes tipos, tendremos que usar lists
  - ○ Character gana a complex, que gana a numeric, que gana a integer, que gana a logical

  • c(...): para definir un vector
  - ○ Si queremos crear un vector de palabras, tenemos que entrarlas obligatoriamente entre comillas

  • scan(): para definir un vector
  - ∗ sep: sirve para indicar el símbolo usado para separar entradas consecutivas si no son espacios en blanco. Para ello se ha de igualar sep a este símbolo entrecomillado
  - ∗ dec: sirve para indicar el símbolo que separa la parte entera de la decimal en los números reales si no es un punto. Esto se indica igualando dec al símbolo correspondiente entre comillas
  - ∗ what: sirve para indicar a R de qué tipo tiene que considerar los datos que se le entren
  - ○ R abre un entorno de diálogo donde podemos ir entrando datos separados por espacios en blanco; cada vez que pulsemos Intro, R importará los datos que hayamos escrito desde la vez anterior en que la pulsamos y abrirá una nueva línea donde esperará más datos; cuando hayamos acabado, dejamos la última línea en blanco (pulsando por última vez Intro) y R cerrará el vector
  - ○ La función scan también se puede usar para copiar en un vector el contenido de un fichero de texto situado en el directorio de trabajo, o del que conozcamos su dirección en Internet

  • rep($a, b$): para definir un vector constante que contiene el valor $a$ repetido $b$ veces
  - ∗ $a$: número o vector
  - ∗ $b$: número o vector
  - ∗ times: repetimos el vector en bloque las veces que introduzcamos
  - ∗ each: repetimos cada valor las veces que introduzcamos

  • seq($a, b$, by= $p$): para generar una progresión aritmética de paso $p$ que empieza en $a$ hasta llegar a $b$
  - ∗ seq($a, b$, length.out=$n$): define progresión aritmética de longitud $n$ que va de $a$ a $b$ con paso $p$. Por tanto $p = (b - a)/(n - 1)$

    - seq($a$,by=$p$, length.out=$n$): define la progresión aritmética de longitud $n$ y paso $p$ que empieza en $a$
    - La instrucción $a : b$ define la secuencia de números enteros consecutivos entre dos números $a$ y $b$
- fix($x$): para modificar fácilmente el vector $x$
    - R abre una ventana de edición. Hasta que no la cerremos no volveremos a nuestra sesión de R
- sapply(*vector*,FUN=*función*): para aplicar funciones a todos los elementos de un vector
- sqrt($x$): calcula un nuevo vector con las raíces cuadradas de cada uno de los elementos del vector $x$
- length($x$): calcula la longitud del vector $x$
- max($x$): calcula el máximo del vector $x$
- min($x$): calcula el mínimo del vector $x$
- sum($x$): calcula la suma de las entradas del vector $x$
    - útil para evaluar sumatorios
- prod($x$): calcula el producto de las entradas del vector $x$
- mean($x$): calcula la media aritmética de las entradas del vector $x$
- diff($x$): calcula el vector formado por las diferencias sucesivas entre entradas del vector original $x$
- cumsum($x$): calcula el vector formado por las sumas acumuladas de las entradas del vector original $x$.
    - Cada entrada de cumsum($x$) es la suma de las entradas de $x$ hasta su posición
    - Permite definir sucesiones descritas mediante sumatorios
    - No admite el parámetro na.rm=TRUE
- sort($x$): ordena el vector en orden natural de los objetos que lo forman: el orden numérico creciente, orden alfabético....
    - dec: parámetro que si lo igualamos a TRUE estamos pidiendo que ordene el vector original $x$ en orden decreciente
- rev($x$): invierte el orden de los elementos del vector $x$
- *vector*[$i$]: da la $i$-ésima entrada del *vector*
    - Los índices en R empiezan en 1
    - *vector*[length(*vector*)]: nos da la última entrada del *vector*
    - *vector*[length(*vector*)$-i$]: nos da la $(i+1)$-ésima entrada del *vector*
    - *vector*[$-i$]: si $i$ es un número, este subvector está formado por todas las entradas del vector original, *vector*, menos la entrada $i$-ésima del original
    - Si $i$ en vez de un número resulta ser un vector, entonces es un vector de índices y crea un nuevo vector, *vector*[$i$], con las entradas del vector original, *vector*, cuyos índices pertenecen a $i$. Es una forma de definir un subvector de *vector*
    - *vector*[$-i$]: si $i$ es un vector (de índices), entonces este es el complementario de *vector*[$i$]. Sus entradas son las del *vector* original cuyos índices no pertenecen a $i$

- *vector*[$a$ : $b$]: define un subvector de *vector* cuyas entradas van de la $a$-ésima a la $b$-ésima del vector original

✎ Para crear subvectores también podemos utilizar operadores lógicos:

  - ◇ ==: $=$
  - ◇ !=: $\neq$
  - ◇ <:<
  - ◇ >:>
  - ◇ <=: $\leq$
  - ◇ >=: $\geq$
  - ◇ !: NO lógico
  - ◇ &: Y lógico
  - ◇ |: O lógico
  - ○ Si un vector no contiene ningún elemento que satisfaga la condición que imponemos, obtenemos como respuesta un vector vacío. Naturalmente, tiene longitud 0
  - ○ Los operadores lógicos también se pueden usar para pedir si una condición sobre dos números concretos se satisface o no

- which($x$ cumple *condición*): para obtener los índices de las entradas del vector $x$ que satisfacen la *condición* dada
- which.min($x$): nos da la primera posición en la que el vector $x$ toma su valor mínimo
  - ∗ which($x$==min($x$)): da todas las posiciones en las que el vector $x$ toma sus valores mínimos
- which.max($x$): nos da la primera posición en la que el vector $x$ toma su valor máximo
  - ∗ which($x$==max($x$)): da todas las posiciones en las que el vector $x$ toma sus valores máximos
- $x$[$i$]=$a$: podemos modificar entradas de un vector simplemente declarando sus nuevos valores. Se puede hacer entrada a entrada o para todo un subvector de golpe
  - ○ Puede ocurrir que añadamos un valor $n$ posiciones más allá de la última entrada. Entonces, las entradas sin valor, R las iguala a NA (Not Avilable), indicando que esas entradas no existen
- na.rm=TRUE: es un parámetro que tienen muchas de las funciones para vectores. Si la igualamos a TRUE hace que la función solo tenga en cuenta las entradas definidas
  - ○ Este parámetro es interesante puesto que una función aplicada a un vector que contenga algún NA da NA
- is.na($x$): sirve para extraer las entradas no definidas de un vector $x$
  - ○ Por consiguiente which(is.na($x$)) nos da los índices de las entradas NA del vector $x$
  - ○ $x$[!is.na($x$)]: creará el subvector de $x$ sin entradas NA
- na.omit($x$): borra las entradas no definidas de $x$
  - ○ El resultado de na.omit($x$) contiene primero un vector formado por las entradas del vector original que no son NA, luego una serie de atributos: los índices de las entradas que ha eliminado y el tipo de acción que ha llevado a cabo

- attr(*objeto, atributo*)=NULL: elimina los atributos indicados del objeto indicado
- factor: es como un vector, pero con una estructura interna más rica que permite usarlo para clasificar observaciones
  - levels: atributo del factor. Cada elemento del factor es igual a un nivel. Los niveles clasifican las entradas del factor. Se ordenan por orden alfabético
  - Para definir un factor, primero hemos de definir un vector y trasformarlo por medio de una de las funciones factor o as.factor.
- as.factor($x$): convierte el vector $x$ en un factor y toma como sus niveles los diferentes valores que aparecen en el vector
- factor(*vector*,levels=...): define un factor a partir del vector y dispone de algunos parámetros que permiten modificar el factor que se crea:
  - levels: permite especificar los niveles e incluso añadir niveles que no aparecen en el vector
  - labels: permite cambiar los nombres de los niveles
- levels(*factor*): para obtener los niveles del *factor*
  - Esta función también nos permite cambiar los nombres de los niveles de un factor
  - Esta función también nos permite agrupar algunos niveles de un factor en uno solo
- ordered(*vector*,levels=...): función que define un factor ordenado y tiene los mismos parámetros que factor
- list(...): función que crea una list (lista formada por objetos que pueden ser de clases diferentes)
  - La sintaxis de la función list es entrar como argumento los diferentes objetos que van a formar la list, poniendo a cada uno un nombre adecuado. Dicho nombre es interno
  - Para obtener una componente concreta usamos la instrucción *lista*\$*nombre*
  - También podemos indicar el objeto por su posición usando dobles corchetes *lista*[[$i$]]. Lo que obtendremos es un list formada por esa única componente, no el objeto que forma la componente
- str(*list*): para conocer la estructura interna de una list
- names(*list*): para saber los nombres de la list

- **Tema 4. Matrices**
  - matrix(*vector*, nrow=$n$, byrow=*valor lógico*): para definir una matriz de $n$ filas formada por las entradas del vector
    - nrow: número de filas
    - byrow: si se iguala a TRUE, la matriz se construye por filas; si se iguala a FALSE, se construye por columnas. Valor por defecto, FALSE
    - ncol: en vez de nrow podemos usar este parámetro (que indica el número de columnas)
    - R muestra las matrices indicando como [$i$,] la fila $i$-ésima y [,$j$] la columna $j$-ésima
    - Para construir una matriz de $n$ filas o $n$ columnas, es necesario que la longitud del vector al que se aplica la función matrix sea múltiplo de $n$. Si no es así, R rellena la última fila o columna con entradas del principio del vector y emite un mensaje de advertencia
    - En particular se puede definir una matriz constante aplicando la función matrix a un número. En este caso se han de usar los parámetros nrow y ncol para especificar el orden de la matriz
    - Todas las entradas de una matriz han de ser del mismo tipo de datos
  - rbind(*vector1*, *vector2*, ...): construye la matriz de filas *vector1*, *vector2*,...
    - Los vectores han de tener la misma longitud
    - También sirve para añadir filas a una matriz o concatenar por filas matrices con el mismo número de columnas
  - cbind(*vector1*, *vector2*, ...): construye la matriz de columnas *vector1*, *vector2*,...
    - Los vectores han de tener la misma longitud
    - También sirve para añadir columnas a una matriz o concatenar por columnas matrices con el mismo número de filas
  - *matriz*[$i, j$]: indica la entrada $(i, j)$ de la *matriz*
    - Esta construcción sirve también para definir submatrices, si $i$ y $j$ son vectores de índices, estaremos definiendo la submatriz con las filas pertenecientes al vector $i$ y las columnas pertenecientes al vector $j$
  - *matriz*[$i$,]: indica la fila $i$-ésima de la *matriz*
    - El resultado es un vector
    - Si lo que queremos es una matriz fila, dentro de los corchetes añadimos drop=FALSE
    - Esta construcción sirve para definir submatrices, si $i$ es un vector de índices, estaremos definiendo la submatriz con las filas pertenecientes al vector $i$
  - *matriz*[,$j$]: indica la columna $j$-ésima de la *matriz*
    - El resultado es un vector
    - Si lo que queremos es una matriz columna, dentro de los corchetes añadimos drop=FALSE
    - Esta construcción sirve para definir submatrices, si $j$ es un vector de índices, estaremos definiendo la submatriz con las columnas pertenecientes al vector $j$
  - diag(*matriz*): para obtener la diagonal de la matriz
  - nrow(*matriz*): nos devuelve el número de filas de la *matriz*

- ncol(*matriz*): nos devuelve el número de columnas de la *matriz*
- dim(*matriz*): nos devuelve las dimensiones de la *matriz*
- sum(*matriz*): obtenemos la suma de todas las entradas de la *matriz*
- prod(*matriz*): obtenemos el producto de todas las entradas de la *matriz*
- mean(*matriz*): obtenemos la media aritmética de todas las entradas de la *matriz*
- colSums(*matriz*): obtenemos las sumas por columnas de la *matriz*
- rowSums(*matriz*): obtenemos las sumas por filas de la *matriz*
- colMeans(*matriz*): obtenemos las medias aritméticas por columnas de la *matriz*
- rowMeans(*matriz*): obtenemos las medias aritméticas por filas de la *matriz*
- apply(*matriz*, MARGIN=..., FUN=*función*): para aplicar otras funciones a las filas o las columnas de una matriz
  - MARGIN: ha de ser 1 si queremos aplicar la función por filas; 2 si queremos aplicarla por columnas, o c(1,2) si la queremos aplicar a cada entrada
- t(*matriz*): para obtener la transpuesta de la *matriz*
- +: para sumar matrices
- *: para el producto de un escalar por una matriz
- %*%: para multiplicar matrices
  - Si multiplicamos matrices con el símbolo *, obtendremos una matriz que tiene en cada entrada $(i, j)$ la multiplicación de las entradas $(i, j)$ de cada una de las dos matrices
  - Lo mismo ocurre si aplicamos $matriz^n$: el resultado es una matriz con todas las entradas de la *matriz* elevadas a $n$
- mtx.exp(*matriz*,$n$): para elevar la *matriz* a $n$
  - Del paquete Biodem
  - No calcula las potencias exactas, las aproxima
- %^%: para elevar matrices
  - Del paquete expm
  - No calcula las potencias exactas, las aproxima
- det(*matriz*): para calcular el determinante de la *matriz*
  - Para calcular el determinante de una matriz de complejos podemos hacerlo mediante prod(eigen(*matriz*)$values) ya que es el producto de los valores propios contados con su multiplicidad
- qr(*matriz*)$rank: para calcular el rango de la *matriz*
- solve(*matriz*): para calcular la inversa de una matriz invertible
  - También sirve para resolver sistemas de ecuaciones lineales. Para ello introducimos solve(*matriz*,$b$), donde $b$ es el vector de términos independientes
- eigen(*matriz*): para calcular los valores y vectores propios
  - eigen(*matriz*)$values: nos da el vector con los valores propios de la *matriz*
  - eigen(*matriz*)$vectors: nos da una matriz cuyas columnas son los vectores propios de la *matriz*
  - Da los valores propios en orden decreciente de su valor absoluto y repetidos tantas veces como su multiplicidad

- Si hay algún valor propio con multiplicidad mayor que 1, da tantos valores de este valor propio como su multiplicidad. Además, en este caso procura que estos vectores propios sean linealmente independientes. Por tanto, cuando da vectores propios repetidos de algún valor propio es porque para este valor propio no existen tantos vectores propios linealmente independientes como su multiplicidad y, por consiguiente, la matriz no es diagonalizable
- Del resultado de eigen(*matriz*) se puede obtener una descomposición canónica *matriz*$= PDP^{-1}$ de una matriz diagonalizable *matriz*: basta tomar como $D$ la matriz diagonal que tiene como diagonal principal el vector eigen(*matriz*)$values y como P la matriz eigen(*matriz*)$vectors

• diag(*vector*): para construir una matriz diagonal con un vector dado.
- Si aplicamos diag a un número $n$, produce una matriz identidad de orden $n$

## ▪ Tema 5. Gráficos I

• plot(*vector,vector*): para dibujar su gráfico básico de puntos
- Si aplicamos plot a un solo vector $x = (x_1, ..., x_n)$, R produce el gráfico de los puntos $(1, x_1), ..., (n, x_n)$. Es decir, plot$(x)$ es una abreviatura de plot$(1 : n, x)$
- plot también sirve para representar el gráfico de una función definida mediante function: plot(*función*)

∗ log: para indicar si queremos un eje en escala logarítmica

∗ main="*título*": para poner título al gráfico

∗ xlab="*etiqueta*": para poner una etiqueta al eje de coordenadas $x$

∗ ylab="*etiqueta*": para poner una etiqueta al eje de coordenadas $y$
- Los valores de estos parámetros se tienen que entrar entre comillas o, si son fórmulas matemáticas, aplicarles la función expression() para que aparezcan en un formato matemático más adecuado

∗ pch=*número natural*: para especificar el símbolo. Puede tomar como valor cualquier número natural $n \in [0, 25]$
- Por defecto plot dibuja círculos vacíos (pch = 1)

∗ cex = *factor de escalado*: para modificar el tamaño de los símbolos

∗ col = *nombre del color en inglés*: para especificar el color de los símbolos o las líneas
- La paleta de R consta de 502 colores
- Los símbolos del 20 al 25 admiten un color para la línea (se especifica con el parámetro col) y uno para el relleno, el cual se especifica con el parámetro bg (el cual funciona del mismo modo que col)

∗ type: para indicar el tipo de gráfico que queremos producir. El valor del parámetro se tiene que entrar entre comillas y puede ser:

✓ p: para dibujar los puntos como simples puntos (es el valor por defecto)

✓ l: para dibujar los puntos unidos por líneas rectas sin que se vean los puntos
- La función plot aplicada a una función $f$ en realidad produce el gráfico de tipo "l" de una familia de puntos $(x, f(x))$. Por defecto, 101 puntos distribuidos

uniformemente a lo largo del dominio (valor que puede ser modificado con el parámetro n)
    - ✓ b: para dibujar puntos unidos por líneas rectas de manera que se vean los puntos, pero sin que las rectas entren dentro de los símbolos que representan los puntos
    - ✓ o: como b pero ahora sí que las rectas entran dentro de los símbolos que representan los puntos
    - ✓ h: para dibujar líneas verticales desde el eje de abscisas a cada punto (un histograma de líneas)
    - ✓ s: para dibujar un histograma de escalones
    - ✓ n: para no dibujar los puntos, solo el exterior del gráfico
  - ∗ lty: para especificar el tipo de línea
    - ✓ "solid" (ó 1): produce una línea continua. Es el valor por defecto
    - ✓ "dashed" (ó 2): produce una línea discontinua
    - ✓ "dotted" (ó 3): produce una línea de puntos
    - ✓ "dotdashed" (ó 4): produce una línea que alterna puntos y rallas
  - ∗ lwd=*factor de escalado*: para especificar el grosor de las líneas
  - ∗ xlim=*vector de extremos del rango*: para modificar el rango del eje $x$
  - ∗ ylim=*vector de extremos del rango*: para modificar el rango del eje $y$
    - Si el argumento de plot son dos vectores, por defecto los rangos de los ejes de coordenadas van del mínimo al máximo de los vectores correspondientes
    - Si el argumento de plot es una función $f$, por defecto el rango del eje de abscisas es el intervalo $[0, 1]$ y el rango del eje de ordenadas va del valor mínimo al máximo de $f$ sobre el rango de las abscisas.
  - ∗ xaxp: para modificar las posiciones de las marcas en el eje de abscisas
  - ∗ yaxp: para modificar las posiciones de las marcas en el eje de ordenadas
    - Mediante la expresión xaxp=c($n1, n2, m$), imponemos que R dibuje $m + 1$ marcas igualmente espaciadas entre los puntos $n1$ y $n2$ del eje de abscisas. La sintaxis para yaxp es la misma
    - Estas instrucciones no definen los rangos de los ejes de coordenadas, que se han de especificar con xlim e ylim si se quieren modificar
- points$(x, y)$: añade un punto de coordenadas $(x, y)$ a un gráfico ya existente
  - Podemos declarar el color, el símbolo etc., de este punto mediante los parámetros usuales
  - También sirve para añadir una familia de puntos. En este caso, hay que entrar en el argumento de points$(x, y)$ la lista $x$ de sus primeras coordenadas y la lista $y$ de sus segundas coordenadas
- abline: para añadir una recta a un gráfico ya existente
  - ✓ abline$(a, b)$: añade la recta $y = ax + b$
  - ✓ abline$(v = x_0)$: añade la recta vertical $x = x_0$
  - ✓ abline$(h = y_0)$: añade la recta horizontal $y = y_0$
  - Podemos especificar las características de estas rectas, como su grosor, su estilo o su color, mediante los parámetros pertinentes

- Los parámetros $v$ y $h$ de abline se pueden igualar a vectores numéricos, en cuyo caso la instrucción añade en un solo paso todas las rectas verticales u horizontales correspondientes, todas del mismo estilo

- text($x, y$, labels=...): añade en el punto de coordenadas $(x, y)$ el texto especificado como argumento de labels.
  - El texto se puede entrar entre comillas o en una expression
  - pos: permite indicar la posición del texto alrededor de las coordenadas $(x, y)$. Admite los siguientes valores:
    - ✓ 1: abajo
    - ✓ 2: izquierda
    - ✓ 3: arriba
    - ✓ 4: derecha
    - ✓ sin especificar: el texto se sitúa centrado en el punto $(x, y)$
  - La función text se puede usar para añadir varios textos en un solo paso. En este caso, $x$ e $y$ han de ser los vectores de abscisas y ordenadas de los puntos donde se añadirán los textos, labels el vector de textos, y pos el vector de sus posiciones; en este último vector, los textos que queremos centrados en su posición se han de indicar con NULL
- lines(*vector numérico*,*vector numérico*): añade a un gráfico existente una línea poligonal que une los puntos $(x_i, y_i)$ sucesivos
  - La apariencia de las líneas la podemos modificar con los parámetros usuales de grosor, color, estilo, etc.
- curve: permite añadir la gráfica de una curva a un gráfico existente
  - add=TRUE. Si no, la curva no se añade
  - La curva se puede especificar mediante una expresión algebraica con variable $x$, o mediante su nombre si la hemos definido antes
  - La función curve también se puede usar para producir la gráfica de una función, como plot, con la ventaja sobre esta última que no sólo se puede aplicar a una función definida con function, sino también a una expresión algebraica. Además, la función curve admite todos los parámetros de plot
- legend(*posición*, legend=c(*vector de nombres de las curvas*): para añadir una leyenda
  - posición: indica donde queremos situar la leyenda, y puede ser o bien dos números para especificar las coordenadas de su esquina superior izquierda, o bien una de las palabras siguientes:
    - ✓ 'bottomright"
    - ✓ "bottom"
    - ✓ "bottomleft"
    - ✓ "left"
    - ✓ "topleft"
    - ✓ "top"
    - ✓ "topright"
    - ✓ "right"

✓ "center"

* legend: vector que contiene los nombres (entre comillas o dentro de expression) con los que queremos identificar las curvas dentro de la leyenda

- segments: para añadir segmentos a un gráfico existente
- arrows: para añadir flechas a un gráfico existente
- symbols: para añadir símbolos a un gráfico existente
- polygon: para añadir polígonos cerrados especificando sus vértices a un gráfico existente

## Tema 6. Data frames

Un data frame es una tabla de doble entrada, formada por variables en las columnas y observaciones de estas variables en las filas, de manera que cada fila contiene los valores de las variables para un mismo caso o un mismo individuo.

- data(): para abrir una ventana con la lista de los objetos de datos a los que tenemos acceso en la sesión actual de R (los que lleva la instalación básica de R y los que aportan los paquetes que tengamos cargados)
  - Estos objetos no aparecen cuando hacemos ls() ni se borran con rm(list=ls())
  - Si entramos data(package=.packages(all.available=TRUE)) obtendremos la lista de todos los objetos de datos a los que tenemos acceso, incluyendo los de los paquetes que tengamos instalados, pero que no estén cargados en la sesión actual
- head(*data frame*,$n$): para mostrar las $n$ primeras filas del data frame. Por defecto se muestran las 6 primeras filas
- tail(*data frame*,$n$): para mostrar las $n$ últimas filas del data frame. Por defecto se muestran las 6 últimas
- str(*data frame*): para conocer la estructura global de un data frame
- names(*data frame*): para producir un vector con los nombres de las columnas
- rownames(*data frame*): para producir un vector con los identificadores de las filas
  - R entiende siempre que estos identificadores son palabras, aunque sean números, de ahí que los imprima entre comillas
- dimnames(*data frame*): para producir una list formada por dos vectores (el de los identificadores de las filas y el de los nombres de las columnas)
- dim(*data frame*): para producir un vector con el número de filas y el de columnas
- *data frame*\$*nombre de la variable*: para obtener una columna concreta de un data frame
  - El resultado será un vector o un factor, según cómo esté definida la columna dentro del data frame
  - Las variables de un data frame son internas, no están definidas en el entorno global de trabajo de R
- *data frame*$[n, m]$: para extraer "trozos" del data frame por filas y columnas (depende de como definamos $n$ y $m$. Funciona del mismo modo que para matrices)
  - $n$ y $m$ pueden definirse como:

    - ✓ intervalos. Por ejemplo *data frame*[1:3,5:8]
    - ✓ condiciones. Por ejemplo *data frame*[d.f$Species==“dog” & d.f$Age >7,]
    - ✓ números. Por ejemplo *data frame*[1,5] (que equivale a pedir la entrada (1,5))
    - ✓ no poner nada. Por ejemplo, *data frame*[,5] (donde estamos pidiendo la quinta columna con todas sus filas)
    - Si sólo queremos definir la subtabla quedándonos con algunas variables, basta aplicar el nombre del data frame al vector de variables (sin necesidad de dejar el espacio en blanco seguido de una coma que serviría para indicar que nos referimos a columnas y no a filas)
    - Estas construcciones se pueden usar también para reordenar las filas o columnas
- read.table(): para definir un data frame a partir de una tabla de datos contenida en fichero
    - Este fichero puede estar en nuestro ordenador, y en este caso lo más práctico es que la tengamos en el directorio de trabajo de R, o bien podemos conocer su url. Sea cual sea el caso, se aplica la función al nombre del fichero o a la dirección entre comillas
    - sep: para especificar las separaciones entre columnas en el fichero (si son distintas a espacios en blanco). Si es así, hay que introducir el parámetro pertinente entre comillas
    - header=TRUE: para indicar si la tabla que importamos tiene una primera fila con los nombres de las columnas
        - En caso contrario, o bien igualar header=FALSE, o directamente no especificar el parámetro, ya que ese es su valor por defecto
    - dec: para especificar el signo que separa la parte entera de la decimal en los números, si no es un punto. Si es así, hay que introducir el parámetro pertinente entre comillas
    - stringsAsFactors: para prohibir la transformación de las columnas de palabras en factores debemos usar stringsAsFactors=FALSE (ya que por defecto, R realiza dicha transformación)
    - Para importar un fichero de una página web segura (cuyo url empiece con https), no podemos entrar directamente la dirección en read.table; una solución es instalar y cargar el paquete RCurl y entonces usar la instrucción read.table (textConnection(getURL(“url ”)),...).
- read.csv: para importar ficheros en formato CSV
- read.xls ó read.xlsx: para importar hojas de cálculo tipo Excel u OpenOffice en formato XLS o XLSX, respectivamente. Son del paquete xlsx
- read.mtb: para importar tablas de datos de Minitab (del paquete foreign)
- read.spss: para importar tablas de datos SPSS (del paquete foreign)
- write.table(*data frame*, file=“*nombre del fichero*”): para exportar un data frame a un fichero
    - Podemos usar el parámetro sep para indicar el símbolo de separación de columnas en el fichero que creemos y el parámetro dec para indicar la separación entre parte entera y parte decimal en los números

- data.frame(*vector1, ..., vectorn*): para construir un data frame a partir de unos vectores introducidos en el orden en el que queremos disponer las columnas de la tabla.
  - R considera del mismo tipo de datos todas las entradas de una columna de un data frame
  - Las variables tomarán los nombres de los vectores. Estos nombres también se pueden especificar en el argumento de data.frame, entrando una construcción de la forma *nombre de la variable = vector con el contenido de la variable*
  - row.names: para especificar los identificadores de las filas
    - Si hemos asignado identificadores a las filas, podemos usarlos para extraer subtablas del data frame (aunque también podemos seguir usando los números de las filas). Eso sí, hay que recordar que si se usan los identificadores, como son palabras, hay que escribirlos entre comillas
  - stringsAsFactors: para prohibir la transformación de las columnas de palabras en factores debemos usar stringsAsFactors=FALSE (ya que por defecto, R realiza dicha transformación)
- fix(*data frame*): para crear/editar un data frame con el editor de datos
- names(*data frame*)=*vector de nombres de variables*: para cambiar los nombres de las variables
- rownames(*data frame*)=*vector de nombres de filas*: para modificar los identificadores de las filas
  - Como cada identificador ha de determinar el individuo al cual corresponde la fila, conviene que estos identificadores sean todos diferentes
- paste(): para pegar vectores, entrada a entrada, usando como separador el que especificamos con sep (el valor por defecto es el espacio en blanco)
  - Si en lugar de pegar un vector pegamos un elemento, éste se entiende como un vector constante formado por el número adecuado de copias
- dimnames(*data frame*)=list(*vector con los nombres de las filas, vector con los nombres de las columnas*): para modificar los nombres de las filas y de las columnas simultáneamente
- as.character: para transformar todos los datos de un objeto en palabras
- as.integer: para transformar todos los datos de un objeto en números enteros
- as.numeric: para transformar todos los datos de un objeto en números reales
- *data frame*[*número de fila*, ]=c(...): para añadir una fila a un data frame
  - Acción nada recomendable, ya que puede provocar resultados no deseados:
    - ✓ Las filas que añadimos de esta manera son vectores, y por lo tanto sus entradas han de ser todas del mismo tipo. Por consiguiente, esta opción sólo se puede usar en data frames cuyas variables contengan todas el mismo tipo de datos
    - ✓ Si no añadimos las filas inmediatamente siguientes a la última del data frame, los valores entre su última fila y las que añadimos quedarán no definidos, y aparecerán como NA; esto puede ser un problema a la hora de aplicar funciones al data frame y además estropea los factores

    - La mejor manera de añadir filas a un data frame es organiza?ndolas en un nuevo data frame con los mismos nombres de las variables, y a continuación concatenarlas al data frame usando la función rbind
- *data frame*$*nueva variable*: para añadir una nueva variable al data frame
    - También podemos concatenar columnas a un data frame usando la función cbind; en este caso, se puede añadir directamente la columna, sin necesidad de convertirla previamente en un data frame. La variable añadida ha de tener la misma longitud que las variables del data frame original; en caso contrario, se añadirán valores NA a las variables del data frame original o a la variable que añadimos hasta completar la misma longitud
    - Sobra decir que podemos usar también el editor
- droplevels(*data frame*): para borrar los niveles sobrantes de todos los factores, ya que las columnas que son factores heredan en los subdata frames todos los niveles del factor original, aunque no aparezcan en el trozo que hemos extraído
- select(*data frame, parámetros*): para especificar las variables que queremos extraer de un data frame
    - Del paquete dplyr
    * starts_with("*x*"): extrae del data frame las variables cuyo nombre empieza con la palabra *x*
    * ends_with("*x*")) extrae del data frame las variables cuyo nombre termina con la palabra *x*
    * contains("*x*")) extrae del data frame las variables cuyo nombre contiene en algún sitio la palabra *x*
- subset(*data frame, condición*, select=*columnas*): para extraer del data frame las filas que cumplen la condición y las columnas especificadas
    - Si queremos todas las filas, no hay que especificar ninguna condición
    - Si queremos todas las columnas, no hace falta especificar el parámetro select
    - Las variables en la condición se especifican con su nombre, sin añadir antes el nombre del data frame
- sapply(*data frame, función*): para aplicar una función a todas las columnas de un data frame en un solo paso
    * na.rm=TRUE: para evitar que el valor que devuelva la función para las columnas que contengan algún NA sea NA
- aggregate(*variable(s)~factor(es)*, data=*data frame*, FUN=*función*): para aplicar una función a variables de un data frame clasificadas por los niveles de un, o más de un, factor
    - Si queremos aplicar la función a más de una variable, tenemos que agruparlas a la izquierda de la tilde con cbind
    - Si queremos separar las variables mediante más de un factor, tenemos que agruparlos a la derecha de la tilde con signos +: *factor1+factor2+...*
- attach(*data frame*): para hacer que R entienda sus variables como globales y que las podamos usar por su nombre, sin necesidad de añadir delante el nombre del data frame y el símbolo $

    - Si ya hubiera existido una variable definida con el mismo nombre que una variable del data frame al que aplicamos attach, hubiéramos obtenido un mensaje de error al ejecutar esta función, y no se hubiera reescrito la variable global original
- detach(*data frame*): para devolver la situación original, eliminando del entorno global las variables del data frame

- **Tema 8. Datos cualitativos**
  - table(*vector ó factor*): muestra la tabla de frecuencias absolutas
    ∗table(*x,y*) siendo el vector $x$ el de las filas e $y$ el de las columnas: permite construir tablas de frecuencias conjuntas de dos o más variables
    ∗Si le aplicamos la función table a un data frame de variables cualitativas, obtendremos su tabla de frecuencias absolutas, con las variables ordenadas tal y como aparecen en el data frame
  - names(table(*x*)): muestra los nombres de las columnas de un table
    ∗Para mostrar los nombres de un vector que tienen frecuencia 0, hay que transformar el vector en un factor con los niveles deseados
  - table(*x*)[*índice ó nombre*]: comando para referirnos a una entrada del table
    ∗table(*x*)[*índice ó nombre, índice o nombre*]: para referirnos a una entrada de una table bidimensional
  - names(which(table(*x*)==*n*)): muestra los niveles que tienen frecuencia absoluta $n$ en $x$
  - names(which(table(*x*)==max(table(*x*)))): nos da la moda de $x$
  - prop.table(table(*x*)): muestra la tabla de frecuencias relativas
    ∗También podemos calcular la tabla de frecuencias relativas de un vector de la forma: table(*x*)/lenght(*x*)
    ∗prop.table(table(*x,y*)): calcula la tabla de frecuencias relativas globales
    ∗prop.table(table(*x,y*), margin=*número*): con el parámetro margin especificamos la variable que define las subpoblaciones
  - t(table(*x,y*)): para trasponer la table bidimensional sin tener que recalcularla
  - aperm(*table*, perm=...): permite cambiar el orden de las variables en una tabla multidimensional, igualando el parámetro perm a la lista de las variables en el orden deseado
  - CrossTable(*x,y*, prop.chisq=FALSE) (del paquete gmodels): permite producir un resumen de la tabla de frecuencias absolutas y las tres tablas de frecuencias relativas
  - ftable(*x,y,z*): muestra el table (en este caso tridimensional) en formato plano
    ∗ftable(*x,y,z*, row.vars ó col.vars=c(*x,y*)): permite especificar qué variables queremos que aparezcan como filas o como columnas
    ∗También lo podemos usar con data frames.
  - sum(*table*): nos permite obtener el número total de individuos de la muestra
  - summary(*data frame*): obtenemos una tabla con las frecuencias absolutas de cada variable del data frame de variables cualitativas. (Â¡Â¡Esta tabla solo sirve para ver la información!!)
  - apply(*data frame*, MARGIN=2, FUN=table): para calcular en un solo paso la table de cada variable

- barplot(table($x$), col=*color/es*,main=*título*)
  ∗La función \n produce un cambio de línea si se encuentra dentro de una frase entre comillas
  ∗La función \" escribe unas comillas en el texto
  ∗Por defecto, un gráfico de barras de un table bidimensional es un diagrama de barras apiladas
- barplot(table$x,y$, beside=TRUE): organiza las barras una junto a la otra obteniendo un diagrama de barras por bloques
- barplot(table($x$, $y$), legend.text=TRUE): permite poner una leyenda
  ∗La leyenda se puede modificar con el parámetro args.legend igualado a una list con los parámetros que usaríamos en la función legend
- barplot(table$x,y$, names=*vector*): permite cambiar los nombres de los niveles que muestra debajo del eje horizontal
- barplot(table$x$, horiz=TRUE): permite poner las barras en horizontal
- pie(table($x$)): produce un diagrama circular
- plot ó mosaicplot(table($x,y,z$)):produce un gráfico de mosaico
- cotabplot(table($x,y,z$))(del paquete vcd): produce un diagrama de mosaico para cada nivel de la tercera variable
- mosaic3d(table($x,y$)) (del paquete vcdExtra): produce un mosaico tridimensional

▪ **Tema 9. Datos ordinales**

- cumsum(table($x$)): calcula la tabla de frecuencias absolutas acumuladas
- cumsum(prop.table(table($x$))): calcula la tabla de frecuencias relativas acumuladas
  ∗También se puede calcular de la forma: cumsum(table($x$))/length($x$) ó cumsum(table($x$)/length($x$))
- barplot(cumsum(table($x$))): para crear el gráfico de barras de las frecuencias absolutas acumuladas
  ∗barplot(cumsum(prop.table(table($x$)))): para crear el gráfico de barras de las frecuencias absolutas acumuladas
- apply(table($x$), MARGIN=... , FUN=cumsum): para calcular frecuencias acumuladas en una tabla multidimensional, donde el valor de MARGIN es 1 si queremos calcular las frecuencias acumuladas por filas o 2 si las queremos calcular por columnas
- cut(data frame,*variable*, breaks=*vector extremos intervalos*, right=FALSE, labels=*vector etiquetas*): para agrupar las entradas de una variable en niveles
  ∗El parámetro right=FALSE sirve para indicar que los puntos de corte pertenecen al intervalo a su derecha

▪ **Tema 10. Datos cuantitativos**

- as.numeric(names(which(table($x$)==max(table($x$))))): muestra la moda
- mean($x$): muestra la media aritmética
- median($x$): muestra la mediana
- length($x$): muestra el número de objetos de la muestra
- $x$=sort($x$): ordena de menor a mayor la muestra
- quantile($x$, p): calcula el cuantil de orden p del vector $x$

- range($x$): nos da los valores máximo y mínimo de $x$
- diff(range($x$)): calcula el rango de $x$
- IQR($x$): calcula el rango intercuantílico
  ∗Con el parámetro type se pueden especificar los cuantiles
- var($x$): calcula la varianza muestral
- sd($x$): calcula la desviación típica muestral
- var($x$)*(length($x$)-1)/length($x$): calcula la varianza "verdadera"
- sd($x$)*sqrt((length($x$)-1)/length($x$)): calcula la desviación típica "verdadera"
  ∗sqrt(var($x$)*(length($x$)-1)/length($x$)): otra forma de calcular la desviación típica "verdadera"
- summary(*vector*): muestra un resumen estadístico del vector
- summary(*data frame*): muestra un resumen estadístico de todas sus variables
- by(*columnas, factor*, FUN=summary): aplica la función summary a algunas columnas de un data frame segmentándolas según los niveles de un factor
- boxplot($x$): realiza un diagrama de caja del vector $x$
  ∗También se puede aplicar a un data frame
- boxplot(*variable numérica* ∼ *variable factor*, data=*data frame*): para dibujar en un único gráfico los diagramas de caja de una variable numérica de un data frame segmentada por un factor del data frame
  ∗El parámetro notch=TRUE añade una muesca en la medina de la caja. Si las de dos diagramas de caja no se solapan, se puede tomar como evidencia significativa de que las medianas de las poblaciones correspondientes son diferentes
- str(boxplot(*variable ó data frame*)): muestra la estructura interna del boxplot
  ∗stats: componente de la list que nos da, para cada diagrama del gráfico, los valores de sus 5 líneas horizontales
  ∗out: nos da los valores atípicos
  ∗group: si hay más de un diagrama de caja, nos da los diagramas a los cuales pertenecen los valores atípicos

▪ **Tema 11. Datos cuantitativos agrupados**

- Para establecer el número de $k$ clases en que vamos a dividir los datos, podemos usar:
  - $\lceil k = \sqrt{n} \rceil$: Regla de la raíz cuadrada, donde $n$ es el número total de muestras
  - nclass.Sturges($x$): Regla de Sturges
  - nclass.scott($x$): Regla de Scott
  - nclass.FD($x$): Regla de Freedman-Diaconis
- Para calcular la amplitud, $A$, dividiremos el rango de los datos entre el número k de clases y redondearemos por exceso a un valor de la precisión de la medida. En el caso improbable de que el cociente del rango entre el número de clases dé un valor exacto en la precisión de la medida, tomaremos como $A$ este cociente más una unidad de precisión
- $L_1 = min(x) - \frac{1}{2} \cdot precision$: para calcular el extremo izquierdo del primer intérvalo
- $L_i = L_1 + (i-1)A, \forall i = 2, ..., k+1$: para calcular los extremos izquierdos del resto de intérvalos
  ∗Los extremos nunca coincidirán con valores del conjunto

- $X_i = \frac{L_i + L_{i+1}}{2}$: para la marca de la clase
  $*X = (L[1 : (length(L) - 1)] + L[2 : length(L)])/2$: otra forma de calcular la marca de la clase (formando un vector), siendo $L$ el vector de los límites izquierdos de cada intérvalo
- cut($x$, breaks=..., labels=..., right=...): función básica de R para agrupar un vector de datos numéricos y codificar sus valores con las clases a las que pertenecen
  $*x$ es el vector
  *breaks puede ser o bien un vector con los extremos de los intervalos, o bien un número $k$ que indica las clases
  *labels es un vector con las etiquetas de los intervalos
  *right permite indicar qué tipo de intervalos queremos. right=FALSE es cerrado por la izquierda y abierto por la derecha
  *include.lowest=TRUE (combinado con right=FALSE) impone que el primer intervalo sea cerrado
- hist($x$, breaks=..., right=FALSE, plot=FALSE)$ count: para calcular frecuencias absolutas
  *Es conveniente igualar el parámetro breaks al vector de los extremos de los intervalos
  *plot=FALSE impide que se dibuje el histograma
  *hist tiene el parámetro mids el cual está formado por los puntos medios
- Tabla_frec_agrup=function($x$,$k$,$A$,$p$){Definir la función}: para calcular la tabla de frecuencias si vamos a tomar todas las clases de la misma amplitud. $x$ vector, $k$ clases, $A$ amplitud y $p$ precisión
- Tabla_frec_agrup_L=function($x$,$L$,$V$){Definir la función}: para calcular la tabla de frecuencias si conocemos los extremos de los intervalos. $x$ vector, $L$ vector de extremos de clases y $V$ valor lógico (= TRUE si queremos que el último intervalo sea cerrado)
- $M = L_c + A_c \cdot \frac{\frac{n}{2} - N_{c-1}}{n_c}$: aproximación a la mediana de los datos "reales". Siendo $L_c$ el límite izquierdo del intervalo crítico, $N_{c-1}$ la frecuencia absoluta acumulada del intervalo anterior al crítico, $n_c$ la frecuencia absoluta del intervalo crítico y $A_c$ su amplitud
- $Q_p = L_c + A_c \cdot \frac{pn - N_{c-1}}{n_c}$: para aproximar cuantiles de los datos "reales" a partir de los datos agrupados, donde ahora el intervalo crítico es el primero que tiene frecuencia relativa acumulada mayor o igual que $p$ y el resto de valores se definen relativos a este intervalo crítico
- hist_abs($x$,$L$): para hacer rápidamente un histograma de frecuencias absolutas
- rug($x$): para indicar las frecuencias absolutas de las barras en el histograma
  *rug(jitter($x$)): añade ruido a rug para distinguir mejor los empates
- hist_abs.cum($x$,$L$): para dibujar histogramas de frecuencias absolutas acumuladas
- plot(density($x$)): para estimar la densidad de la distribución
- hist_rel($x$,$L$): para dibujar un histograma de frecuencias relativas con curva de densidad
- hist_rel.cum($x$,$L$): para dibujar histograma de frecuencias relativas acumuladas y la función de distribución estimada

- dnorm(x, mu=media, sd=desviación típica): para añadir al histograma la gráfica de la función densidad de una distribución normal

## 2. R.II

### Tema 1. Conceptos básicos de muestreo estadístico

- sample($x$,$n$,replace=...): generador de muestras aleatorias de un vector
  - $x$: vector o un número natural que representa el vector 1,2,...$x$
  - $n$: tamaño de la muestra que deseamos
  - replace=TRUE: muestra con reposición, es decir, simple (m.a.s para abreviar); replace=FALSE (por defecto): muestra aleatoria sin reposición
- set.seed(*valor*): se iguala la semilla al valor que le entramos y permite fijar el resultado de una función aleatoria
  - También fija los resultados de todas las instrucciones siguientes que generen vectores aleatorios
- set.seed(NULL): para volver a reiniciar la semilla de la aleatoriedad tras haber usado set.seed()
- replicate($n$,*instrucción*): para tomar diversas muestras aleatorias de una misma población
  - $n$: número de repeticiones

### Tema 2. Intervalos de confianza

En primer lugar, instalamos y cargamos el paquete IntervalosMatesII.tar.gz

- ICZ.exact($x$,sigma,$n$,conf.level=...,na.rm=...): I.C para la media de una población normal con varianza poblacional conocida
  - $x$: puede ser o bien un vector numérico con la muestra aleatoria simple, o bien un número que representa su media muestral
  - sigma: desviación típica poblacional $\sigma$
  - $n$: tamño de la muestra. Si como primer parámetro $x$ hemos entrado un vector, no hace falta especificar este tamaño
  - conf.level: nivel de confianza $1-\alpha$ expresado en tanto por uno. Valor por defecto 0,95
  - na.rm: su valor por defecto es FALSE. Si como primer parámetro $x$ hemos entrado un vector, hay que especificar na.rm=TRUE
  - El resultado es un data frame con las columnas siguientes:
    - mean: media muestral
    - size: tamaño de la muestra
    - lower and upper: extremos inferior y superior del intervalo de confianza
    - conf.level: nivel de confianza
  - *variable*[1,c(3,4)]: si queremos los extremos del intervalo como data frame, siendo *variable* la que tiene guardada la función ICZ.exact

    - c(*variable*[1,3],*variable*[1,4]): si queremos solo el vector con los extremos del intervalo de confianza, siendo *variable* la que tiene guardada la función ICZ.exact
- NMin.mu($A$,sigma,conf.level=...): Cálculo del tamaño muestral para la media fijados la amplitud, la desviación típica poblacional y el nivel de confianza
    - conf.level: valor por defecto 0,95
- ICT.exact($x$,sdm,$n$,conf.level=...,na.rm=...): I.C para la media de una población normal con varianza poblacional desconocida
    - $x$: puede ser o bien un vector numérico con la muestra aleatoria simple, o bien un número que representa su media muestral
    - sdm: desviación típica muestral. Si como primer parámetro hemos entrado un vector, no hace falta especificar este parámetro
    - $n$: tamaño de la muestra. Si como primer parámetro $x$ hemos entrado un vector, no hace falta especificar este tamaño
    - conf.level y na.rm tienen el mismo significado y uso que en ICZ.exact
- ICZ.approx($x$,sdm,$n$,conf.level=...,na.rm=...): I.C para la media cuando la muestra es grande
- binom.exact($x$,$n$,conf.level): I.C para la proporción poblacional
    - Hay que cargar el paquete epitools para poder usar esta función
    - $x$: número de éxitos
    - $n$: tamaño de la muestra
    - conf.level=$1-\alpha$. Valor por defecto 0.95
    - Conviene redondear a 3 cifras decimales
- binom.wilson($x$,$n$,conf.level): Método de Wilson para calcular el intervalo de confianza para la proporción poblacional de una muestra grande ($\geq 40$)
    - Hay que cargar el paquete epitools para poder usar esta función
    - Conviene redondear a 3 cifras decimales
- binom.approx($x$,$n$,conf.level): I.C para la proporción cuando $n$ es muy grande ($n \geq 100$, $np \geq 10$, $nq \geq 10$)
    - Hay que cargar el paquete epitools para poder usar esta función
- ICZ.p($x$,$n$,conf.level=...): calcula exactamente la función anterior
    - $x$: puede ser un vector de 0 y 1, en cuyo caso no hay que especificar su tamaño $n$
    - Genera un mensaje de advertencia si las condiciones explicadas más arriba para poder aplicar la fórmula de Laplace no se satisfacen
- NMin.p($A$,conf.level=...): Cálculo del tamaño muestral para una proporción fijados la amplitud y el nivel de confianza
- IC.var($x$,$n$,conf.level=...,na.rm=...): I.C para la varianza de una población normal
    - $x$: puede ser o bien un vector numérico con la muestra aleatoria simple, o bien un número que representa su varianza muestral
    - $n$: tamño de la muestra. Si como primer parámetro $x$ hemos entrado un vector, no hace falta especificar este tamaño
    - conf.level y na.rm tienen el mismo significado y uso que en ICZ.exact

- **Tema 3. Contrastes de hipótesis**

  En primer lugar cargamos e instalamos los paquetes TeachingDemos y pwr.

  - z.test(x,mu=...,sd=...,alternative=...,conf.level=...)
    - x: vector de datos
    - mu=$\mu_0$
    - sd=$\sigma$
    - alternative: puede tomar 3 valores: "two.sided"(representa $H_1$: $\mu \neq \mu_0$)' less'($\mu \leq \mu_0$),"greater"($\mu \geq \mu_0$). Valor por defecto "two.sided"
    - conf.level=$1-\alpha$. Valor por defecto 0.95
  - t.test(x,y,mu=...,alternative=...,conf.level=...,paired=...,var.equal=...,na.omit=...)
    - x: vector de datos
    - y: vector opcional; si lo entramos, R entiende que estamos haciendo un contraste de las dos medias, con hipótesis nula la igualdad de estas medias
      - Podemos sustituir los vectores $x$ e $y$ por una fórmula $variable1$ $variable2$ que indique que separamos la variable numérica $variable1$ en dos vectores definidos por los niveles de un factor $variable2$ de dos niveles (o de otra variable asimilable a un factor de dos niveles, como por ejemplo una variable numérica que sólo tome dos valores diferentes)
    - mu=$\mu_0$. Solo lo tenemos que especificar si hemos entrado una sola muestra
    - alternative. Valor por defecto "two.sided"
      - Si es un contraste de una muestra, igual que en z.test
      - Si es un contraste de dos muestras, "two.sided"(representa $H_1$: $\mu_x \neq \mu_y$),'less'($\mu_x \leq \mu_y$),"greater"($\mu_x \geq \mu_y$)
    - conf.level=$1-\alpha$. Valor por defecto 0.95
    - paired: sólo lo tenemos que especificar si llevamos a cabo un contraste de dos medias. En este caso, si entramos paired=TRUE, estamos diciendo que las muestras son emparejadas, mientras que si entramos paired=FALSE, estamos diciendo que las muestras son independientes. Si se trata de muestras emparejadas, los vectores x e y tienen que tener la misma longitud, naturalmente. Valor por defecto, FALSE
    - var.equal: sólo lo tenemos que especificar si llevamos a cabo un contraste de dos medias de poblaciones independientes, y en este caso sirve para indicar si queremos considerar las dos varianzas iguales (igualándolo a TRUE) o diferentes (igualándolo a FALSE). Valor por defecto, FALSE
    - na.omit=TRUE: elimina las entradas NA de los vectores
  - sigma.test(x,sigma=...,alternative=...,conf.level=...)
    - x: vector de datos
    - sigma=$\sigma$
    - alternative y conf.level: igual que en casos anteriores
  - var.test(x,y,alternative=...,conf.level=...)
    - Nos da el I.C para el cociente de las varianzas $\frac{\sigma_x^2}{\sigma_y^2}$
  - binom.test(x,n,p=...,alternative=...,conf.level=...)

    - x: número de éxitos de la muestra
    - n: número de intentos de la muestra
    - p: probabilidad de éxito
- prop.test(x,n,p=...,alternative=...,conf.level=...)
    - x: número de éxitos de la muestra o vector de dos números (para realizar contrastes de dos muestras, donde estos números son los éxitos de las muestras)
    - n: número de intentos de la muestra o vector con dos entradas (las longitudes de ambas muestras)
    - p: proporción poblacional. En caso de contraste de dos muestras no hay que especificarlo
- fisher.test(x,alternative=...,conf.level=...)
    - x: matriz$\in M_2(\mathbb{R})$
- mcnemar.test(x)
    - x: matriz$\in M_2(\mathbb{R})$
- paquete pwr: Cálculo de la potencia de un contraste
    - pwr.t.test(n,d,sig.level=...,power=...,type=...,alternative=...): para utilizar en t-tests de una media, de dos medias emparejadas o de dos medias independientes usando muestras del mismo tamaño
    - pwr.t2n.test(n1,n2,d,sig.level=...,power=...,alternative=...): para utilizar en t-tests de dos medias independientes usando muestras de distinto tamaño
    - pwr.p.test(n,h,sig.level=...,power=...,alternative=...): para utilizar en contrastes binomiales de una proporción
    - pwr.2p.test(n,h,sig.level=...,power=...,alternative=...): para utilizar en contrastes binomiales de dos proporciones independientes usando que usen muestras del mismo tamaño
    - pwr.2p2n.test(n1,n2,h,sig.level=...,power=...,alternative=...): para utilizar en contrastes binomiales de dos proporciones independientes que usen muestras de distinto tamaño
    - Los parámetros son:
        - n: tamaño de la muestra
        - d o h: magnitud del efecto
        - sig.level: nivel de significación $\alpha$
        - power: potencia $1-\beta$
        - type: tipo de contraste ("one.sample","two.sample","paired")
- cohen.ES(test=..., size=...)$effect.size: para recordar los valores de magnitud del efecto

## Tema 4. Estadística multidimensional

- scale(X,center=...,scale=...): transformación lineal a una tabla de datos, en particular de centrarla o tipificarla
    - X: tanto matriz como data frame

    - center: para restar a las columnas, TRUE (o no, FALSE), el vector de las medias de $X$ .Valor por defecto, center=TRUE
    - scale: para dividir las columnas, TRUE (o no, FALSE), por el vector de desviaciones típicas muestrales
    - Para tipificar, no podemos hacer simplemente scale($X$), sino que hay que multiplicar el resultado por $\sqrt{\frac{n}{n-1}}$
- attr(..., "scaled:center")=NULL: para eliminar el atributo
- attr(..., "scaled:scale")=NULL: para eliminar el atributo
- cov($X, Y$): covarianza muestral
    - Si $X$ es un data frame, realizamos la matriz de covarianzas muestrales del data frame
    - use='complete.obs': para no tener en cuenta los valores NA
    - use="pairwise.complete.obs": para no tener en cuenta los valores NA. Se eliminan de cada una de ellas sus entradas NA y aquellas en cuya fila la otra tiene un NA
- *variable*=as.matrix(*data frame*): para pasar un data frame a matriz
- cor($X, Y$): correlación de Pearson de dos vectores
    - Si $X$ es un data frame, realizamos la matriz de covarianzas muestrales del data frame
    - use='complete.obs': para no tener en cuenta los valores NA
    - use="pairwise.complete.obs": para no tener en cuenta los valores NA. Se eliminan de cada una de ellas sus entradas NA y aquellas en cuya fila la otra tiene un NA
- cov2cor($X$): aplicada a la matriz de covarianzas (muestrales o no) da la matriz de correlaciones
- upper.tri($X$): produce matriz triangular superior de valores lógicos de una matriz cuadradda $X$
- lower.tri($X$): produce matriz triangular inferior de valores lógicos de una matriz cuadradda $X$
- $M[L]$: Si $M$ es una matriz y $L$ es una matriz de valores lógicos del mismo orden, produce el vector construido de la manera siguiente: de cada columna, se queda sólo con las entradas de $M$ cuya entrada correspondiente en $L$ es TRUE, y a continuación concatena estas columnas, de izquierda a derecha, en un vector
- scatterplot3d(): para hacer diagramas de dispersión tridimensionales
- plot(*data frame*, *vector columnas*): para obtener las matrices formadas por los diagramas de dispersión de todos sus pares de columnas
- pairs(*data frame*, *vector columnas*): produce exactamente lo mismo que la instrucción anterior. Ventaja: se puede aplicar a una matriz, plot no
- spm(*data frame*, *vector columnas*): permite dibujar matrices de diagramas de dispersión enriquecidos con información descriptiva extra de las variables de la tabla de datos
    - Hace falta descargar el paquete car
- hist2d(x, y, nbins=..., col=...): para dibujar histogramas bidimensionales
    - x, y: vectores

    - nbins: número de clases
    - col: colores
    - Hace falta el paquete gplots
  - brewer.pal(n,*paleta predefinida*): para elegir esquemas de colores bien disñados
    - n: número de colores
  - display.brewer.all(): muestra nombre y contenido de todas las paletas predefinidas
  - colorRampPalette(brewer.pal(...))($m$): produce una nueva paleta de $m$ colores a partir del resultado debrewer.pal, interpolando nuevos colores
  - rev(colorRampPalette(brewer.pal(...))($m$)): invierte el orden de los colores

- **Tema 5. Bondad de ajuste**
  - fitdistr($x$,densfun='distribución',start=...): para estimar un parámetro de una distribución a partir de una muestra y además obtener el error estándar de dicha estimación
    - Esta función es del paquete MASS
    - Esta función calcula los estimadores máximo verosímiles de los parámetros de la mayoría de las familias de distribuciones
    - $x$: muestra que usamos para estimar los parámetros
    - distribución: es el nombre de la familia de distribuciones; debe entrarse entre comillas
    - $m$: parámetro locacional $\mu$
    - $s$: parámetro locacional de escala $\sigma$
    - Si fitdistr no dispone de fórmulas cerradas para los estimadores máximo verosímiles de los parámetros, usa un algoritmo numérico para aproximarlos que requiere de un valor inicial para arrancar. Entonces, en start se ha de especificar una list con cada parámetro a estimar igualado a un valor inicial
    - La desviación típica que nos da esta función es la 'verdadera' y no la muestral
    - El resultado de fitdistr es una list. Por tanto, para obtener el valor estimado, añadiremos $estimate y para el error estándar, $sd
  - qqplot($x, y, ...$)
    - $x, y$: dos muestras cuyos cuantiles queremos comparar
    - Si lo que queremos es comparar una muestra contra una distribución teórica, entramos como $x$ la muestra y, como $y$, la lista de los cuantiles $k/n$ de la distribución teórica
    - Como nos interesa comparar los QQ-puntos con la diagonal $y = x$, es conveniente añadir esta última con abline(0,1)
  - qqPlot($x$, distribution='distribución', *parámetros*,...): produce QQ-plots de muestras contra distribuciones teóricas más informativos
    - En el paquete car
    - $x$: la muestra
    - distribución: es el nombre de la familia de distribuciones; se ha de entrar entre comillas

- col.lines: para especificar el color de las líneas. Por defecto: rojo
- La línea recta es la que une los puntos definidos por los cuartiles primero y tercero: se llama recta cuartil-cuartil. Un ajuste de los QQ-puntos a esta recta significa que la muestra se ajusta a la distribución teórica, pero posiblemente con parámetros diferentes a los anteriores
- Las curvas discontinuas abrazan una región que representa un intervalo de confianza del 95 % para el QQ-plot: si todos los puntos caen dentro de esta franja, no hay evidencia para rechazar que la muestra provenga de la distribución teórica

- chisq.test($x, p$ =...,rescale.p=...,simulate.p.value=...): para realizar un test $\chi^2$
  - $x$: tabla o vector de frecuencias absolutas observadas de las clases en la muestra
  - $p$: vector de probabilidades teóricas de las clases para la distribución que queremos contrastar. Valor por defecto: la probabilidad es la misma para todas las clases. Obviamente, estas probabilidades se tienen que especificar en el mismo orden que las frecuencias de $x$ y, han de sumar 1
  - rescale.p: parámetro lógico que si se iguala a TRUE, indica que los valores de $p$ no son probabilidades, sino solo proporcionales a las probabilidades; esto hace que R tome como probabilidades teóricas los valores de $p$ partidos por su suma para que sumen 1. Valor por defecto: FALSE
  - simulate.p.value: parámetro lógico que indica a la función si debe optar por una simulación para el cálculo del p-valor del contraste. Valor por defecto: FALSE
  - Si se especifica simulate.p.value=TRUE, R realiza una serie de replicaciones aleatorias de la situación teórica (por defecto 2000, pero su número se puede especificar mediante el parámetro B). Cuando no se satisfacen las condiciones para que $X^2$ siga aproximadamente una distribución $\chi^2$, estimar el p-valor mediante simulaciones puede ser una buena alternativa
  - Podemos obtener el valor del estadístico $X^2$ con el sufijo $statistic
  - Podemos obtener el valor de los grados de libertado con el sufijo $parameter
  - Podemos obtener el p-valor con el sufijo $p.value
  - Si estimamos valores, el p-valor es incorrecto ya que R no lo sabe. Tendremos que calcular el p-valor a mano
  - Si queremos realizar este test para variables continuas debemos definir los intervalos de clase en primer lugar. Podemos seguir dos estrategias razonables: reutilizar los generados por la función hist o dividir el rango de la variable en un número prefijado $K$ de intervalos de amplitud fija
- ks.test($x, y$,*parámetros*): para realizar el test K-S
  - $x$: muestra de una variable continua
  - $y$: tanto puede ser un segundo vector (y entonces se contrasta si ambos vectores han sido generados por la misma distribución continua) o el nombre de la función de distribución que queremos contrastar, entre comillas
  - parámetros: de la función de distribución si se ha especificado una
- unique($x$): esta función aplicada a un vector $x$ nos da la lista de sus elementos sin repeticiones
- lillieTest($x$): para hacer el test K-S-L

    - Del paquete nortest
    - Tiene un inconveniente: le cuesta detectar diferencias prominentes en un extremo u otro de la distribución. Este inconveniente lo resuelve el test A-D
    - Se comporta mal con muestras grandes. Este otro inconveniente lo resuelve el test Shapiro-Wilk
    - Sensible a las repeticiones
  - ad.test($x$): test Anderson-Darling
    - Del paquete nortest
    - Se comporta mal con muestras grandes. Este inconveniente lo resuelve el test Shapiro-Wilk
    - Sensible a las repeticiones
  - shapiro.test($x$): test Shapiro-Wilk
    - Sensible a las repeticiones
  - dagoTest($x$): cuantifica lo diferentes que son la simetría y la curtosis de la muestra
    - El p-valor relevante es el de 'Omnibus test'
    - Del paquete fBasics

- **Tema 6. Independencia y homogeneidad**
  - addmargins(*tabla*,margin=...,FUN=...)
    - tabla: es una table
    - margin: Valor por defecto c(1,2). Es un parámetro que puede tomar los valores siguientes:
      - 1: si queremos una nueva fila con las marginales de cada columna
      - 2: si queremos una nueva columna con las marginales de cada fila
      - c(1,2): si queremos las marginales por filas y columnas
    - FUN: es la función que se aplica a las filas o columnas para obtener el valor marginal. Valor por defecto: suma
  - colSums(*tabla*): Si solo nos interesa la columna de marginales
  - rowSums(*tabla*): Si solo nos interesa la fila de marginales
  - pwr.chisq.test(N=...,df=...,sig.level=...,w=...,power=...): para calcular la potencia de un contraste $\chi^2$
    - N: tamaño de la muestra
    - df: grados de libertad
    - sig.level: nivel de significación $\alpha$
    - w: magnitud del efecto $\sqrt{\chi^2/N}$, donde $\chi^2$ es el estadístico de contraste
    - power: potencia $1-\beta$
    - Si se especifican todos estos parámetros menos 1, la función da el valor del parámetro que falta
    - Del paquete pwr

- **Tema 7. ANOVA**
  Los modelos de ANOVA en R: Sea $X$ una variable numérica y $F1$ y $F2$ dos factores de una cierta tabla de datos

  - La fórmula $X \tilde{} F1$ se usa para indicar el ANOVA de un factor $F1$ de la variable $X$
  - La fórmula $X \tilde{} F1 + F2$ se usa para indicar el ANOVA de dos factores de la variable $X$, sin tener en cuenta la interacción entre los factores. Es el tipo de fórmula que se usa en los ANOVA de bloques
  - La fórmula $X \tilde{} F1 * F2$ se usa para indicar el ANOVA de dos factores de la variable $X$ teniendo en cuenta además la interacción entre esos factores, es decir, ANOVA de dos vías
  - La fórmula $X \tilde{} F1 : F2$ se usa para indicar el ANOVA de un factor que tiene como niveles los pares de niveles de $F1$ y $F2$

  **Condiciones del ANOVA:**

  - La normalidad
  - La igualdad de varianzas

  - aov(formula,data=...): función básica de R para realizar un ANOVA
    - formula: una fórmula que especifique un modelo de ANOVA
    - data: opcional; sirve para especificar, si es necesario, el data frame al que pertenecen las variables utilizadas en la fórmula
    - El resultado no es la tabla del ANOVA. Para obtenerla hay que aplicar summary al resultado de aov:
      - ◇ En la primera columna, dos etiquetas: el nombre del factor y Residuals, que representa los errores o residuos del ANOVA
      - ◇ La segunda columna nos da los grados de libertad correspondientes al factor y a los residuos
      - ◇ La tercera columna, nos muestra las sumas de los cuadrados del factor
      - ◇ La cuarta columna contiene las medias de los cuadrados del factor
      - ◇ La quinta columna nos da el valor del estadístico de contraste
      - ◇ La sexta columna muestra el p-valor del contraste
      - ◇ La séptima indica el nivel de significación del p-valor según el código usual. A mayor número de asteriscos, más significativo es el p-valor y por lo tanto hay más evidencia para rechazar la $H_0$ de que las medias comparadas son iguales
    - options(show.signif.stars=FALSE): permite eliminar los símbolos que marcan los niveles de significación de los p-valores
    - Para extraer los datos de la tabla ANOVA obtenida con la función summary(...) hay que añadir el sufijo adecuado *variable*[[1]]$*etiqueta*
    - Siempre se ha de usar un factor (o varios) para separar la variable numérica en subpoblaciones
  - anova(lm(formula,data=...)): Es una manera alternativa de realizar un ANOVA. Con esta construcción obtenemos directamente la tabla, sin necesidad de aplicar summary

  - El resultado de anova(lm(...)) es un data frame, por lo que para extraer los valores de la tabla ANOVA que produce, podemos usar la sintaxis usual de los data frames
  - Siempre se ha de usar un factor (o varios) para separar la variable numérica en subpoblaciones
- interaction.plot(*factor 1, factor 2, variable*): Gráfico de interacción entre factores
  - Si queremos cambiar la etiqueta del factor en la leyenda usaremos trace.label
  - Si queremos reordenar los factores en dicha leyenda usamos fixed=TRUE
  - En estos gráficos se dibuja una línea quebrada para cada nivel del primer factor. Esta línea une, mediante segmentos, los valores medios que toma la variable $X$ para cada nivel del segundo factor en el nivel del primer factor correspondiente. Si no hay ninguna interacción entre estos factores, las líneas serán paralelas. Cuanto más se alejen de ser paralelas, más evidencia de interacción habrá entre estos dos factores
- subset(*data frame, condición*): Define un data frame con las filas del data frame introducido como parámetro que cumplen la condición
- bartlett.test(*variable ~ factor*, data=...): Test de Bartlett para el contraste de igualdad de varianzas
  - En los ANOVA de dos vías, se ha de comprobar la igualdad de varianzas para las combinaciones de niveles de los factores. Esto se hace mediante la instrucción bartlett.test(*variable* ~ interaction(*factor1,factor2*), data=...)
- fligner.test(): Test de Fligner-Killeen, otro test de igualdad de varianzas
- leveneTest(): Test de Levene, otro test de igualdad de varianzas
  - Del paquete car
  - No requiere que las poblaciones sean normales. Además, cuando estas son normales, el test de Bartlett es más potente
- pairwise.t.test(*varianza, factor*, paired=...,p.adjust,method="..."): t-tests por parejas
  - p.adjust.method: método de ajuste de p-valores que deseamos usar. Unas de las posibles opciones son "holm", "bonferroni","hochberg". Valor por defecto es el de Holm
  - paired: sirve para indicar si las muestras son independientes (FALSE) o emparejadas(TRUE). Valor por defecto, FALSE. Cuando el ANOVA es de bloques, las muestras son emparejadas
  - Hochberg es el más potente que Holm, que a su vez es más potente que Bonferroni
  - Los p-valores ya están ajustados por lo que se deben comparar directamente con el nivel de significación global $\alpha$
- duncan.test(*anova, factor*,alpha=...,group=...)$*sufijo*: Test de Duncan para comparar los pares de medias
  - anova: resultado de la función aov (sin summary) con la que hemos calculado el ANOVA de partida
  - factor: es el factor del ANOVA y se ha de entrar entrecomillado y con el mismo nombre que se ha usado en la fórmula del aov

- alpha: para entrar el nivel de significación $\alpha$. Valor por defecto, $\alpha = 0{,}05$
- group: puede valer TRUE o FALSE, y hace que el resultado se presente de forma diferente
- sufijo: tiene que ser groups si group=TRUE y comparison si group=FALSE
- Del paquete agricolae

- TukeyJSD(*anova*): método HSD de Tukey. Es el método más preciso de comparación de parejas de medias para contrastes ANOVA de un factor en los que cada nivel tenga el mismo número de observaciones

## Tema 8. Regresión lineal simple y múltiple

- lm(formula,data=...,subset=...): Para realizar la regresión lineal
  - formula: describe la variable dependiente y las variables independientes del modelo. En primer lugar se indica la variable dependiente que necesariamente tiene que ser una variable numérica. A continuación, tras ~ se escriben las variables independientes separadas por el símbolo +. Es decir, formula=$y \sim x_1 + ... + x_k$
  - data: Opcional, sirve para especificar el *data frame* al que pertenecen las variables utilizadas
  - subset: Opcional, sirve para especificar que la regresión solo tenga en cuenta un subconjunto de observaciones
  - Si disponemos de muchas variables independientes y se quiere obtener la recta de regresión considerándolas todas hay que introducir la instrucción lm($y \sim$.,data=*data frame*)
- identify$(x, y)$: para obtener a qué observación corresponde un punto
  - $x, y$: coordenadas del punto
- summary(lm(...)): función que nos da más información sobre la recta de regresión calculada
  - Residuals: se proporciona un resumen de los residuos o errores $e_i$ del modelo
  - Coefficients: en la columna Estimate se dan los coeficientes de cada variable de la recta de regresión, junto a sus respectivas desviaciones estándar en la columna Std. Error. Las columnas t value y Pr($> |t|$) proporcional el valor del estadístico y el p-valor del contraste $H_0 : \beta_i = 0$ y $H_1 : \beta_i \neq 0$
  - Residual standard error: corresponde a la raíz del valor estimado de la varianza común de los residuos, junto con los grados de libertad $n - k - 1$
  - Multiple R-squared: valor del coeficiente de determinación. Cuanto su valor sea más cercano a 1, mejor aproxima la ecuación de regresión el conjunto de puntos
  - Adjusted R-squared: valor del coeficiente de determinación ajustado. Cuanto su valor sea más cercano a 1, mejor aproxima la ecuación de regresión el conjunto de puntos
  - En la última fila se indican el valor del estadístico $F$, los grados de libertad $n - k - 1$ y el p-valor en este orden del siguiente contraste ANOVA:

$$\begin{cases} H_0: & \beta_1 = ... = \beta_k = 0 \\ H_1: & \text{Existe al menos un } \beta_i \neq 0 \end{cases}$$

- update($x$,formula,subset=...): Permite recalcular la recta de regresión a partir de una recta de regresión anterior
  - $x$: model generado por una función como lm
  - formula: indica un cambio en la fórmula especificada para obtener el nuevo modelo. Su estructura es . ~.-$X_i$ donde se indica que se utilice la misma fórmula considerada en el modelo $x$ pero ahora sin tener en cuenta la variable independiente $X_i$
  - subset: opcional y con el mismo significado que la función lm
  - summary(update(...)): la salida es similar a la de lm
- confint(object, parm, level=0.95): cálculo de I.C. para los coeficientes $\beta_i$
  - object: modelo de regresión lineal, es decir, la salida de la función lm
  - parm: indica para qué parámetros se tienen que calcular los I.C. Tanto se puede introducir un vector de números como un vector de nombres de parámetros. Por defecto, se calculan los intervalos para todos los parámetros
  - level: nivel de confianza del intervalo. Valor por defecto: 95 %
  - En la salida de la función, se indican el extremo inferior y superior del I.C mediante el nivel del cuantil utilizado para su cálculo para el parámetro correspondiente a cada variable independiente, indicada en la primera columna, más el término independiente
  - Una de las utilidades básicas de los intervalos de confianza es determinar si la variable correspondiente aporta información al modelo o no. Este hecho se puede determinar comprobando si 0 pertenece al intervalo de confianza respectivo
  - El hecho de que 0 pertenezca al I.C. de $\beta_0$ no afecta a la validez del modelo
- predict(object, newdata, interval=c('none','confidence','prediction'), level=0.95): función que resultará útil para el cálculo de I.C. para $\mu_{Y|_{x_1,...,x_k}}$ y para $y_0 = y(x_1, ..., x_k)$
  - object: modelo de regresión lineal, es decir, la salida de la función lm
  - newdata: corresponde a un *data frame* donde la función recaba los valores de las variables con las que se predice. Podemos indicar un único valor para una única variable $x$ como data.frame(x=x0), varios valores para una única variable $x$ como data.frame(x=seq(x0,x1,p)) o valorespara diversas variables $x_1, ..., x_k$ como data.frame(x1=x10, x2=x20, ..., xk=xk0).
  - interval: es un parámetro con tres posibles valores. Si se indica 'none', simplemente se calcula el valor predicho por la recta de regresión para los valores indicados en newdata. Si se indica 'confidence' se determina el I.C. para el valor esperado $\mu_{Y|_{x_10,...,x_k0}}$ donde $x_10, ..., x_k0$ son los valores asignados a las variables independientes en newdata. Finalmente, si se indica 'prediction', se calcula el I.C. para el valor $y_0 = y(x_1, ..., x_k)$
  - level: vuelve a indicar el nivel de confianza del intervalo
  - La salida de la finción predict es un vector con tres componentes. En fit encontramos el valor predicho por la recta de regresión para los valores establecidos en newdata y, en los otros dos, los límites inferior y superior del I.C.
- extractAIC($x, k$): calcula las medidas AIC y BIC para comparar modelos de regresión múltiple con un número distinto de variables

- $x$: modelo lineal obtenido mediante la función lm
- $k$: peso que afecta al número de variables del modelo. Valor por defecto: $k = 2$ (correspondiendo al criterio AIC)
- Si se indica $k$=log($n$) donde $n$ es el número de observaciones, extractAIC calcula la medida BIC.
- Como menor sea el valor de AIC o BIC, mejor es el modelo. Los valores de estas medidas se obtienen a partir de extractAIC($x, k$)[2]

• step($x, k$): para comparar modelos de regresión múltiple con un número distinto de variables
- $x$: modelo lineal obtenido mediante la función lm
- $k$: peso que afecta al número de variables del modelo. Valor por defecto: $k = 2$ (correspondiendo al criterio AIC).
- Si se indica $k$=log($n$) donde $n$ es el número de observaciones, step ejecuta en base a BIC.
- Esta función realiza un proceso secuencial de eliminación o adición de variables consiguiendo en cada paso del algoritmo un modelo con un valor de AIC (o BIC) menor al modelo obtenido en el paso anterior

## 3. Extra

### ▪ Clustering

• kmeans($x$,centres,iter.max=...)
- x: matriz con los puntos $x_i$ como filas
- centres: matriz con los centros $c_i$ de partida como filas, o el número $k$
- iter.max: número máximo de iteraciones
- Si queremos que ejecute el algoritmo de Lloyd hemos de introducir como parámetro algorithm='Lloyd'
- Componentes de la list kmeans:
  - ◇ cluster: asignaciones de elementos a clusters
  - ◇ centers: los centros de los clusters
  - ◇ totss: suma de los cuadrados de las distancias de los puntos al punto medio de todos estos puntos
  - ◇ withinss: vector de las sumas, para cada cluster, de los cuadrados de las distancias de sus puntos al punto medio de su cluster
  - ◇ tot.withinss: usma de withinss, $SS_c$
  - ◇ betweenss: diferencia totss-tot.withinss. Es la suma ponderada por el número de objetos del cluster correspondiente, de los cuadrados de las distancias de los centros de los clusters al punto medio de todos los puntos
  - ∼ A nosotros nos interesa betweenss/totss que mide la fracción de la variabilidad de los datos que explican los clusters. Cuanto más grande mejor

• hclust($d$,method=" ... "): para calcular un clustering jerárquico aglomerativo

- d: matriz de distancias
- method: sirve para especificar el método: 'single', 'complete', 'average', 'ward', 'median', 'centroid',...
- *variable*$merge: muestra la formación de clusters
- *variable*$height: muestra las distancias mínimas

- plot(clust,labels=...,hang=...,...): para representar un clustering por medio de un dendograma
  - clust: es un hclust
  - labels: sirve para poner nombre a los objetos
  - hang: sirve para especificar la posición de las etiquetas
  - Le podemos añadir otros parámetros usuales de los plots
- dist($x$,method=" ... "): para calcular la distancia entre las filas de una matriz
  - $x$: matriz de datos
  - method: sirve para especificar el método: 'euclidean', 'manhattan',...