Master en Big Data. Fundamentos matemáticos del análisis de datos.

Sesión 3. Distribuciones. Valores centrales y dispersión.

Fernando San Segundo

Curso 2020-21. Última actualización: 2020-09-13

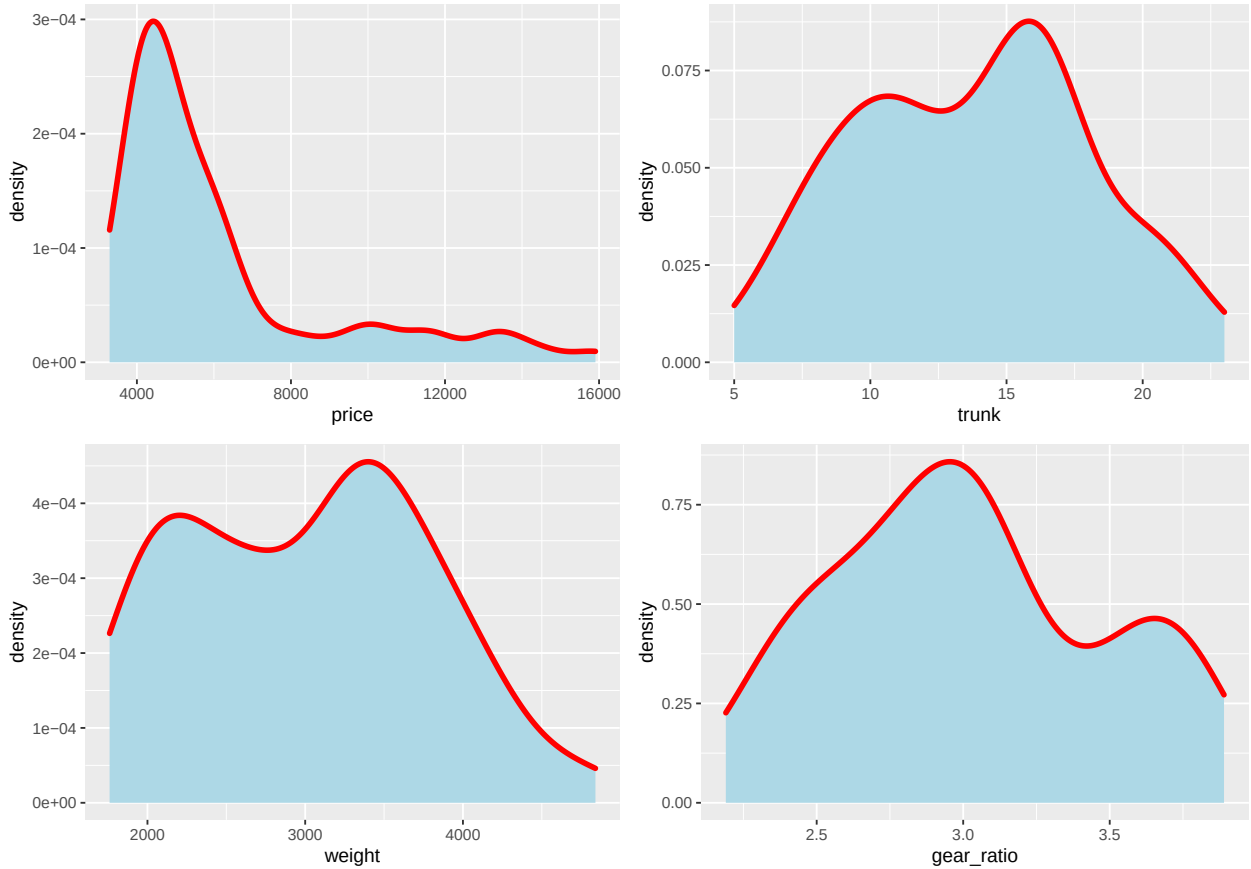
COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI
DEPARTAMENTO DE MATEMÁTICA APLICADA

1. Distribuciones.
2. Valores centrales, de posición y dispersión.
3. Factores.
4. Complementos de R: listas, bucles, funciones, datos ausentes, Rmarkdown.

Section 1

Distribuciones.

- En un ejercicio previo proponíamos dibujar las curvas de densidad de varias variables de la tabla `auto2`. Si lo haces obtendrás curvas como estas:

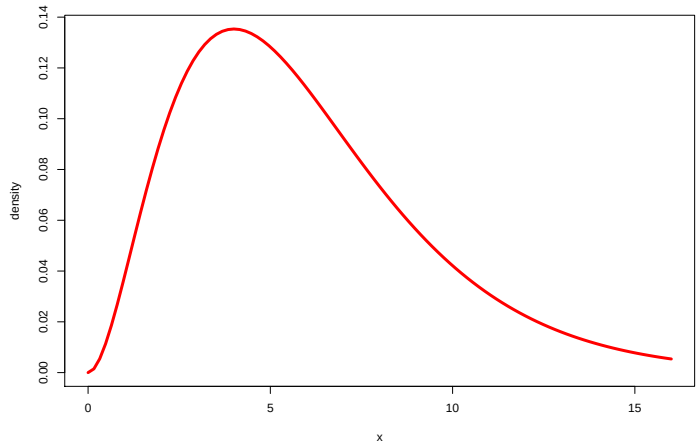


Mira el código de esta sesión para ver como dibujar esta *"tabla de figuras"*.

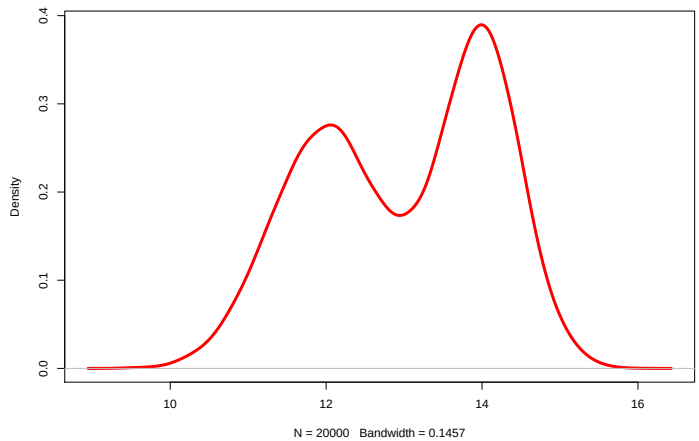
- Cada una de esas curvas muestra una *distribución de los valores*, que es una representación de la forma en la que se reparten los datos entre los valores posibles. Las distribuciones van a jugar un papel central en las próximas sesiones, así que vamos a desarrollar un poco de lenguaje para referirnos a ellas.

Distribuciones unimodales y multimodales.

- Una distribución que presenta un único máximo se denomina *unimodal*

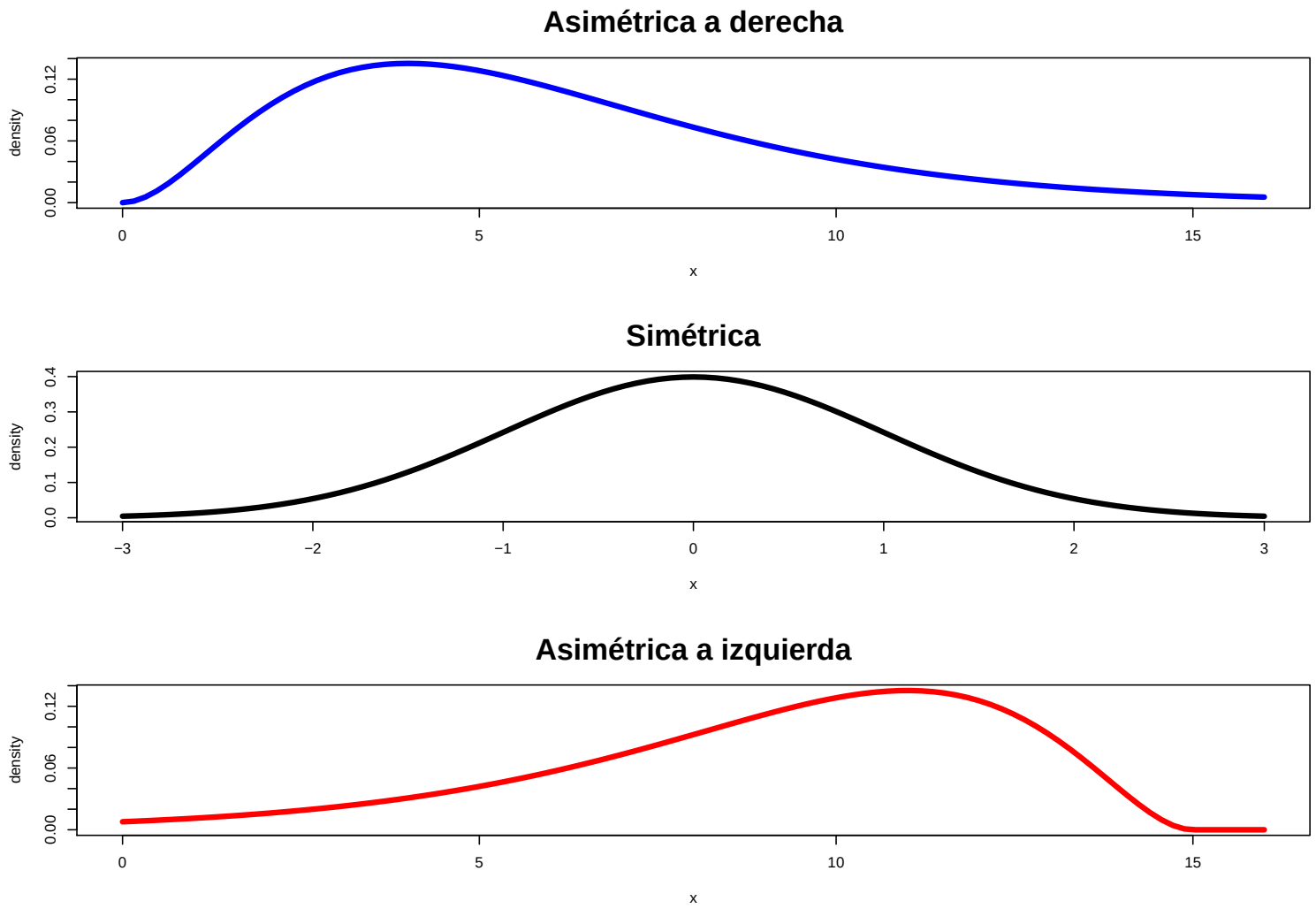


- Mientas que una distribución con dos máximos locales claramente definidos como la de la figura se denomina *bimodal* (o *multimodal* cuando son más de dos).



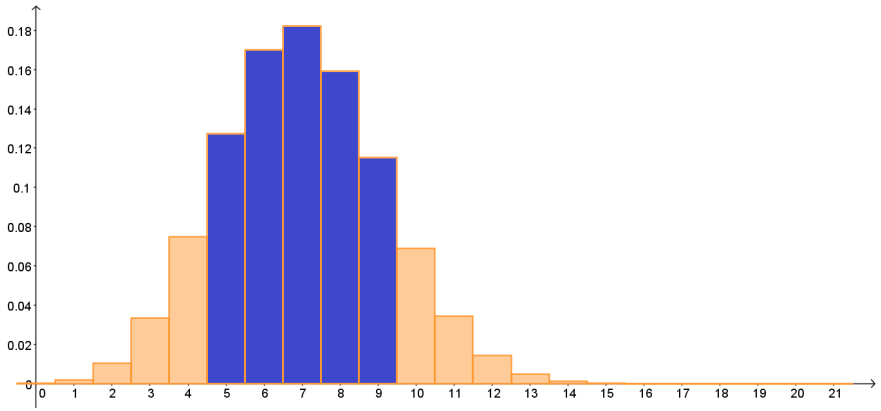
Distribuciones asimétricas.

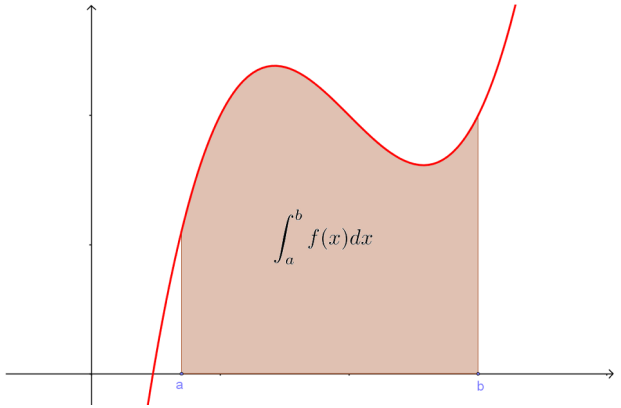
- Otra característica de las distribuciones en la que nos vamos a fijar a menudo es su simetría. Fíjate en que es la *cola* más larga de la distribución la que da nombre a la asimetría. En inglés esta característica se denomina skewness



Distribuciones y cálculo de probabilidad. Discusión informal.

- La distribución de una variable discreta con valores enteros se representa adecuadamente mediante un histograma. Si las alturas de las barras del histograma representan frecuencias relativas entonces la probabilidad de que un valor elegido al azar caiga en el intervalo $[a, b]$ se obtiene sumando el área de todas las barras desde a hasta b, como se ilustra en la parte izquierda de la figura.
- Por su parte, La distribución de una variable continua se representa mediante una curva de densidad. Y entonces la probabilidad de que un valor elegido al azar caiga en el intervalo $[a, b]$ se obtiene calculando (con una integral) el área bajo la curva desde a hasta b





- Don't panic! No hace falta que sepas calcular integrales. Pero es importante entender que el cálculo de probabilidades está estrechamente relacionado con el cálculo de áreas.

Section 2

Valores centrales, de posición y dispersión.

La media aritmética

- Al trabajar con variables cuantitativas muchas veces trataremos de elegir un *valor central*. Es decir, un valor que sea *representativo* de los valores que toma la variable. Un buen valor central debería ser la respuesta a esta pregunta: *"si elijo un valor de la variable al azar ¿lo más probable es que se parezca a . . . ?"*
- El valor central más conocido es la *media aritmética*. Dados n números $x_1, x_2, \ldots, x_n$ su media aritmética es:

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{\sum_{i=1}^{n} x_i}{n}$$

- En R se utiliza la función `mean` para calcular la media de los valores de un vector numérico. Con este código vamos a elegir 20 números al azar entre 0 y 100 (con remplazamiento). Recuerda que los paréntesis sirven para que R muestre el resultado.

```
(muestra = sample(0:100, size = 20, replace = TRUE))
```

```
##  [1] 24 41 68 16 60 75 18 25  8 89 45 71 22 93 71 66 83 46
## [19] 68 66
```

```
(media = mean(muestra))
```

```
## [1] 52.75
```

- **Ejercicio:** Repítelo varias veces y mira las medias que obtienes. ¡Al pensar en esas medias estás empezando a hacer Estadística!

La media y los valores atípicos.

- La media aritmética se usa mucho porque su definición matemática es muy sencilla. Pero tiene un problema: su valor se ve muy afectado por la presencia de valores anormalmente grandes o pequeños de la variable, los llamados *valores atípicos* (que muy pronto vamos a definir con más rigor).
- Por ejemplo, si elegimos 99 números al azar de 0 a 100, su media es parecida a 50 como cabría esperar.

```
set.seed(2019)
muestra = sample(0:100, size = 99, replace = TRUE)
(media = mean(muestra))
```

```
## [1] 51.64646
```

Pero si ahora añadimos un único valor igual a 1000 y volvemos a calcular la media:

```
muestra2 = c(muestra, 1000)
(media2 = mean(muestra2))
```

```
## [1] 61.13
```

Las dos muestras son esencialmente idénticas, así que al elegir un valor representativo (en el sentido probabilístico que hemos discutido) nos gustaría obtener respuestas mucho más parecidas. Pero como ilustra este ejemplo, la media se ve muy afectada por ese único valor atípico.

La mediana.

- Para paliar este problema podemos usar la *mediana* como sustituto o alternativa a la media. La mediana de un conjunto de valores se define con esta receta: se ordenan los números de menor a mayor y se toma el valor que queda en el centro (si hay una cantidad impar de valores; si son pares se promedian los dos valores centrales).
- Por ejemplo, para calcular la mediana de estos 17 valores:

```
set.seed(2019)
(valores = sample(1:100, 17, replace = TRUE))
```

```
##  [1] 25 42 69 17 61 76 19 26  9 90 46 72 23 94 72 67 84
```

los ordenamos y tomamos el valor central

```
(ordenados = sort(valores))
(mediana = ordenados[9])
```

```
##  [1]  9 17 19 23 25 26 42 46 61 67 69 72 72 76 84 90 94
## [1] 61
```

Aunque lo mejor, claro, es usar la función de R:

```
median(valores)
```

```
## [1] 61
```

Más detalles sobre la mediana.

- Por su propia construcción (en términos de posiciones/rangos y no de tamaños) debería estar claro que la mediana no se ve muy afectada por la presencia de valores atípicos. Si volvemos a la muestra de 99 valores que construimos antes y miramos su mediana antes y después de añadir el valor atípico 1000 se obtiene:

```
median(muestra)
```

```
## [1] 54
```

```
median(muestra2)
```

```
## [1] 54
```

  En este ejemplo concreto la mediana no se ve afectada en absoluto por ese valor.

- Y entonces, ¿por qué no se usa siempre la mediana en lugar de la media aritmética? Pues porque la definición de la mediana utiliza unas matemáticas bastante más complicadas que la media aritmética. Gracias al ordenador la importancia de los métodos basados en la mediana ha ido aumentado. Pero los métodos clásicos que usan la media aritmética siguen siendo los más comunes.
- En cualquier caso, recomendamos explorar bien los datos, identificar posibles valores atípicos y comprobar si la media y la mediana son parecidas.

Otros valores de posición. Recorrido. Función summary.

- **Cuartiles y percentiles.** La mediana se puede pensar como el valor que es mayor o igual que el 50% de los valores de nuestro conjunto de datos. De la misma forma se definen el *primer y tercer cuartil* como los valores que son mayores o iguales respectivamente que el 25% y el 75% de los valores. Esto se generaliza a la noción de percentil. El percentil 43, por ejemplo, es un valor que deja por debajo al 43% de los valores. En R ese percentil se calcula para una variable como `cty` de `mpg` mediante:

```
quantile(mpg$cty, probs = 0.43)
```

```
## 43%
##  16
```

- El *recorrido* (del inglés *range*) de un conjunto de valores es el intervalo que va del valor mínimo al máximo. Cuidado: en español se usa a menudo erróneamente *rango*. Pero los rangos en Estadística son otra cosa (que corresponde a *rank* en inglés).
- Una manera sencilla de obtener varias de estas medidas de posición es usando `summary`

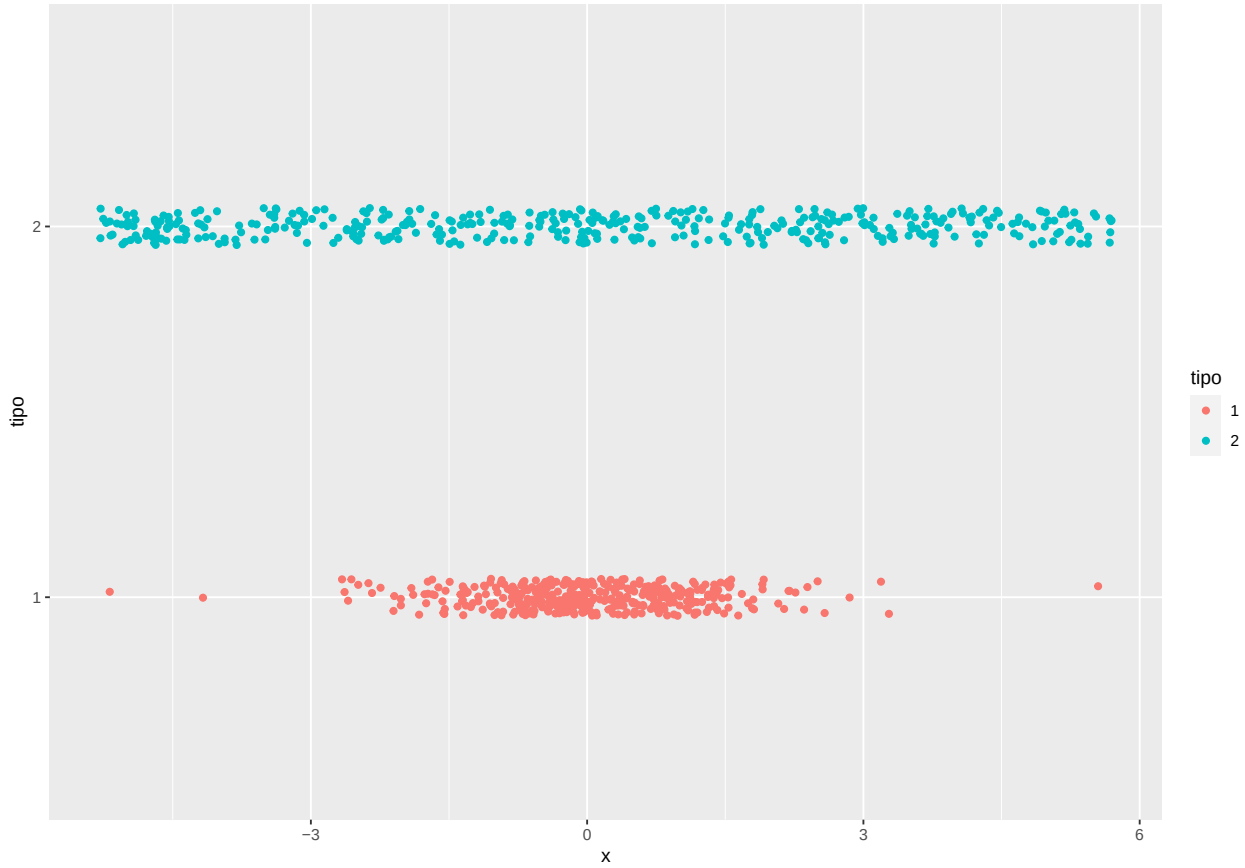
```
summary(mpg$cty)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   14.00   17.00   16.86   19.00   35.00
```

- **Ejercicio.** Prueba a aplicar la función a todo el `data.frame` con `summary(mpg)`.

Dispersión

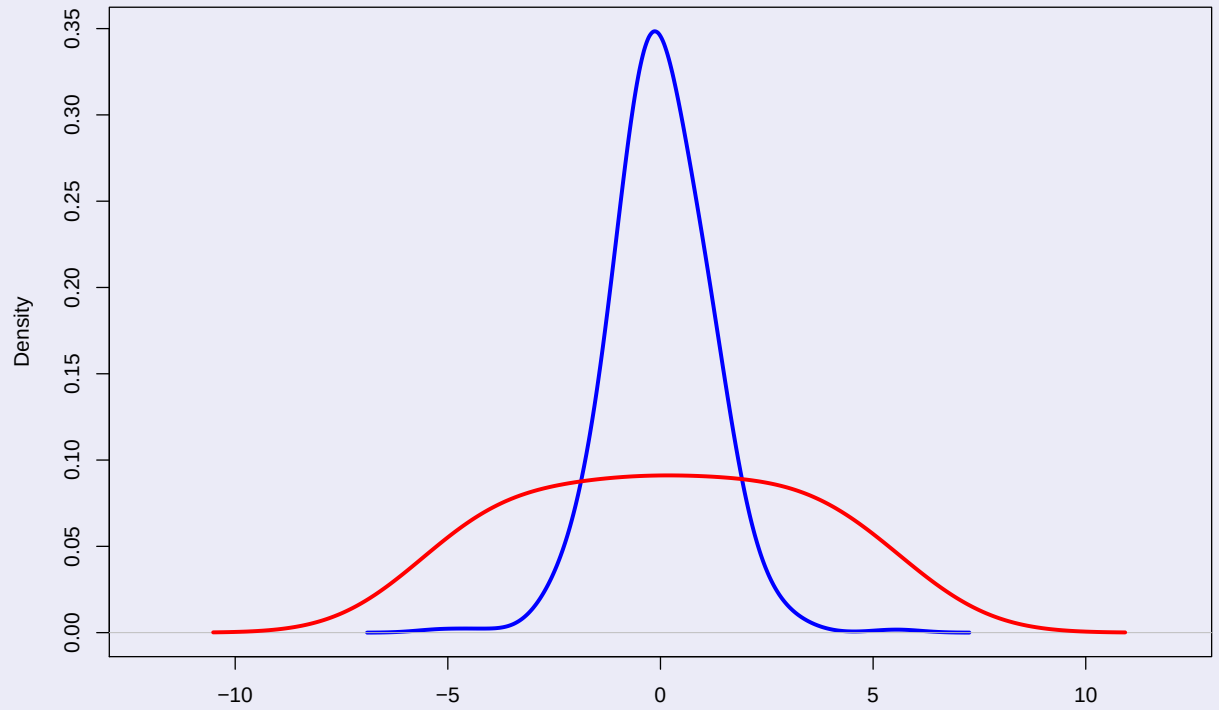
- La siguiente figura muestra dos conjuntos de valores, ambas con media 0 y el mismo número de puntos. Las coordenadas verticales de los puntos se han modificado con un poco de *ruido* para hacer mejorar la visualización de ambos conjuntos. Puedes mirar el código de esta sesión para ver como se ha construido esta figura.



- ¿Qué diferencia a estos dos conjuntos de datos?

Dispersión en los diagramas de densidad.

- En esta otra figura puedes ver juntas las curvas de densidad de esos dos conjuntos de datos.



  Este diagrama ilustra lo que la mostraban las nubes de puntos de la figura previa: los puntos de un conjunto están más agrupados en torno al centro que los del otro conjunto. La diferencia entre ambos conjuntos está en la **dispersión**.
- La idea intuitiva de la dispersión está clara: se trata de medir como de agrupados en torno al centro están los valores. ¿Como se define rigurosamente? Al igual que sucedía con los valores centrales (media vs mediana) hay varias posibilidades.

Recorrido intercuartílico. Valores atípicos.

- La primera forma de medir la dispersión que vamos a ver se basa en observar la diferencia entre los cuartiles tercero (75%) y primero (25%). Ese número es el *recorrido intercuartílico*; un intervalo de longitud IQR desde el primer hasta el tercer cuartil contiene siempre al 50% central de los valores. En R se calcula con:

```
IQR(mpg$cty)
```

```
## [1] 5
```

Fíjate en que es precisamente la diferencia entre esos dos cuartiles.

```
summary(mpg$cty)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   14.00   17.00   16.86   19.00   35.00
```

- El IQR nos permite dar una definición formal de los **valores atípicos:** son aquellos valores que superan (tercer cuartil) + 1.5 · IQR o quedan por debajo de (primer cuartil) - 1.5 · IQR. Por ejemplo, en `mpg$cty` será atípico cualquier valor fuera de este intervalo

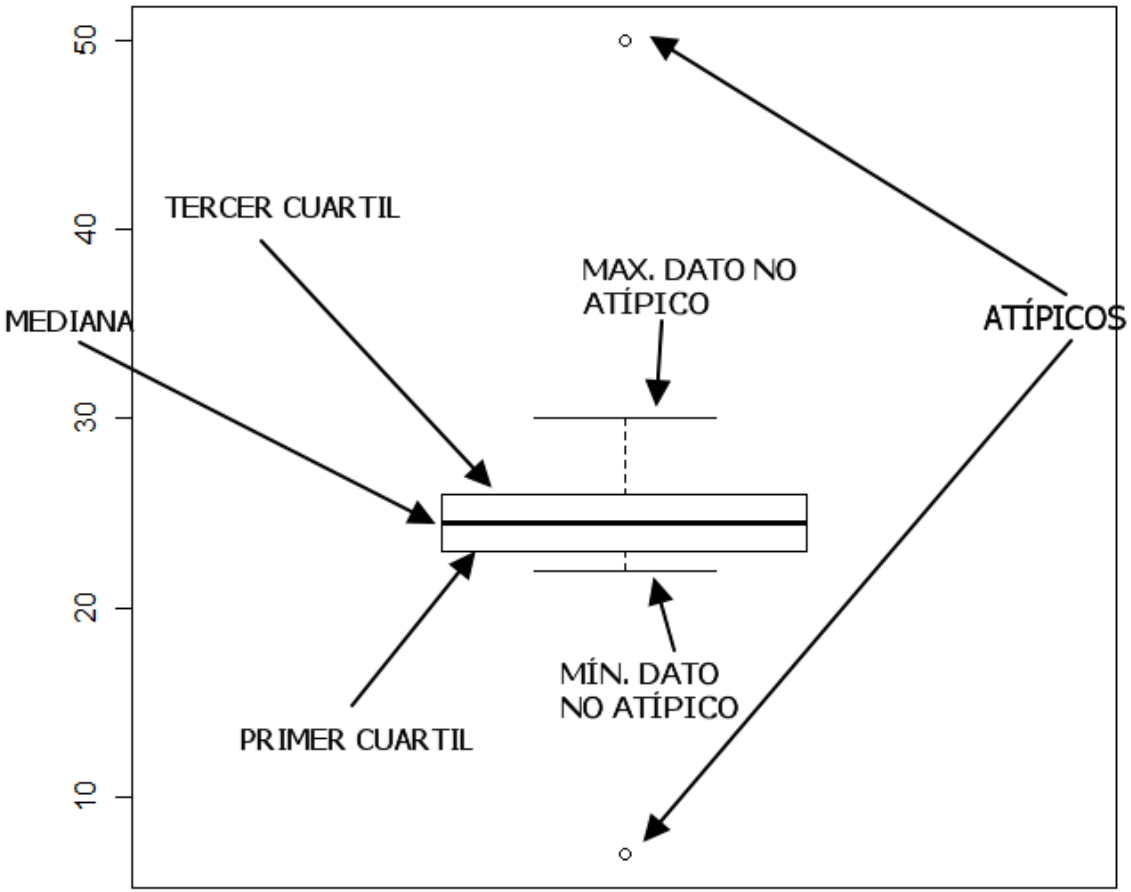
```
unname(quantile(mpg$cty, probs = c(1/4, 3/4)) + c(-1, 1) * 1.5 * IQR(mpg$cty))
```

```
## [1]  6.5 26.5
```

- **Ejercicio:** Mira el código de esta sesión y calcula el IQR de los dos conjuntos de datos de las páginas previas.

Boxplots

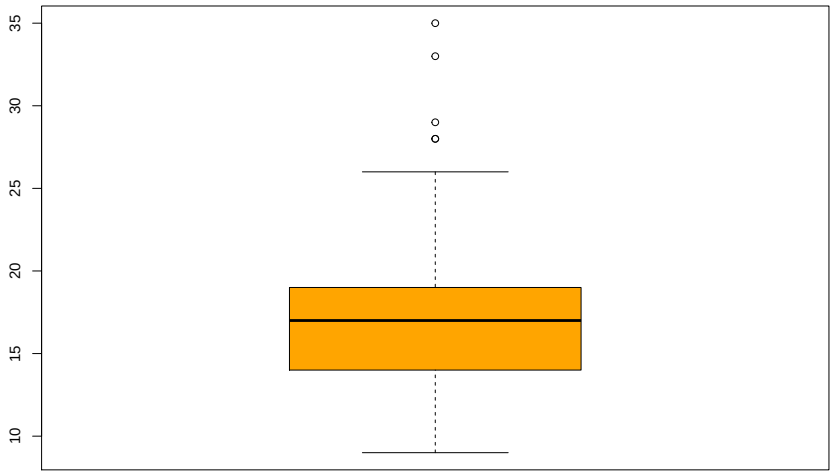
- El *boxplot* o diagrama de caja y bigotes (box and whiskers) es una forma de representar gráficamente estas medidas de posición. La estructura de un boxplot es la que se describe en esta figura:



Boxplots con R básico.

- Dibujar un boxplot con R básico es muy sencillo. Por ejemplo, para la variable `cty`:

```
bxp_cty = boxplot(mpg$cty, col="orange")
```



- Además al darle un nombre al boxplot podemos usar ese nombre para acceder a varios componentes del boxplot. Por ejemplo los valores atípicos en el vector de datos:

```
bxp_cty$out
```

```
## [1] 28 28 33 35 29
```

- **Ejercicio:** ¿Cuáles son las *posiciones que ocupan* los valores atípicos de esta variable? Además haz un boxplot de `speed` en `accidentes` y busca en la ayuda del boxplot como dibujar el gráfico sin los valores atípicos.

Boxplot con ggplot

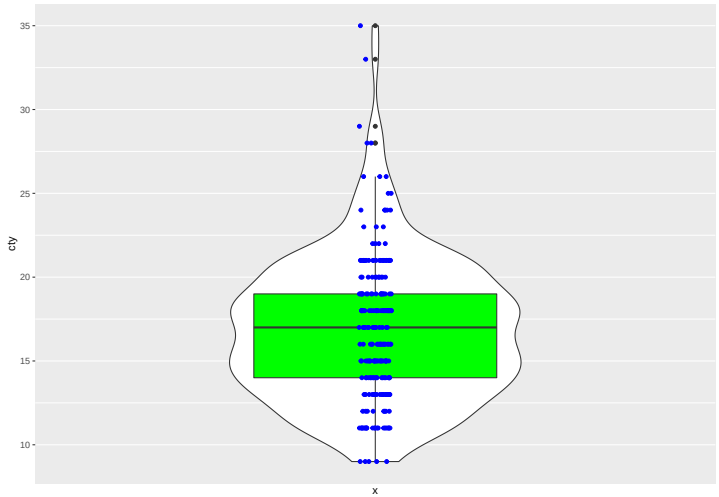
- En este caso usamos:

```
ggplot(mpg) +
  geom_boxplot(mapping = aes(y = cty), fill="orange") +
  scale_x_discrete(breaks = c())
```

Boxplot con datos y violinplots.

- Aunque los boxplots son muy útiles y se usan mucho, tienen sus limitaciones. La principal de ellas es que no muestran la distribución subyacente de valores (¡la *forma*!). Hay dos formas de combatir ese problema:
  - incorporando los valores al gráfico
  - usando un *violinplot* que es básicamente un boxplot con curvas de densidad añadidas.
- Esos dos remedios se pueden aplicar por separado o combinados como hemos hecho en este gráfico de ggplot:

```
ggplot(mpg) +
  geom_violin(mapping = aes(x=0, y = cty)) +
  scale_x_discrete(breaks = c()) +
  geom_boxplot(mapping = aes(y = cty), fill="green") +
  geom_jitter(aes(x=0, y = cty),
              position = position_jitter(w=0.05, h= 0), col="blue")
```



Desviación absoluta mediana

- Además del rango intercuartílico se usan otras medidas de dispersión. Las más comunes están basadas en la idea de medir la *desviación absoluta* de cada dato individual con respecto a un valor central. Si los datos son $x_1, x_2, \cdots, x_n$ y el valor central (puede ser la media o la mediana) es c entonces las desviaciones absolutas son:

$$d_1 = |x_1 - c|, \quad d_2 = |x_2 - c|, \quad \cdots \quad d_n = |x_n - c|$$

  Usamos valores absolutos para evitar que las desviaciones por exceso compensen a las desviaciones por defecto. Aunque el valor absoluto es complicado...

- Para obtener una medida de dispersión buscamos un valor representativo (valor central) de las desviaciones absolutas. Por ejemplo, la mediana de las desviaciones absolutas respecto de la mediana (usando c igual a la mediana de los datos). Este valor es la **desviación absoluta mediana (MAD)**, que en R se calcula así:

```
library(readxl)
accidentes = read_excel("../datos/train_acc_2010.xls")
mad(accidentes$Speed, constant = 1)
```

```
## [1] 4
```

- **Ejercicio:** calcula este valor a partir de los datos usando `median`. Mira lo que sucede si quitas la opción `constant = 1`. Lee la ayuda de `mad` y esta página.

- **Ejercicio:** calcula `mean(accidentes$Speed - mean(accidentes$Speed))`. Cambia de variable y de tabla y repite el cálculo. ¿Por qué pasa esto?

Varianza y desviación típica (definición poblacional).

- Usar los valores IQR o MAD como medidas de dispersión tiene inconvenientes similares a los que planteamos al comparar media y mediana. Las matemáticas de medianas y valores absolutos son complicadas y los resultados teóricos asociadas son más difíciles. Por esa razón tradicionalmente la Estadística ha usado una medida de dispersión basada en la media y con matemáticas más simples.
- Dados los valores numéricos $x_1, x_2, \cdots, x_n$ su **varianza (en sentido poblacional)** es la *media de las desviaciones cuadráticas respecto de la media aritmética* $\bar{x}$:

$$\sigma_x^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$$

y la **desviación típica (también poblacional)** es la raíz cuadrada de la varianza:

$$\sigma_x = \sqrt{\sigma_x^2} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

Elevamos al cuadrado para conseguir un efecto similar al valor absoluto, pero usando una función derivable. Pero al hacerlo cambiamos las unidades y tenemos que tomar raíz cuadrada para obtener la dispersión en las unidades originales del problema.

Varianza y desviación típica muestrales.

- Al introducir la varianza y desviación típica les hemos puesto el apellido de *poblacionales*. Y tal vez te hayas dado cuenta de que no hemos dicho como calcular esos valores en R.
- La razón para hacer esto empezará a quedar más clara en las próximas sesiones y tiene que ver con la esencia misma de la Estadística. La misión fundamental de la Estadística se puede resumir en **obtener información fiable sobre la población a partir de una muestra.** En particular, para obtener información sobre la media y la dispersión en la población usaremos la media y la dispersión en la muestra. Y ese es el problema: usar en la muestra la misma fórmula de la varianza que en la población *no funciona bien*.
- El remedio es sencillo, afortunadamente. Hay que usar en la muestra las fórmulas *muestrales* para la varianza y desviación típica:

$$S_x^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1} \qquad s_x = \sqrt{s_x^2} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

como ves, la diferencia estriba en que dividimos por $n-1$ en lugar de n.

Las funciones var y sd.

- Para calcular la varianza muestral en R usamos `var`:

```
var(mpg$displ)
```

```
## [1] 1.669158
```

puedes comprobar que esto es lo mismo que:

```
n = length(mpg$displ)
media = mean(mpg$displ)
sum((mpg$displ - media)^2) / (n - 1 )
```

```
## [1] 1.669158
```

- Y para calcular la desviación típica muestral usamos `sd` (de *standard deviation*):

```
sd(mpg$displ)
```

```
## [1] 1.291959
```

puedes comprobar que esto es lo mismo que:

```
sqrt(var(mpg$displ))
```

```
## [1] 1.291959
```

Section 3

Factores.

Tablas de frecuencia y terminología para factores.

- Hasta ahora hemos centrado la discusión en variables cuantitativas, con valores numéricos. Vamos a tratar aquí muy brevemente las variables cualitativas o factores. Habrá muchas ocasiones en las próximas sesiones de practicar su uso.
- Al trabajar con factores sigue teniendo sentido usar tablas de frecuencias absolutas y relativas (pero no acumuladas, en general):

```
table(accidentes$TrkType)
```

```
##
## Industry     Main Not rptd   Siding     Yard
##      247      975        3       56     1340
```

```
prop.table(table(accidentes$TrkType))
```

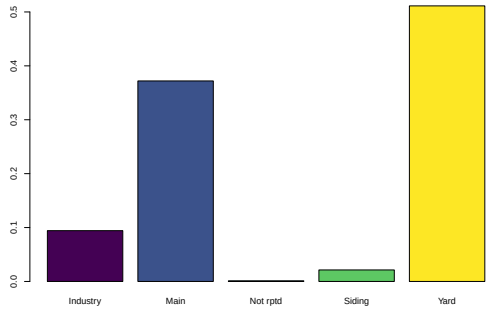
```
##
##    Industry        Main    Not rptd      Siding        Yard
## 0.094238840 0.371995422 0.001144601 0.021365891 0.511255246
```

- **¡Con factores genéricos los valores centrales o la dispersión no tienen sentido!**
- Los distintos valores de un factor se suelen llamar *niveles (levels)* del factor. Un factor *dicotómico* (o binario) tiene dos niveles; es decir, solo toma dos valores distintos. Si toma más de dos valores el factor es *politómico*. El nivel más frecuente es la *moda* del factor.

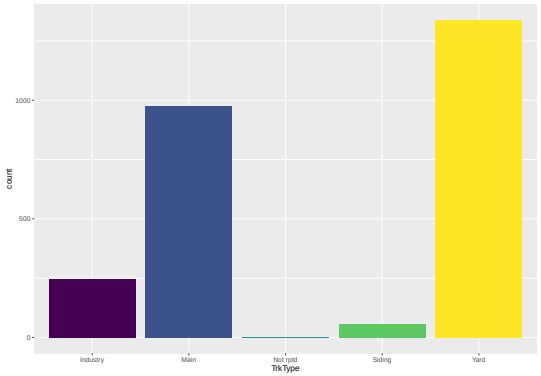
Gráficos para factores: diagramas de barras.

- El diagrama más simple y a menudo el más efectivo para representar la tabla de frecuencias de los niveles de un factor es el diagrama de barras. En R básico y ggplot respectivamente:

```
library(viridisLite)
barplot(prop.table(table(accidentes$TrkType)), col=viridis(5))
```




```
ggplot(accidentes) +
  geom_bar(mapping = aes(x = TrkType), fill= viridis(5))
```



- **Se desaconseja** el uso de diagramas de tarta (*piecharts* en inglés).

Factores dicotómicos: variables binarias (de Bernouilli).

- Un factor dicotómico solo tiene dos niveles (valores). Esos valores se pueden representar como sí/no, cierto/falso, éxito/fallo, etc. Los factores dicotómicos también se llaman a menudo variables binarias o de Bernouilli.
- *Media como proporción.* Si pensamos en los niveles de un factor binario como cierto/falso, entonces al igual que hace R podemos identificar cierto/falso con 1/0. Esta representación numérica tiene una ventaja importante. Aunque hemos dicho que los valores centrales no tienen sentido en factores genéricos, en este caso concreto la media aritmética de un vector de unos y ceros *mide la proporción* de valores iguales a 1.
- Veamos un ejemplo. La variable `male` de la tabla `fhs` es un factor binario que vale 1 si el paciente es un hombre y 0 si es una mujer. Para averiguar la proporción de hombres en ese estudio basta con hacer:

```
fhs = read_csv("../datos/framingham.csv")
mean(fhs$male)
```

```
## [1] 0.4292453
```

Aproximadamente el 43% de los pacientes son hombres.

Section 4

Complementos de R: listas, bucles, funciones, datos ausentes, Rmarkdown.

Listas.

- Como referencias para este apartado puedes usar (Boehmke 2016, capítulo 11), (Matloff 2011, capítulo 4).
- A diferencia de los vectores, las listas sirven para guardar elementos heterogéneos (incluidas sublistas). La forma más sencilla de crear una lista es usando `list`:

```
(planeta = list(nombre = "Marte", exterior = TRUE,
                radio = 3389.5, satelites = list("Fobos", "Deimos")))
```

```
## $nombre
## [1] "Marte"
##
## $exterior
## [1] TRUE
##
## $radio
## [1] 3389.5
##
## $satelites
## $satelites[[1]]
## [1] "Fobos"
##
## $satelites[[2]]
## [1] "Deimos"
```

Accediendo a los elementos de una lista.

- R usa $ o doble corchete [[ ]] para identificar los elementos de la lista.

```
planeta[[1]]
```

```
## [1] "Marte"
```

```
planeta$exterior
```

```
## [1] TRUE
```

```
planeta$satelites[[1]]
```

```
## [1] "Fobos"
```

La salida es del tipo de objeto que hay en esa posición de la lista.

- Pero fíjate en la diferencia si usamos un único corchete:

```
planeta[1]
```

```
## $nombre
## [1] "Marte"
```

```
planeta["exterior"]
```

```
## $exterior
## [1] TRUE
```

En este caso la salida *siempre es una lista*.

Funciones `list`, `append` y `c`.

- Atención a esta diferencia:

```
(l1 = list("A", "B"))
```

```
## [[1]]
## [1] "A"
##
## [[2]]
## [1] "B"
```

```
(l2 = list(c("A", "B")))
```

```
## [[1]]
## [1] "A" "B"
```

La función `list` siempre crea *listas anidadas*. Por ejemplo este comando (no se muestra la salida) crea una lista con dos componentes y el primero es `l2`:

```
(l3 = list(l2, "C"))
```

- Las funciones `append` y `c` *adjuntan* elementos. Estos comandos son equivalentes:

```
l4 = append(l2, "D")
(l4 = c(l2, "D"))
```

```
## [[1]]
## [1] "A" "B"
##
## [[2]]
## [1] "D"
```

También se pueden añadir elementos por nombre, como en `planeta$distSol = 227.9`

Otras propiedades y operaciones con listas.

- La función `length` produce el número de elementos de una lista. Y con `names` se obtienen los nombres de sus elementos (si se han dado nombres).
- **Ejercicio:** Prueba a usar `names` y `length` con varias de las listas que hemos creado. Ejecuta `(sesion = sessionInfo())` para ver lo que hace esa función. Y luego explora como acceder a las componentes usandosesion$‘
- Para eliminar elementos de una lista basta con hacerlos `NULL`.

```
l4[3] = NULL
l4
```

```
## [[1]]
## [1] "A" "B"
##
## [[2]]
## [1] "D"
```

- La función `unlist` *aplana* una lista dando como resultado un vector:

```
unlist(l1)
```

```
## [1] "A" "B"
```

- **Ejercicio:** ¿qué se obtiene al aplicar `unlist` a la siguiente `lista`?
  `lista = list(letters[1:3], matrix(1:12, nrow = 3), TRUE)`.

Estructuras de control en R. Bloques if/else.

- Como referencias para este apartado puedes usar (Boehmke 2016, capítulo 19), (Matloff 2011, capítulo 7).
- **Bloques if/else.** La estructura básica de estos bloques es:

```
if (condición) {
  ...
  sentencias que se ejecutan si condicion = TRUE
  ...
}  else {
  ...
  sentencias que se ejecutan si condicion = FALSE
  ...
}
```

  Si necesitas condiciones anidadas puedes cambiar `else` por `else if` y añadir a continuación otra condición para crear un nuevo nivel de la estructura.

- La estructura `if` está pensada para ejecutarse sobre una *única* condición que produzca un *único* valor `TRUE/FALSE`. Existe también una función vectorializada, llamada `ifelse` que se puede aplicar a un vector de condiciones. Un ejemplo:

```
ifelse(((1:5) < 3), yes = "A",  no = "B")
```

```
## [1] "A" "A" "B" "B" "B"
```

Bucles for.

- El bucle for se utiliza cuando queremos repetir un bloque de comando y conocemos de antemano el número máximo de repeticiones. Su estructura básica es similar a esta:

```
for(k in valores_k) {
  ...
  cuerpo del bucle, se repite a lo sumo length(valores_k) veces
  ...
}
```

  La variable `k` (el nombre es arbitrario) es el *contador* del bucle for. El vector `valores_k` contiene los valores que toma `k` en cada iteración.

- Si en alguna iteración queremos interrumpir el bucle cuando se cumple alguna condición (y no hacer ninguna iteración más), podemos combinar `if` con la función `break`. Si lo que queremos es solamente pasar a la siguiente iteración usamos `next` en lugar de `break`.

- A menudo se usa un bucle for para *"rellenar"* un objeto como un vector o matriz. Es importante recordar que R es poco eficiente haciendo *"crecer"* estos objetos. En esos casos es mucho mejor comenzar creando el objeto completo, con todas sus posiciones, e ir asignado valores a posiciones en cada iteración (R lo inicializa a 0).

- En general **es preferible usar operaciones vectorializadas en lugar de bucles for**. Pronto aprenderemos las funciones de la familia `apply` para hacer esto.

Ejemplo de bucle for con `next` y `break`.

- El siguiente código ilustra un bucle for con el uso de next y break. Ejecútalo varias veces para ver como se comporta según los valores de sus parámetros.

```
valores = numeric(10) # Creamos un vector del tamaño previsto
for (k in 1:10){
  sorteo = sample(1:20, 1)
  print(paste0("k = ", k, ", sorteo = ", sorteo))
  if (k %in% 5:6){
    next # saltamos dos valores
  } else if (sorteo  == 1){
    print("Resultado del sorteo es 1, fin del bucle")
    break # paramos si un valor aleatorio es 1
  }
  valores[k] = sorteo # se ejecuta cuando no se cumplan las condiciones
}
valores
```

- **Ejercicio:** ¿Qué valores asigna R a los elementos de los vectores creados respectivamente con `x = logical(10)` y con `v = character(10)`?

Otros bucles: while y repeat.

- En R también existen estos dos tipos de bucles, comunes a muchos lenguajes. Conviene insistir en que suele ser más eficiente evitar el uso de bucles.
- El bucle `while` tiene esta estructura:

```
while (condición){
    ...
    cuerpo del bucle, que eventualmente debe hacer condición TRUE o usar break
    ...
}
```

- El bucle `repeat` tiene esta estructura:

```
repeat {
    ...
    cuerpo del bucle, que debe usar break
    ...
}
```

  Insistimos: a diferencia de otros lenguajes, en R un bucle `repeat` debe usar explícitamente `break` para detenerse.

- El código de este tema contiene ejemplos de bucle `while` y `repeat` con `break`, que puedes ejecutar varias veces. Observa las diferencias en el comportamiento de ambos bucles.

Funciones de R.

- Aunque R básico y todas las librerías disponibles nos ofrecen miles de funciones para las más diversas tareas, pronto llegará el día en que necesitarás escribir una función para resolver un problema específico.
- Para escribir una función de R podemos usar este esquema básico

```
nombreFuncion = function(argumento1, argumento2, ...){
  ...
  ...

  líneas de código del cuerpo de la función
  ...
  ...
}
```

Como se ve la función tiene un *nombre*, una lista de *argumentos* y un *cuerpo* que contiene las líneas de código R que se ejecutarán al llamar a la función.

Ejemplo

- Crearemos una función `genPasswd` que genere contraseñas aleatorias. Los argumentos serán la longitud de la contraseña `size` y 3 booleanos `upp`, `low` y `nmb` que sirven para incluir o no respectivamente mayúsculas, minúsculas y números. Todos ellos menos `size` tienen valores por defecto.

```
genPasswd = function(size, upp = TRUE, low = TRUE, nmb = TRUE){

  # El vector pool guarda el juego de caracteres del password
  pool = character()

  # Generamos pool según las opciones
  if(upp) pool = c(pool, LETTERS)
  if(low) pool = c(pool, letters)
  if(nmb) pool = c(pool, 0:9)

  # Sorteamos los símbolos que aparecen en el password
  passwd = sample(pool, size, replace = TRUE)
  # Y lo reducimos a un string con paste
  paste(passwd, sep = "", collapse = "")
}
```

La función se ejecuta como cualquier otra función de R (*pero cuidado:* si tratas de ejecutarla sin darle un valor a `size` habrá un error.):

```
genPasswd(size = 15)
```

```
## [1] "vwgOL4VWGKJKzbQ"
```

- **Ejercicio:** lee la ayuda de la función `paste` (y después la de `paste0`). Es una función extremadamente útil para trabajar con texto.

Acceso a las componentes de una función.

- La función `formals` permite acceder la lista de argumentos de cualquier función. Prueba a ejecutar:

```
formals(genPasswd)
```

El resultado es una *lista*. Este es un tipo de estructuras de datos muy importante que aún no hemos tratado, pero que veremos en breve.

- La función `body` permite acceder (¡y modificar!) el cuerpo de la función:

```
body(genPasswd)
```

```
## {
##     pool = character()
##     if (upp)
##         pool = c(pool, LETTERS)
##     if (low)
##         pool = c(pool, letters)
##     if (nmb)
##         pool = c(pool, 0:9)
##     passwd = sample(pool, size, replace = TRUE)
##     paste(passwd, sep = "", collapse = "")
## }
```

Observa el resultado si ahora haces

```
body(genPasswd) = "No me apetece trabajar...invéntate tú el password"
genPasswd(12)
```

Puedes leer más sobre funciones en el Capítulo 18 de (Boehmke 2016).

Manejo de datos ausentes. La función `is.na`

- Hasta ahora hemos tocado sólo tangencialmente el tema de los datos ausentes, pero es sin duda uno de los quebraderos de cabeza más habituales que te encontrarás al trabajar con un nuevo conjunto de datos.
- En R los datos ausentes se representan con el símbolo `NA`. Y disponemos de varias funciones para detectarlos. La más básica es `is.na`. Por ejemplo:

```
x = c(2, 3, -5, NA, 4, 6, NA)
is.na(x)
```

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

  Esta función es muy útil cuando se combina con otras como `which` que ya conoces o como `all` y `any`. Estas dos últimas actúan sobre un vector de booleanos y valen `TRUE` si todos o alguno, respectivamente, de los valores del vector son `TRUE`.

- Por ejemplo, podemos saber si `fhs$glucose` tiene algún valor ausente con

```
any(is.na(fhs$glucose))
```

```
## [1] TRUE
```

Más sobre datos ausentes: `complete.cases` y `na.rm`

- Una función relacionada es `complete.cases`, Aplicada a una tabla (`data.frame`) nos dirá para cada fila si esa fila tiene o no datos ausentes.

```
head(complete.cases(fhs), 17)
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [10]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
```

  El primer `FALSE` corresponde a la fila 15 de `fhs` que tiene un valor ausente en la columna `glucose`, como ya sabemos.

- La presencia de datos ausentes puede hacer que muchas funciones produzcan `NA` como resultado (o peor, que no funcionen correctamente). Por ejemplo, una media aritmética:

```
mean(fhs$glucose)
```

```
## [1] NA
```

  Muchas funciones de R disponen de un argumento `na.rm` para excluir los valores `NA` de la operación que se realice:

```
mean(fhs$glucose, na.rm = TRUE)
```

```
## [1] 81.96366
```

  Puedes encontrar más información en la Sección 7.4 de R for Data Science, la Sección 5.12 de (Peng 2015) y el Capítulo 14 de (Boehmke 2016).

Rmarkdown para la creación de documentos.

- En las sesiones del curso veremos algunos ejemplos sencillos para que puedas iniciarte en el manejo de Rmarkdown.
- Rmarkdown, creado por Yihui Xie, es una herramienta muy general para la creación de documentos, en especial documentos relacionados con el Análisis de Datos. A partir de los ficheros escritos con Rmarkdown se pueden obtener fácilmente salidas en formato pdf o HTML, pero también presentaciones, artículos e informes técnicos, entradas de blog, libros, dashboards con Shiny y la lista sigue creciendo.
  Aprovecharemos las secciones finales de nuestras sesiones para presentar algunas de estas posibilidades.
- Hay dos fuentes de información online básicas para adentrarse en las posibilidades de RMarkdown:
  Del propio Yihui Xie: R Markdown: The Definitive Guide.
  De RStudio: R Markdown: Get Started.
- Recomendamos empezar leyendo una introducción recciente a RMarkdon de T. Goicoa (Univ. Pública de Navarra).

Referencias para la sesión

Enlaces

- Código de esta sesión
- Web del libro PostData y los tutoriales asociados. Para esta sesión se recomienda el Capítulo 2.

Bibliografía

Boehmke, B. C. (2016). *Data Wrangling with R* (p. 508). Springer. https://doi.org/10.1007/978-3-319-45599-0

Matloff, N. S. (2011). *The art of R programming : tour of statistical software design* (p. 373). No Starch Press. https://doi.org/10.1080/09332480.2012.685374

Peng, R. D. (2015). *R Programming for Data Science* (p. 132). Leanpub. https://doi.org/10.1073/pnas.0703993104