Tutorial 08: Distribuciones relacionadas con la binomial.

Atención:

- Este documento pdf lleva adjuntos algunos de los ficheros de datos necesarios. Y está pensado para trabajar con él directamente en tu ordenador. Al usarlo en la pantalla, si es necesario, puedes aumentar alguna de las figuras para ver los detalles. Antes de imprimirlo, piensa si es necesario. Los árboles y nosotros te lo agradeceremos.
- Fecha: 19 de abril de 2017. Si este fichero tiene más de un año, puede resultar obsoleto. Busca si existe una versión más reciente.

Índice

1. Intervalos de confianza y contrastes de hipótesis para el parámetro p, proporción binomial

Esta sección describe como obtener intervalos de confianza y realizar contrastes de hipótesis como los que se discuten en las Secciones 8.1.1 y 8.1.2 del libro.

1.1. Intervalos de confianza para la proporción.

Nuestro primer objetivo es, por lo tanto, calcular intervalos de confianza para una variable aleatoria cuya distribución en la población es de tipo Bernouilli, con parámetro p. El intervalo de confianza, basado en la aproximación normal a la binomial aparece en la Ecuación 8.7 (pág. 280) del libro:

$$\hat{p} - z_{\alpha/2}\sqrt{\frac{\hat{p}\cdot\hat{q}}{n}} \leq p \leq \hat{p} + z_{\alpha/2}\sqrt{\frac{\hat{p}\cdot\hat{q}}{n}}.$$

Para utilizar esta fórmula es muy importante que se cumplan, a la vez, estas condiciones:

$$n > 30, \quad n\cdot\hat{p} > 5, \quad n\cdot\hat{q} > 5,$$

que, en esencia, dicen que n es bastante grande, y p y q no demasiado pequeños. Vamos a ver como calcular estos intervalos y contrastes con varios de los programas que conocemos.

Usando R.

Con R es muy fácil calcular los intervalos de confianza, usando un fichero plantilla como los que conocemos de anteriores tutoriales, para trabajar con los datos resumidos mediante n y $\hat{p}$, El fichero es este:

Aparte de este fichero plantilla, en la Sección 1.2 vamos a describir la función `prop.test`. Aunque esa función está diseñada principalmente para realizar contrastes de hipótesis, como beneficio añadido también ofrece una forma alternativa de calcular estos intervalos de confianza. Veamos un ejemplo.

En una página web ha aparecido recientemente una comparativa entre baterías externas de respaldo para teléfonos móviles y aparatos similares. Además, se realizó una encuesta para conocer cuál era la marca preferida de los lectores de esa web. De un total de 2499 opiniones, la marca ganadora obtuvo 1004 votos. Vamos a calcular un intervalo de confianza al 95 % para el porcentaje de lectores que prefieren esa marca.

Para hacer esto usamos este comando en R:

```
prop.test(x=1004, n=2499, conf.level=0.95, correct=FALSE)

##
##  1-sample proportions test without continuity correction
##
## data:  1004 out of 2499, null probability 0.5
## X-squared = 96.5, df = 1, p-value <2e-16
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.38270 0.42112
## sample estimates:
##       p
## 0.40176
```

Como puedes ver, la función está diseñada para contrates de hipótesis, aunque produce el intervalo de confianza como subproducto. Vamos a explicar brevemente los argumentos que hemos usado para esta función. El argumento `x` se refiere al número de éxitos que aparecen en la muestra, mientras que `n` es el tamaño de la muestra. El argumento `conf.level` es autoexplicativo, así que sólo nos queda por comentar porque hemos incluido `correct=FALSE`. Si consultas la ayuda de `prop.test`, verás que, con esa opción, estamos desactivando la *corrección de continuidad de Yates*. Se trata de un ajuste similar al que hemos discutido en la Sección 5.6.2 (pág. 179) del libro. Si se usa esta corrección se obtienen intervalos de confianza más precisos. Nosotros no lo hemos hecho porque queremos mantener la discusión a un nivel muy simple, aunque sea a costa de sacrificar un poco de precisión. Por eso, para obtener los resultados habituales de los problemas *de libro de texto*, debemos usar `correct=FALSE`.

Usando GeoGebra.

En GeoGebra dispones de la función `IntervaloProporciónZ` para realizar este tipo de cálculos. Su uso es muy parecido al de las funciones que hemos visto en los intervalos para la media, así que dejaremos que lo practiques mediante ejercicios.

Ejercicio 1.

1. *Usa el fichero plantilla de R que hemos incluido en esta sección para comprobar las cuentas del Ejemplo 8.1.2 (pág. 280 del libro), el de los araos embridados.*
2. *Haz lo mismo con* `prop.test`*. Y calcula un intervalo de confianza para los datos de araos embridados del año 2008 (*138 *embridados,* 270 *no embridados).*
3. *Calcula también el intervalo del Ejemplo 8.1.2 usando GeoGebra.*

Soluciones en la página 25. □

Usando Wolfram Alpha.

Con Wolfram Alpha también es muy fácil calcular estos intervalos. En el Ejemplo 8.1.2 de los araos, era $n = 456$, y el número de éxitos (araos embridados) era de 139. Para calcular un intervalo de confianza para p a partir de estos datos, en Wolfram Alpha podemos escribir:

```
binomial confidence interval n=456, p-hat=139/456
```

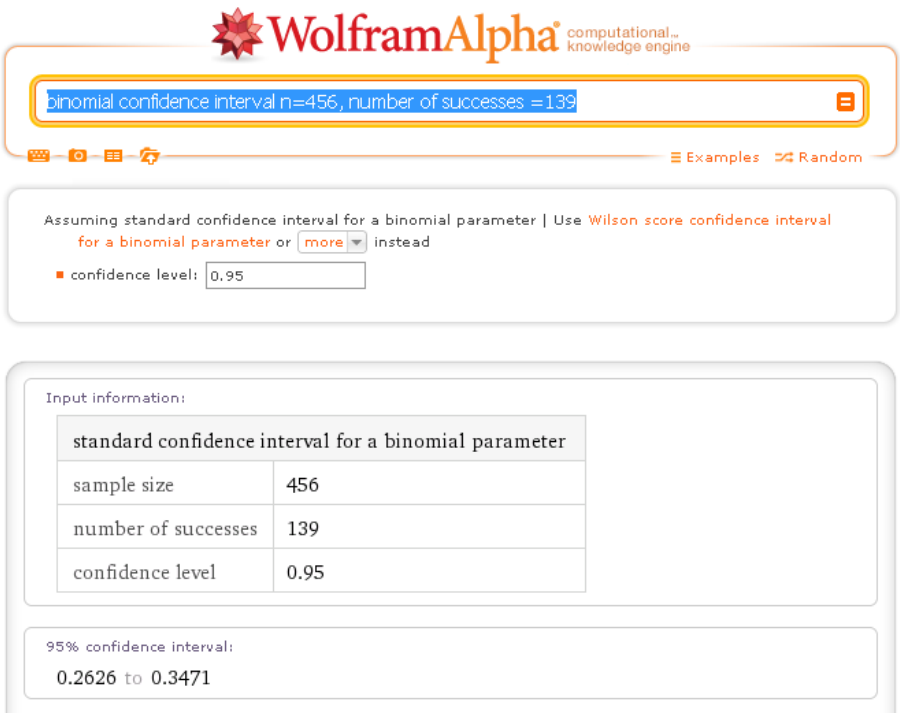
O también

```
binomial confidence interval n=456, p-hat=0.3048
```

o incluso, usando el número de éxitos (*successes*):

```
binomial confidence interval n=456, number of successes =139
```

Mostramos, en la siguiente figura, (parte de) el resultado del último de estos comandos:



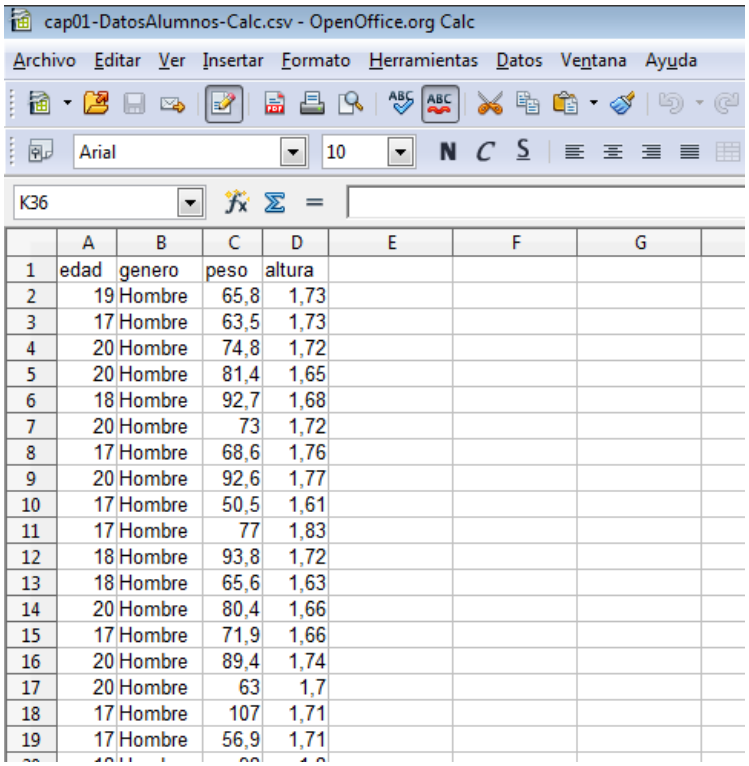
Las proporciones como caso especial de variables cualitativas (factores).

Tal vez te preguntes por qué en este caso no hemos incluido un fichero para trabajar con los datos de la muestra "en bruto" (*raw data*, en inglés). ¿Cómo sería el fichero del Ejemplo 8.1.2 del libro (pág. 280), el de los araos embridados? Tendría que ser una lista con 456 valores como estos:

```
embridado
noEmbridado
noEmbridado
embridado
noEmbridado
embridado
...
```

Las proporciones son variables de tipo Bernouilli y, por tanto, son simplemente respuestas de tipo sí/no. Por eso no hemos querido, artificiosamente, hacerte trabajar con ficheros que contengan 456 filas de síes y noes. Es más fácil darte el resumen, como hemos hecho en ese ejemplo, y decir *"hay 139 araos embridados y 317 no embridados"*.

¡Pero cuidado! Eso no significa que no te vayas a encontrar con ficheros que contengan información sobre variables de tipo Bernouilli. Al contrario: es extremadamente frecuente. Porque, como hemos dicho ya varias veces a lo largo del curso, lo habitual es que se midan a la vez varias características en una misma observación. En la Figura 1.1 del libro (pág. 8), que reproducimos aquí, puedes ver algunas líneas del fichero `cap01-DatosAlumnos.csv`,



	A	B	C	D	E	F	G
1	edad	genero	peso	altura			
2	19	Hombre	65,8	1,73			
3	17	Hombre	63,5	1,73			
4	20	Hombre	74,8	1,72			
5	20	Hombre	81,4	1,65			
6	18	Hombre	92,7	1,68			
7	20	Hombre	73	1,72			
8	17	Hombre	68,6	1,76			
9	20	Hombre	92,6	1,77			
10	17	Hombre	50,5	1,61			
11	17	Hombre	77	1,83			
12	18	Hombre	93,8	1,72			
13	18	Hombre	65,6	1,63			
14	20	Hombre	80,4	1,66			
15	17	Hombre	71,9	1,66			
16	20	Hombre	89,4	1,74			
17	20	Hombre	63	1,7			
18	17	Hombre	107	1,71			
19	17	Hombre	56,9	1,71			

La variable `genero` de este fichero es una variable de Bernouilli, con sólo dos posibles variables y su papel en este conjunto de datos es una muestra típica de las situaciones en las que intervienen esas variables, que sirven para *clasificar* los datos en subgrupos. Por supuesto, todas las variables cualitativas (factores) sirven para esto, pero las variables tipo Bernouilli son especialmente sencillas, porque definen clasificaciones con sólo dos posibles clases.

En R, la forma natural de manejar un conjunto de datos como el de esta figura es mediante los `data.frames` que hemos visto en el Tutorial04. Un `data.frame` puede incluir una o varias variables de tipo Bernouilli en sus columnas. Y, en ese caso, si deseas hacer un intervalo de confianza para una de esas variables, sí que tendrás que trabajar a partir de los datos en bruto y hacer una tabla de frecuencias para obtener las proporciones muestrales. En la sección 4 de este tutorial vamos a avanzar en nuestra comprensión del trabajo con factores dentro de R.

1.2. Contrastes de hipótesis para la proporción.

Para los contrastes de hipótesis podemos repetir el esquema básico que hemos visto para los intervalos.

Usando R.

Empezamos con un fichero plantilla de R para este tipo de contrastes:

Tut08-Contraste-Proporcion-UsandoZ.R

Ejercicio 2.

1. *Usa ese fichero para comprobar las cuentas del contraste de hipótesis que aparece en el Ejemplo 8.1.3 (pág. 281).*

2. *Según el* Barómetro *del CIS de enero del 2013, en una muestra de 2452 personas residentes en España, 568 de ellas se declararon* no creyentes. *Usa estos datos para contrastar, al 95 %, la hipótesis de que la proporción de* no creyentes *es inferior al 25 %.*

Soluciones en la página 28. □

Como ya hemos adelantado, la función `prop.test` está pensada para este tipo de contrastes. Como sucedía con `t.test`, `var.test` y otras funciones similares, el argumento `alternative` nos permite

seleccionar un contraste bilateral o unilateral y, en este segundo caso, si usamos la cola derecha o izquierda. Por ejemplo, el contraste del Ejemplo 8.1.3 (pág. 281) se obtiene así con `prop.test`:

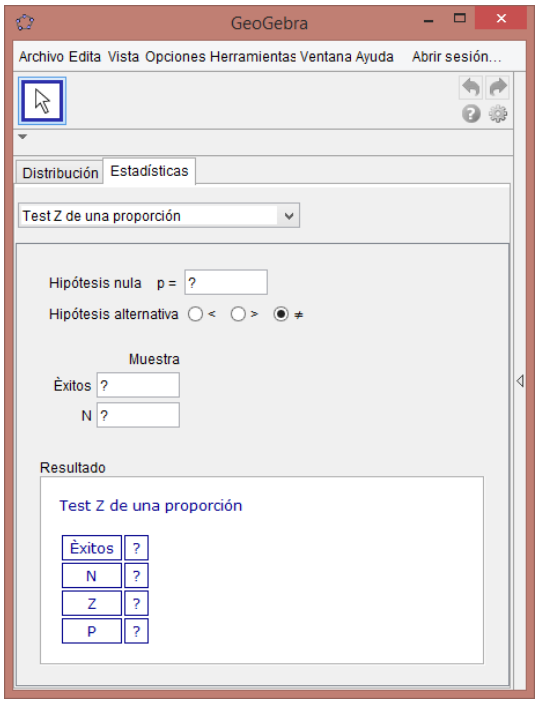
```
(contraste = prop.test(x=39, n=105, alternative = "greater",
                       p=0.35, conf.level = 0.95, correct = FALSE))

##
##  1-sample proportions test without continuity correction
##
## data:  39 out of 105, null probability 0.35
## X-squared = 0.212, df = 1, p-value = 0.32
## alternative hypothesis: true p is greater than 0.35
## 95 percent confidence interval:
##  0.29801 1.00000
## sample estimates:
##       p
## 0.37143
```

Recuerda que `greater` se usa cuando $H_a = \{p > p_0\}$, que `less` se usa con $H_a = \{p < p_0\}$ y que `two.sided` se usa con $H_a = \{p \neq p_0\}$.

Por razones técnicas, que quedarán más claras en el Capítulo 12 del libro, el estadístico que obtenemos en `prop.test`, y que aparece como tt X-squared, es en realidad el cuadrado del que obtendrás si usas el fichero plantilla que hemos visto antes. Además, hemos usado la opción `correct=FALSE` por las mismas razones que discutimos en el caso de los intervalos de confianza.

Usando GeoGebra.

En la *Calculadora de Probabilidades* de GeoGebra tenemos la posibilidad de realizar uno de estos contrastes, dentro de la pestaña *Estadísticas*, como se muestra en la figura:



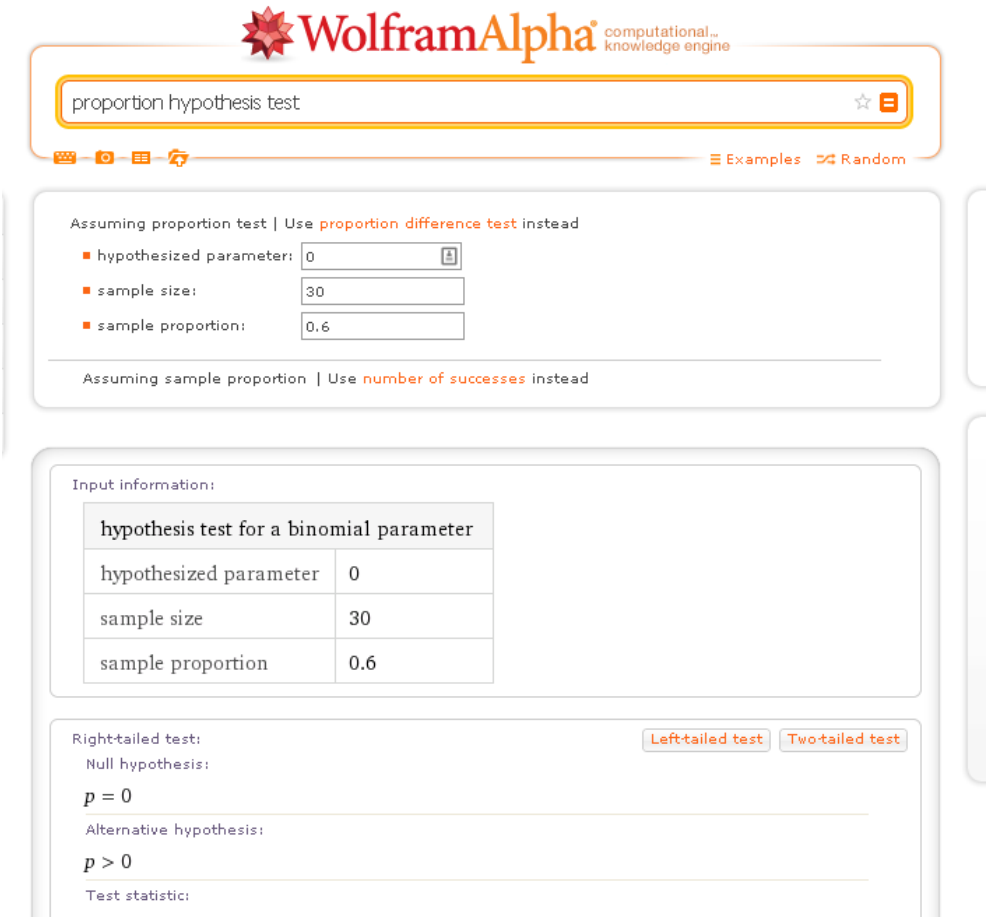
Ejercicio 3. *Usa GeoGebra para repetir el Ejercicio 2. Solución en la página 31.* □

Usando Wolfram Alpha.

En Wolfram Alpha podemos usar el comando

`proportion hypothesis test`

para llegar a un interfaz en el que introducir los valores necesarios para el contraste, que se muestra en esta figura:



2. Intervalos de confianza exactos para la proporción. Método de Clopper-Pearson.

Atención: Esta sección es opcional en una primera lectura.

La Sección 8.1.3 del libro (pág. 282) describe él método exacto de Clopper-Pearson para hacer inferencia sobre proporciones en casos que no cubre la aproximación por la distribución normal. En esta breve sección sólo queremos comentar que R dispone de la función `binom.test`, que permite obtener este tipo de resultados de una forma muy cómoda.

La forma de usar `binom.test` nos debe resultar muy familiar a estas alturas, porque se comporta de manera similar a `t.test` y funciones semejantes. A modo de ilustración, para obtener los resultados del Ejemplo 8.1.4 del libro (pág. 282) usaríamos `binom.test` de esta manera:

```
binom.test(x = 2, n = 15, p = 0.1, alternative = "greater", conf.level = 0.95)

##
##  Exact binomial test
##
## data:  2 and 15
## number of successes = 2, number of trials = 15, p-value = 0.45
## alternative hypothesis: true probability of success is greater than 0.1
## 95 percent confidence interval:
##  0.024226 1.000000
## sample estimates:
```

```
## probability of success
##               0.13333
```

Como ves, el p-valor coincide con el que aparece en el Ejemplo 8.1.4. Además, como subproducto del cálculo obtenemos un intervalo de confianza para la proporción. Pero ten en cuenta que como la hipótesis alternativa es unilateral, ese intervalo es también unilateral. Si se desea un intervalo de confianza bilateral, debemos usar la opción `two.sided` en `binom.test`. Así, el intervalo de confianza del Ejemplo 8.1.5 del libro (pág. 284) se obtiene con:

```
binom.test(x = 2, n = 15, p = 0.1, alternative = "two.sided", conf.level = 0.95)

##
##  Exact binomial test
##
## data:  2 and 15
## number of successes = 2, number of trials = 15, p-value = 0.66
## alternative hypothesis: true probability of success is not equal to 0.1
## 95 percent confidence interval:
##  0.016576 0.404603
## sample estimates:
## probability of success
##               0.13333
```

Naturalmente, también es posible construir estos intervalos directamente, usando `pbinom` o una herramienta análoga, y siguiendo los pasos que se describen el el libro. Eso abre la puerta al cálculo de estos contrastes e intervalos usando otros programas como GeoGebra, Wolfram Alpha o, en general, cualquier programa que permita calcular probabilidades binomiales. No nos vamos a extender más aquí sobre el tema.

3. La distribución de Poisson

En la sección 8.2 del libro (pág. 286) se presenta la distribución de Poisson, y se explica el papel que juega esta distribución como una alternativa para aquellos casos en que tendríamos que trabajar con una distribución binomial con n grande pero a la vez p pequeño. Aquí vamos a ver algunas herramientas computacionales útiles para trabajar con esa distribución.

3.1. La distribución de Poisson en R.

Para trabajar con la distribución de Poisson disponemos en R de cuatro funciones análogas a las que vimos en el caso de la distribución binomial:

`dpois` `ppois` `qpois` `rpois`

Suponemos que, a estas alturas, es fácil intuir la finalidad y funcionamiento de todas ellas. Vamos a comentarlas por tanto brevemente, deteniéndonos sólo en los aspectos que pueden plantear alguna dificultad:

- `dpois` es la función de densidad, y por lo tanto, si X es una variable de tipo Pois(λ), pongamos por ejemplo con $\lambda = 5$, entonces para calcular:

  $$P(X = 7),$$

  ejecutaríamos este comando:

  ```
  dpois(7, lambda=5)

  ## [1] 0.10444
  ```

- `ppois` nos permite calcular la probabilidad de la cola izquierda de una distribución de Poisson. Y como en el caso de la binomial, al tratarse de una **distribución discreta**, tenemos que ser cuidadosos. La cola izquierda aquí se define usando la desigualdad $\leq$. Es decir, que si, como antes, $\lambda = 5$, entonces para calcular

$$P(X \leq 4),$$

debemos ejecutar:

```
ppois(4, lambda=5)

## [1] 0.44049
```

Mientras que, si lo que queremos es calcular la probabilidad con una desigualdad estricta, debemos convertirla primero en no estricta, como en este ejemplo:

$$P(X < 4) = P(X \leq 3),$$

y ahora calcular

```
ppois(3, lambda=5)

## [1] 0.26503
```

Para calcular la probabilidad de una cola derecha hay que usar, como de costumbre `1-ppois` o la opción `lower.tail=FALSE`, con las mismas precauciones al tratarse de una distribución discreta.

- `qpois` sirve para resolver problemas inversos de probabilidad, usando la cola izquierda y una desigualdad no estricta $\leq$. Además, debe tenerse en cuenta que, puesto que se trata de una variable discreta, los problemas inversos se definen, en general, como desigualdades sobre la probabilidad. Es decir, que por ejemplo, si X es de tipo Pois(15) (es decir, $\lambda = 15$), y buscamos el valor a que tiene la propiedad de que:

$$P(X \leq a) = 0.35$$

Es muy posible que ninguno de los posibles valores de X (que son los números naturales $0, 1, 2, \ldots$) tenga esa propiedad. Si, en cambio, preguntamos por el primer valor (el valor más pequeño) para el que se cumple:

$$P(X \leq a) \geq 0.35$$

entonces la pregunta está bien definida, y la respuesta se obtiene con

```
qpois(0.35, lambda=15)

## [1] 13
```

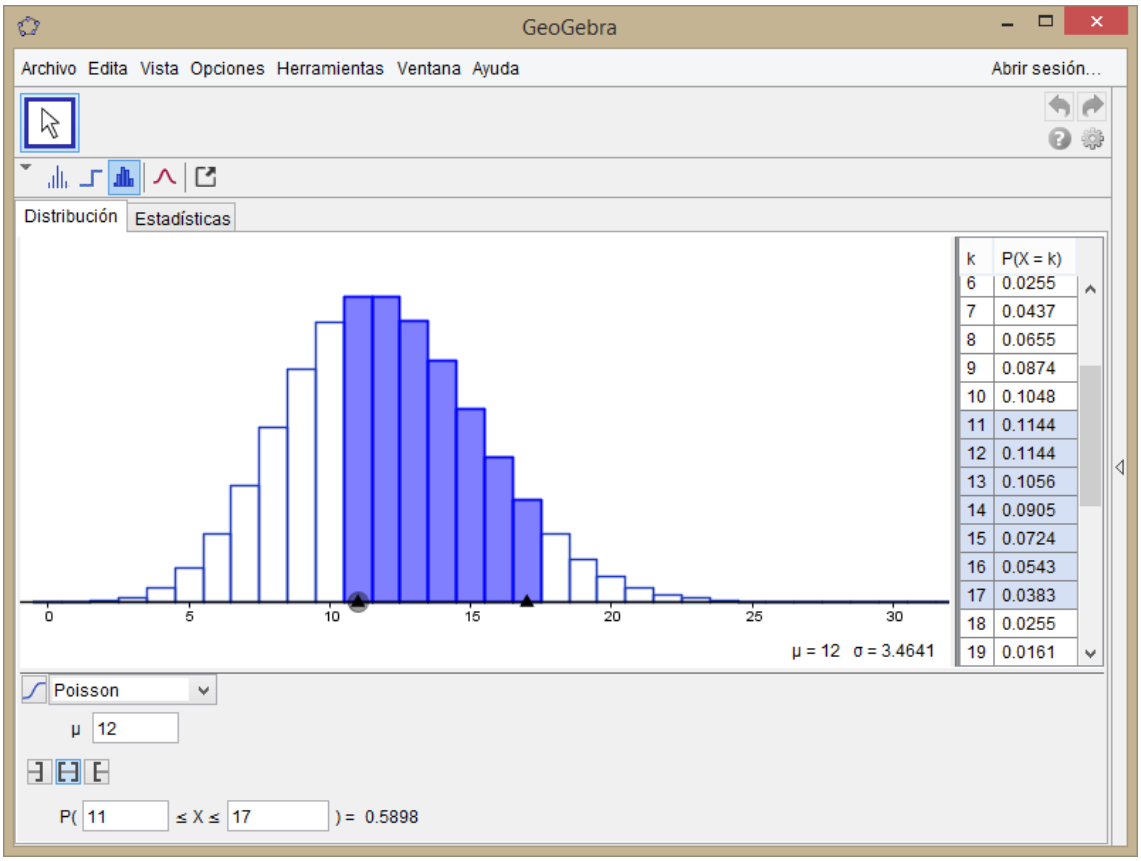
Y puedes comprobarlo calculando la probabilidad acumulada para ese valor (el 13) y para los valores adyacentes 12 y 14.

```
ppois(12:14, lambda=15)

## [1] 0.26761 0.36322 0.46565
```

- `rpois` sirve para obtener valores aleatorios de una distribución de Poisson y se usa, sobre todo, para realizar simulaciones.

Usando GeoGebra.

En GeoGebra puedes usar la calculadora de Probabilidades en la forma habitual para resolver problemas de probabilidad directos e inversos que involucren a la distribución de Poisson. La siguiente figura muestra la interfaz de la calculadora mientras resolvemos uno de esos problemas.



Aparte de esto, si utilizas la *Línea de entrada* o el panel de *Cálculo Simbólico* puedes usar, en cualquiera de ellos, los comandos:

```
Poisson[ <Media>, <Valor>, <Acumulada o no (true/false)> ]
PoissonInversa[ <Media>, <Probabilidad> ]
PoissonAleatoria[ <Media> ]
```

La `media` es el valor de λ, claro. El comando `Poisson` condensa las funcionalidades que en R se obtienen con `dpois` y `ppois`. Debes usar la opción *Acumulada* para elegir:

- entre la función de distribución (como `ppois` en R) cuando *Acumulada* es `true`
- y la función de densidad (como `dpois` en R) cuando es `false`.

`PoissonInversa` se comporta de forma similar a `qpois` (también con la cola izquierda), mientras que `PoissonAleatoria` devuelve un único valor aleatorio de la distribución de Poisson.

Usando Wolfram Alpha.

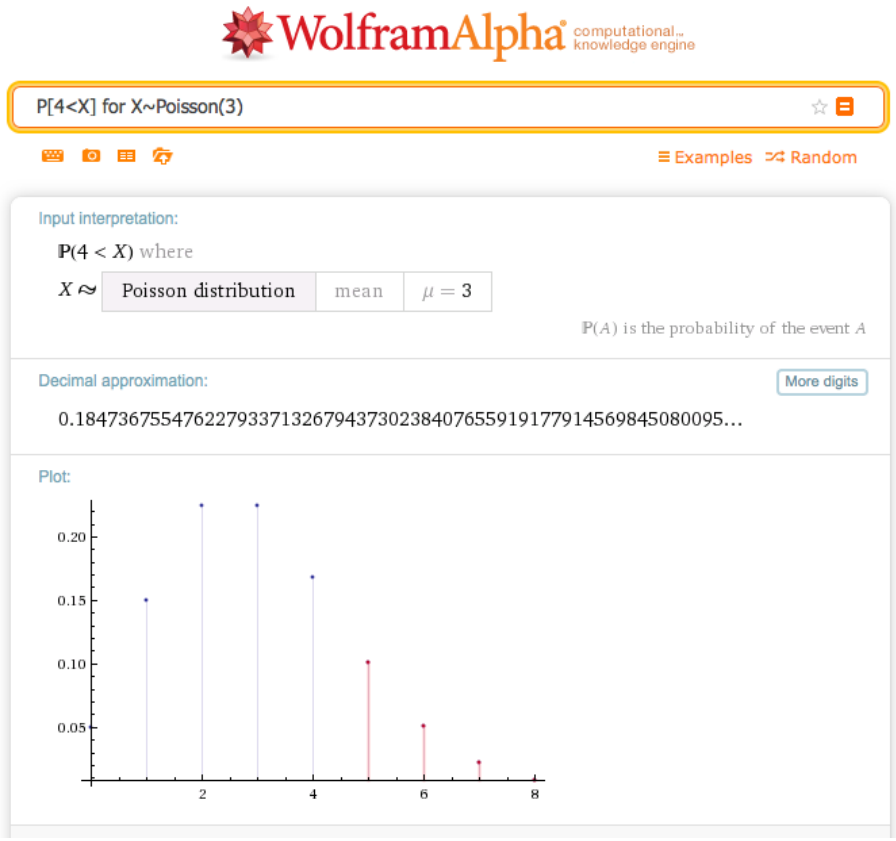
Para realizar cálculos de probabilidad directa con la distribución de Poisson, podemos usar comandos como este:

```
P[4<X] for X~Poisson(3)
```

cuyo equivalente en R es:

```r
1 - ppois(4, lambda=3)

## [1] 0.18474
```

El resultado en Wolfram Alpha se muestra en la siguiente figura:



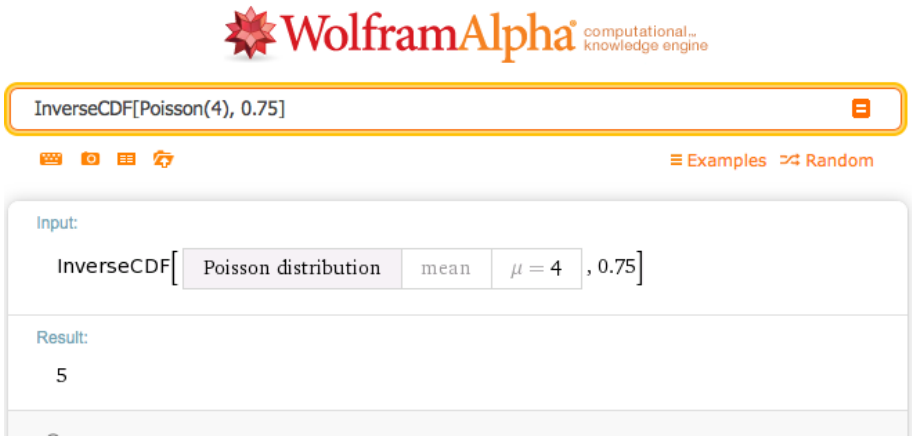
Mientras que

```
InverseCDF[Poisson(4), 0.75]
```

es el equivalente en Wolfram Alpha del comando de R

```r
qpois(0.75, lambda=4)

## [1] 5
```

Aquí `CDF` es una abreviatura de *cumulative distribution function* o *función de distribución acumulada*, lo que nosotros llamamos simplemente *función de distribución*.



Ejercicio 4.

1. *Comprueba los cálculos de probabilidad del Ejemplo 8.2.2 del libro (pág. 292) usando R, GeoGebra y Wolfram Alpha.*
2. *Comprueba también los cálculos de probabilidad del Ejemplo 8.2.3 del libro (pág. 292). En este caso, te recomendamos que uses R.*

Soluciones en la página 32. □

3.2. Procesos de Poisson.

Opcional: esta sección puede omitirse en una primera lectura.

En esta sección nos vamos a limitar a incluir el código en R que te va a permitir reconstruir cálculos como los de la Sección 8.2.2 del libro (pág. 287).

Puesto que vamos a simular algunos valores, empezamos fijando la semilla del generador de números aleatorios de R.

```
set.seed(2013)
```

A continuación introducimos los datos del INE sobre muertes por infarto en España en el año 2011, calculamos la tasa por habitante y la usamos para *estimar* el número de muertes en Madrid.

```
poblacionPais = 47190493
muertesInfarto = 18101

(probAnualMuerteInfarto = muertesInfarto/poblacionPais  )

## [1] 0.00038357

poblacionMadrid = 6489680
(muertosAnualesInfartoMadrid = poblacionMadrid * probAnualMuerteInfarto)

## [1] 2489.3
```

A continuación empecemos con la primera simulación que se describe en el libro. Primero calculamos la probabilidad de que un madrileño cualquiera muera de infarto un día cualquiera de 2011:

```
Dias = 365
(probDiariaMuerteInfarto = probAnualMuerteInfarto/365)

## [1] 0.0000010509
```

Naturalmente, esta simulación es una simplificación poco realista. Estamos suponiendo que esa probabilidad es la misma para un saludable atleta de 20 años que para un oficinista sedentario de 50 años con antecedentes familiares de enfermades coronarias...

Pero en aras del modelo ignoremos por el momento esa falta de realismo y sigamos adelante. Vamos a usar esa probabilidad para simular el número de madrileños que mueren en cada uno de los 365 días del año. Lo haremos calcuando 365 valores de la correspondiente binomial, una binomial para cada día del año. Cada una de esas binomiales es $B(n,p)$ siendo n el número de habitantes de Madrid y p la probabilidad diaria de muerte por infarto para cada uno de ellos. Así que la simulación es:

```
recuentoAnual = numeric(Dias)
p = probDiariaMuerteInfarto
for(i in 1:365){
  (casosDia = rbinom(1, size=poblacionMadrid, prob=p))
  recuentoAnual[i] = casosDia
}
```

Y en esta simulación (distinta de la que aparece en el libro) obtenemos esta tabla de frecuencias:

```
table(recuentoAnual)

## recuentoAnual
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14
##  5  9 24 34 50 49 60 46 30 29 16  8  4  1
```

La primera fila indica el número de fallecimientos por día y la segunda el número de días del año en los que se produjo esa cifra. Como ves el número más frecuente de muertes por infarto y por día es 7. En esta simulación ese número de muertes ha ocurrido en 60 de los 365 días posibles. Pero también hay 5 días con una única muerte y un día con 14 muertes. El número total de muertes en todo el año y para esta simulación asciende a:

```
sum(recuentoAnual)

## [1] 2461
```

¡No nos cansamos de repetir que es una simulación y que no pretende reproducir el número real observado ese año!

Para seguir adelante, en lugar de días usamos horas. El número de horas en un año es

```
(Horas = Dias * 24)

## [1] 8760
```

y la probabilidad de muerte individual por hora para un madrileño se calcula así:

```
(probHoraMuerteInfarto = probDiariaMuerteInfarto / 24)

## [1] 4.3787e-08
```

Y ahora hacemos otra nueva simulación, esta vez con 8760 binomiales (una por hora), cada una de ellas con n igual al número de habitantes de Madrid pero ahora con p igual a la probabilidad *horaria* de muerte por infarto para cada madrileño.

```
recuentoAnual=numeric(Horas)
(p=probHoraMuerteInfarto)

## [1] 4.3787e-08

for(i in 1:Horas){
  (casosHora=rbinom(1,size=poblacionMadrid,prob=p))
  recuentoAnual[i]=casosHora
}
```

La correspondiente tabla y el total de muertes en esta simulación son:

```
table(recuentoAnual)

## recuentoAnual
##    0    1    2    3    4
## 6602 1855  274   27    2

sum(recuentoAnual)

## [1] 2492
```

Como cabía esperar, puesto que hay más horas en un año que madrileños muertos de infarto en el año 2011, una gran mayoría de las horas están libre de muertes. Concretamente 6602 de las 8760 horas posibles, en esta simulación. Pero todavía, como ves hay dos horas concretas en las que coinciden las muertes de hasta cuatro personas (para averiguar cuales fueron esas horas fatídicas puedes ejecutar `which(recuentoAnual == 4)`. Por supuesto, luego tendrás que traducir esa información a meses, días y horas para situarlas en el calendario, pero aquí no nos vamos a enredar en esta cuenta).

Ahora ya debería estar claro cómo obtener la correspondiente simulación usando minutos en lugar de horas, o segundos en lugar de minutos, etcétera. Y también debería empezar a resultar evidente que la probabilidad de que dos sucesos (dos muertes por infarto, en este ejemplo) coincidan en el mismo minuto es mucho más pequeña que la de que coincidan en la misma hora (por supuesto, también hay 60 veces más segundos que horas). Y que, a su vez, la probabilidad de que dos muertes coincidan en el mismo segundo es todavia mucho más pequeña, etcétera.

Otra cuestión, distinta, es si tiene sentido decir que una muerte por infarto ocurre en un segundo concreto. Pero en las muchas simplificaciones que hemos impuesto en este modelo estamos asumiendo que podemos atribuir esos sucesos a intervalos arbitrariamente pequeños de tiempo.

3.3. Media y varianza de una variable de Poisson.

Sólo queremos aprovechar que conoces Wolfram Alpha para animarte a que lo uses para comprobar la suma infinita (serie) que hemos usado para calcular la media de una variable de Poisson. Prueba a ejecutar este comando en Wolfram Alpha:

```
sum(exp(-lambda) * k * lambda^k / k!) from k=0 to oo
```

Y obtendrás λ como resultado, confirmando que si $X \sim \text{Pois}(\lambda)$, entonces $\mu_X = \lambda$.

Ejercicio 5. *Usa Wolfram Alpha para comprobar que la varianza también es igual a λ.* *Solución en la página 33.* □

3.4. Inferencia en la distribución de Poisson.

Queremos ocuparnos ahora de los resultados sobre inferencia exacta en la distribución de Poisson que hemos visto en la Sección 8.2.3 del libro (pág. 295). En este apartado nos vamos a centrar en R, porque ninguno de los otros programas que usamos habitualmente ofrece facilidades comparables. En R, no obstante, disponemos de la función `poisson.test`, análoga a `t.test` y a muchas otras que ya hemos visto. Con esa función el contraste del Ejemplo 8.2.4 del libro (pág. 295) se obtiene mediante:

```
poisson.test(11, T=1, r=7, alternative="greater", conf.level=0.95)

##
##  Exact Poisson test
##
## data:  11 time base: 1
## number of events = 11, time base = 1, p-value = 0.099
## alternative hypothesis: true event rate is greater than 7
## 95 percent confidence interval:
##  6.169   Inf
## sample estimates:
## event rate
##         11
```

El único argumento que necesita explicación es `T=1`. Con ese argumento le estamos diciendo a R que 7 es el número de sucesos esperados en $T = 1$ unidades de tiempo. Este argumento se usa porque a veces tenemos, por ejemplo, el número de sucesos por hora (60 minutos), y queremos hacer un contraste con datos que representan los sucesos medidos en un cierto número de minutos. Por ejemplo, para ayudarte a ver claro que es un cambio de escala, si en el Ejemplo 8.2.4 del libro

estamos midiendo el número de éxitos por hora (se observaron 11 éxitos en una hora), entonces el cálculo del contraste se puede realizar así:

```
poisson.test(11, T=60, r=7/60, alternative="greater", conf.level=0.95)
```

```
##
##  Exact Poisson test
##
## data:  11 time base: 60
## number of events = 11, time base = 60, p-value = 0.099
## alternative hypothesis: true event rate is greater than 0.11667
## 95 percent confidence interval:
##  0.10282     Inf
## sample estimates:
## event rate
##    0.18333
```

Aquí `r = 7/60` representa la tasa de éxitos por minuto, $T = 60$ indica que el periodo de medición han sido 60 minutos y que se han observado 11 éxitos en esos 60 minutos. Simplemente estamos traduciendo los mismos datos muestrales de horas a minutos, pero como los datos son los mismos y la hipótesis nula es la misma, el p-valor es el mismo. Por contra, si en otro experimento sólo hubiéramos observado durante media hora, en la que hemos medido 5 éxitos, entonces para hacer el contraste **con la misma hipótesis nula** de 7 éxitos por hora, escribiríamos:

```
poisson.test(5, T=30, r=7/60, alternative="greater", conf.level=0.95)
```

```
##
##  Exact Poisson test
##
## data:  5 time base: 30
## number of events = 5, time base = 30, p-value = 0.27
## alternative hypothesis: true event rate is greater than 0.11667
## 95 percent confidence interval:
##  0.065672      Inf
## sample estimates:
## event rate
##    0.16667
```

Este resultado corresponde a una muestra distinta y por tanto el p-valor es diferente.

Habrás observado, en estos ejemplos, que la función `poisson.test` se puede usar para intervalos de confianza. Sin extendernos mucho más, recuerda que para hacer esto, es necesario seleccionar un contraste bilateral. Para obtener el intervalo del Ejemplo 8.2.5 del libro (pág. 296) usamos así la función:

```
poisson.test(11, T=1, r=7, alternative="two.sided", conf.level=0.95)
```

```
##
##  Exact Poisson test
##
## data:  11 time base: 1
## number of events = 11, time base = 1, p-value = 0.13
## alternative hypothesis: true event rate is not equal to 7
## 95 percent confidence interval:
##   5.4912 19.6820
## sample estimates:
## event rate
##         11
```

4. Variables cualitativas (factores) en R

Opcional: esta sección puede omitirse en una primera lectura.

En los últimos capítulos del curso, y en los tutoriales precedentes, nuestro trabajo se ha centrado en el análisis de valores de variables cuantitativas; es decir, números. Pero sabemos que a veces es necesario trabajar con variables cualitativas, que también hemos llamado `factores` (y sus valores se llaman `niveles`). Y de hecho esas variables van a tener un papel protagonista en varios de los capítulos de la cuarta parte del curso.

Recuerda que una variable cualitativa se usa para establecer clasificaciones nominales en los datos. Por ejemplo, la clasificación en *hombre* o *mujer* entre los pacientes que siguen un cierto tratamiento. O la clasificación taxonómica por especies.

Es cierto que siempre podríamos codificar las variables cualitativas mediante números. Pero no es menos cierto que lo más cómodo (y prudente) es poder utilizar nombres como valores de las variables. Ya hemos visto (en la Sección 2 del Tutorial04) que R nos permite crear un cierto tipo de valores, concretamente los valores de tipo `character`, que son simplemente palabras o frases entrecomilladas (en general las llamamos cadenas de texto). Recomendamos una relectura rápida de esa Sección antes de seguir adelante.

Por ejemplo, este vector podría representar el género de los diez pacientes que se han sometido a un cierto tratamiento:

```
pacientesPorGenero=c("hombre", "mujer", "mujer", "hombre", "mujer", "hombre",
    "hombre", "hombre", "mujer", "mujer", "mujer")
class(pacientesPorGenero)

## [1] "character"
```

Como ves, los elementos del vector `pacientesPorGenero` son, para R, de tipo `character`. Ese es el tipo de dato que se utiliza en R para representar los valores de una variable cualitativa.

Naturalmente, si intentas realizar operaciones numéricas con ese vector, como estas,

```
pacientesPorGenero^2

## Error in pacientesPorGenero^2: argumento no-numérico para operador binario

mean(pacientesPorGenero)

## Warning in mean.default(pacientesPorGenero): argument is not numeric or logical:
returning NA

## [1] NA
```

R te obsequia con un surtido de insultos más o menos ofensivos en la consola de comandos. En el segundo caso (el intento fallido de calcular la media), el valor que devuelve R es interesante: se obtiene `NA`, que es el valor que R devuelve cuando un resultado numérico es imposible de calcular, o no está disponible (*Not Available*, de ahí el nombre).

Queremos recordar también que en el Tutorial04 vimos que los `data.frames` de R son las estructuras de datos adecuadas para representar tablas en las que se mezclan datos cuantitativos y cualitativos (las matrices, sin embargo, almacenan datos que son todos del mismo tipo).

4.1. Factores.

Una situación frecuente en Estadística es esta: hemos medido una serie de valores de una variable cuantitativa X (es decir, los valores de X son números), pero esos valores aparecen agrupados de manera natural. Por ejemplo, cuando estamos midiendo la respuesta X (un número) de una serie de pacientes frente a varios tratamientos, es evidente que lo natural es agrupar los resultados según

el tratamiento empleado. Una manera de hacer esto, en R, es usar un `data.frame` que contenga los valores de X, junto con los valores de una variable que indique el tipo de tratamiento empleado. Vamos a llamar

$$T_1, T_2, \ldots, T_k$$

a los distintos tratamientos. La terminología estándar en Estadística (que ya vimos en la Sección 1.1.1, pág. 5 del libro) consiste en decir que el tratamiento T es un factor, y que sus valores $T_1, \ldots, T_k$ son los niveles del factor. Podríamos entonces pensar en guardar los resultados en un fichero de datos como el fichero adjunto,

Tut08-Tratamiento.csv

Abre el fichero primero en un editor de texto (como el *Bloc de Notas*, en Windows), para hacerte una idea de su estructura.

Podríamos trabajar directamente con los nombres de los tratamientos, usando en R variables de tipo `character` para los factores. Pero eso tendría un impacto negativo en el rendimiento de R, porque manejar esas cadenas de caracteres consume muchos recursos de memoria y tiempo del procesador. Y, dado que los factores son omnipresentes en Estadística, los creadores de R han optado por incluir un tipo especial de datos, el tipo `factor`, para representar los factores y sus niveles.

De hecho, si leemos un fichero como el anterior usando `read.table` (que también vimos en el Tutorial04), el comportamiento por defecto de R es convertir las variables cualitativas en factores. Vamos a ver esto en funcionamiento. Asegúrate de haber seleccionado el directorio de trabajo y ejecuta estos comandos:

```
experimento = read.table(file="../datos/Tut08-Tratamiento.csv",
    header = TRUE, sep=" ")

class(experimento$Respuesta)

## [1] "numeric"

class(experimento$Tratamiento)

## [1] "factor"
```

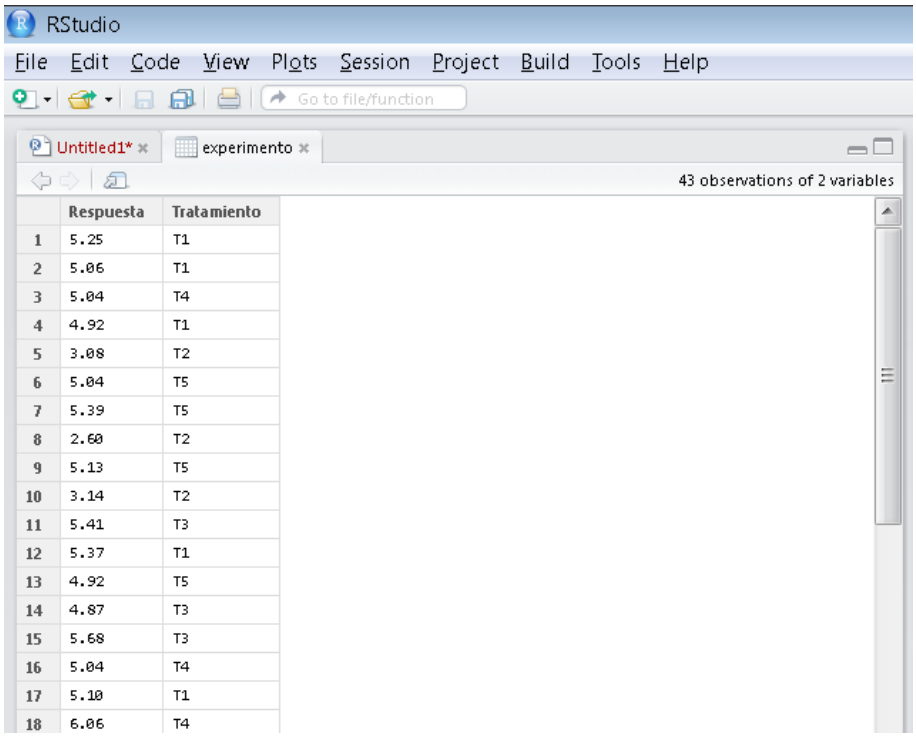
Lo que hemos hecho aquí es guardar el contenido del fichero en un `data.frame` llamado `experimento`, con dos campos (cuyos tipos hemos mostrado usando `class`):

- `experimento$Respuesta`, de tipo numérico que almacena los valores de la variable X (la respuesta individual de cada uno de los pacientes).
- `experimento$Tratamiento`, de tipo factor que almacena los valores de la variable T (el tipo de tratamiento empleado).

Recuerda que, para acceder a las distintas variables de un `data.frame` (piensa en ellas como las columnas de una tabla), utilizamos la notación de corchetes o también el símbolo `$`, entre el nombre del `data.frame` y el de la variable. En RStudio puedes ver el contenido del `data.frame` `experimento` (en una nueva pestaña del *Editor de Código*) usando el comando

```
View(experimento)
```

Esto es especialmente interesante cuando el `data.frame` es muy grande, y verlo en la consola no resulta práctico. Pero recuerda también las funciones `head` y `tail`, que muchas veces son suficientes para esto.



	Respuesta	Tratamiento
1	5.25	T1
2	5.06	T1
3	5.04	T4
4	4.92	T1
5	3.08	T2
6	5.04	T5
7	5.39	T5
8	2.60	T2
9	5.13	T5
10	3.14	T2
11	5.41	T3
12	5.37	T1
13	4.92	T5
14	4.87	T3
15	5.68	T3
16	5.04	T4
17	5.10	T1
18	6.06	T4

¿Cómo se usan los factores en R? La contestación no puede ser completa en el espacio de este tutorial, porque los factores son un ingrediente clave del lenguaje de R, e intervienen en muchísimas construcciones del sistema. Pero, en cualquier caso, podemos empezar por lo más sencillo. ¿Qué aspecto tiene un vector de tipo factor? En el caso del `data.frame experimento` que acabamos de crear, al mostrar el vector `experimento$Tratamiento` se obtiene esto:

```
experimento$Tratamiento

##  [1] T1 T1 T4 T1 T2 T5 T5 T2 T5 T2 T3 T1 T5 T3 T3 T4 T1 T4 T1 T5 T4 T3 T4
## [24] T4 T1 T4 T1 T2 T3 T5 T5 T4 T2 T4 T4 T5 T1 T3 T3 T1 T3 T2 T2
## Levels: T1 T2 T3 T4 T5
```

Como se ve, R muestra el contenido del vector, junto con sus niveles. Si sólo queremos mostrar los niveles del factor, podemos usar la función `levels`, así:

```
levels(experimento$Tratamiento)

## [1] "T1" "T2" "T3" "T4" "T5"
```

Fíjate en que en el factor no aparecían comillas, pero que al usar `levels` sí que aparecen. ¿Qué diferencia hay entre las dos situaciones? Para entender esto, es preciso recordar que en R las comillas indican que estamos ante un vector de tipo character.

Para verlo más claramente podemos pedirle a R que convierta el `factor` en `character` mediante la función `as.character`:

```
as.character(experimento$Tratamiento)

##  [1] "T1" "T1" "T4" "T1" "T2" "T5" "T5" "T2" "T5" "T2" "T3" "T1" "T5" "T3"
## [15] "T3" "T4" "T1" "T4" "T1" "T5" "T4" "T3" "T4" "T4" "T1" "T4" "T1" "T2"
## [29] "T3" "T5" "T5" "T4" "T2" "T4" "T4" "T5" "T1" "T3" "T3" "T1" "T3" "T2"
## [43] "T2"
```

Las comillas que aparecen en este caso indican precisamente que se trata de un vector de tipo `character`. Los factores, en cambio, son otra cosa: aunque los factores representan variables cualitativas, R los gestiona internamente (de forma más eficiente) mediante códigos numéricos. Al

mostrarlos, a petición nuestra, el uso de las comillas permite distinguir entre un vector de tipo `character` (con comillas) y un vector de tipo `factor` (sin comillas).

Cuando usamos `levels` le estamos pidiendo a R que nos enseñe los *nombres* de los niveles del factor. Y, en tanto que nombres, estamos hablando de valores de tipo `character`. Pero no hay que confundir el nivel con el *nombre del nivel*. Para tratar de aclararlo un poco más: si tienes un factor `genero`, con muchos datos (imagínate miles de ellos), es posible que sus niveles se llamen `hombre` y `mujer`. Si un usuario de habla inglesa recibe ese fichero de datos, es probable que quiera cambiar *los nombres* de los niveles por `man` y `woman` respectivamente, pero ese cambio de nombre sólo afecta a dos *palabras* y no afecta en absoluto a la forma en la que R gestiona internamente los niveles (los miles de datos).

4.1.1. Creando un factor manualmente.

Como hemos dicho, cuando R usa una función como `read.table` para cargar un `data.frame`, los campos alfanuméricos (que contienen cadenas de texto, como el campo `tratamientos` de nuestro ejemplo) se convierten, *por defecto* en factores. Pero puede que, a veces, queramos preservar esos datos como variables de tipo `character`. Si es eso lo que queremos, basta con usar la opción `stringsAsFactors=FALSE` en la función `read.table`.

En cualquier caso, el uso de `read.table` y funciones similares es la forma más habitual de ccrear factores en R. Pero en algunas ocasiones necesitaremos crear directamente un factor a partir de un vector de datos ya existente. La forma de hacerlo es usando la función `factor`. Por ejemplo, antes hemos definido el vector

```
pacientesPorGenero

##  [1] "hombre" "mujer"  "mujer"  "hombre" "mujer"  "hombre" "hombre"
##  [8] "hombre" "mujer"  "mujer"  "mujer"
```

Este vector es de tipo `character`. No es, por lo tanto, un factor (como indican las comillas). Para crear un vector llamado `genero` de tipo `factor` a partir de estos datos, usamos la función factor de la forma más sencilla posible:

```
(genero = factor(pacientesPorGenero))

##  [1] hombre mujer  mujer  hombre mujer  hombre hombre hombre mujer  mujer
## [11] mujer
## Levels: hombre mujer
```

Y ahora los niveles se obtienen con:

```
levels(genero)

## [1] "hombre" "mujer"
```

De hecho nuestro usuario angloparlante podría usar `levels` para traducir los nombres de los niveles, así:

```
levels(genero) = c("man", "woman")
genero

##  [1] man   woman woman man   woman man   man   woman woman woman
## Levels: man woman
```

Aunque también se puede aprovechar la creación del vector factor para cambiar los nombres de los niveles, usando la opción `labels` de la función `factor`:

```
(genero = factor(pacientesPorGenero, labels=c("man", "woman")))

##  [1] man   woman woman man   woman man   man   man   woman woman woman
## Levels: man woman
```

4.1.2. Advertencia: sobre `levels` y `labels`.

El matiz al que ahora nos vamos a referir resulta confuso para muchos usuarios principiantes de R. Así que si al principio te cuesta entenderlo, no te preocupes demasiado. Cuando tengas más experiencia y surja la necesidad de aclararlo, podrás volver sobre estas ideas. Intenta hacer una primera lectura de este apartado, pero si te pierdes, sigue adelante. Ya habrá tiempo más adelante para entenderlo bien.

El método que acabamos de presentar, usando `labels` ha funcionado en este caso porque el orden alfabético de *hombre*, *mujer* en español coincide con el orden de *man*, *woman* en inglés. Pero imagínate que el vector original estuviera en alemán. En ese idioma *hombre* y *mujer* se dicen, respectivamente, *Mann* y *Frau*. Así que, como ves, el orden alfabético es distinto. El vector original, en alemán, sería:

```
pacientesPorGenero=c("Mann", "Frau", "Frau", "Mann", "Frau", "Mann", "Mann", "Mann",
    "Frau", "Frau", "Frau")
```

Y si ahora le aplicas `factor` directamente, como hemos hecho antes, se obtiene:

```
(genero = factor(pacientesPorGenero))

##  [1] Mann Frau Frau Mann Frau Mann Mann Mann Frau Frau Frau
## Levels: Frau Mann
```

Lo más importante es que te fijes en que R escribe los factores **en orden alfabético:** primero `Frau` y luego `Mann`. Y eso tiene una consecuencia muy importante. Si, ingenuamente, tratas de hacer la traducción al español:

```
(genero = factor(pacientesPorGenero, labels=c("hombre", "mujer")))

##  [1] mujer  hombre hombre mujer  hombre mujer  mujer  mujer  hombre hombre
## [11] hombre
## Levels: hombre mujer
```

enseguida descubrirás que hombres y mujeres se han intercambiado. ¿Cómo ha sucedido esto? Vamos despacio, para entender lo que ocurre y ver cómo evitarlo.

Para convertir el vector `pacientesPorGenero` en un factor, lo primero que hace R es aplicarle la función `unique`, para ver cuáles son las *palabras distintas* que aparecen en ese vector:

```
unique(pacientesPorGenero)

## [1] "Mann" "Frau"
```

Pero el resultado no está ordenado alfabéticamente, como queda patente en este ejemplo. Y ahora viene el paso clave: lo siguiente que hace R es ordenar alfabéticamente estas palabras, usando la función `sort`:

```
sort(unique(pacientesPorGenero))

## [1] "Frau" "Mann"
```

Esta lista *ordenada alfabéticamente* es la que R usa para establecer los niveles del factor *y el orden de sus etiquetas*. Así que cuando hemos usado `labels` para traducir, R ha interpretado que *hombre* era la traducción de *Frau* (la primera etiqueta por orden alfabético) mientras que *mujer* es la traducción de *Mann*. Un pequeño desastre, pero que, en algunas ocasiones, le ha costado un disgusto a más de un usuario de R desprevenido.

¿Cómo lo arreglamos? Con un argumento opcional de `factor`, llamado quizá confusamente `levels`, que nos deja escribir explícitamente el orden en el que queremos que R coloque los nombres de los niveles del factor. En este ejemplo, la solución consiste en hacer:

```
(genero = factor(pacientesPorGenero,
        levels = c("Mann", "Frau"), labels=c("hombre", "mujer")))

##  [1] hombre mujer  mujer  hombre mujer  hombre hombre hombre mujer  mujer
## [11] mujer
## Levels: hombre mujer
```

Al trabajar de esta manera, nos aseguramos de que el orden de los términos en `levels` coincide con el de `labels` y evitamos cualquier riesgo de acabar confundidos, *lost in traslation*. Naturalmente, aunque hemos usado un ejemplo con distintos idiomas, esta situación puede presentarse con cualquier cambio en los niveles de un factor. Imagínate que estamos trabajando en gestión del tráfico por carretera y queremos crear un factor que indique el código correspondiente al estado del tráfico en distintas carreteras. Podríamos ser algo como:

```
(estadoCarreteras = factor(c("rojo", "verde", "amarillo", "verde", "rojo",
    "amarillo", "verde", "verde", "rojo", "amarillo")))

##  [1] rojo     verde    amarillo verde    rojo     amarillo verde
##  [8] verde    rojo     amarillo
## Levels: amarillo rojo verde
```

¿Te has fijado en el orden en el que R ha escrito los niveles? Es el orden alfabético, claro. Si quieres mantener un orden concreto como *rojo, amarillo, verde*, debes usar `levels` y hacer algo como esto:

```
(estadoCarreteras = factor(c("rojo", "verde", "amarillo", "verde", "rojo",
    "amarillo", "verde", "verde", "rojo", "amarillo"),
    levels = c("rojo", "amarillo", "verde")))

##  [1] rojo     verde    amarillo verde    rojo     amarillo verde
##  [8] verde    rojo     amarillo
## Levels: rojo amarillo verde
```

Y si además quieres usar `labels` para cambiar las *etiquetas*, asegúrate de que mantienes el mismo orden:

```
(estadoCarreteras = factor(c("rojo", "verde", "amarillo", "verde", "rojo",
    "amarillo", "verde", "verde", "rojo", "amarillo"),
    levels = c("rojo", "amarillo", "verde"),
    labels = c("atasco", "moderado", "fluido")))

##  [1] atasco   fluido   moderado fluido   atasco   moderado fluido
##  [8] fluido   atasco   moderado
## Levels: atasco moderado fluido
```

Para terminar con esta cuestión tan espinosa de los niveles de un factor y sus etiquetas, vamos a suponer que has creado el factor, poniendo cuidado en elegir el orden de los niveles con `levels`:

```
(estadoCarreteras = factor(c("rojo", "verde", "amarillo", "verde", "rojo",
    "amarillo", "verde", "verde", "rojo", "amarillo"),
    levels = c("rojo", "amarillo", "verde")))
```

```
##  [1] rojo     verde    amarillo verde    rojo     amarillo verde
##  [8] verde    rojo     amarillo
## Levels: rojo amarillo verde
```

y que *a posteriori* decides cambiar *los nombres* de las etiquetas. Por ejemplo, vamos a cambiar *rojo, amarillo, verde* por *atasco, moderado, fluido*. Para hacer esto, usamos la función `levels` de nuevo:

```
levels(estadoCarreteras) = c("atasco", "moderado", "fluido")
estadoCarreteras
```

```
##  [1] atasco   fluido   moderado fluido   atasco   moderado fluido
##  [8] fluido   atasco   moderado
## Levels: atasco moderado fluido
```

Personalmente, encuentro este último punto especialmente confuso y en su momento me costó trabajo asimilarlo. Fíjate en que antes, cuando lo hacíamos durante la creación del factor usábamos `labels` pero ahora, a posteriori, debemos usar `levels`. De hecho, si tratas de usar `labels` para esto ¡R se enfada!:

```
labels(estadoCarreteras) = c("atasco", "moderado", "fluido")
```

```
## Error in labels(estadoCarreteras) = c("atasco", "moderado", "fluido"): no se pudo
encontrar la función "labels<-"
```

Como hemos dicho, todo este pequeño lío de `levels` y `labels` te costará algo de trabajo al principio (y no es, desde luego, uno de los aspectos mejor resueltos de R) pero con la experiencia irás comprendiéndolo mejor.

4.2. Usando los factores para explorar los datos

Los factores permiten modular el comportamiento de algunas funciones de R, de manera que la información que se obtiene tiene en cuenta esa organización en niveles de los valores que estamos analizando. Por ejemplo, en el Tutorial02 vimos como usar la función `summary` de R para obtener un resumen descriptivo de un vector de datos (media y percentiles, básicamente). Ahora, que tenemos un `data.frame` con una clasificación por tratamientos, nos gustaría seguramente obtener esa misma información, pero para cada uno de los distintos tratamientos por separado. Lo primero que queremos hacer notar es que no sirve aplicar `summary` directamente al `data.frame`:

```
summary(experimento)
```

```
##    Respuesta    Tratamiento
##  Min.   :2.60   T1:10
##  1st Qu.:4.54   T2: 7
##  Median :5.04   T3: 8
##  Mean   :4.79   T4:10
##  3rd Qu.:5.25   T5: 8
##  Max.   :6.06
```

Como se ve, se obtiene un resumen por separado para cada uno de los campos del `data.frame`, con `Respuesta` por un lado, y `Tratamiento` por otro, sin reconocer el vínculo entre ambas. Naturalmente, puesto que ya hemos aprendido a seleccionar los elementos de un vector mediante condiciones (usando los corchetes `[ ]`), podemos ir separando cada uno de los grupos de tratamiento, y aplicarles la función `summary` a los vectores resultantes. Sería algo como esto (mostramos, por ejemplo el segundo grupo de tratamiento):

```
(tratamiento2 = experimento[experimento$Tratamiento == "T2", 1])

## [1] 3.08 2.60 3.14 3.36 3.31 3.83 3.18

summary(tratamiento2)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.60    3.11    3.18    3.21    3.34    3.83
```

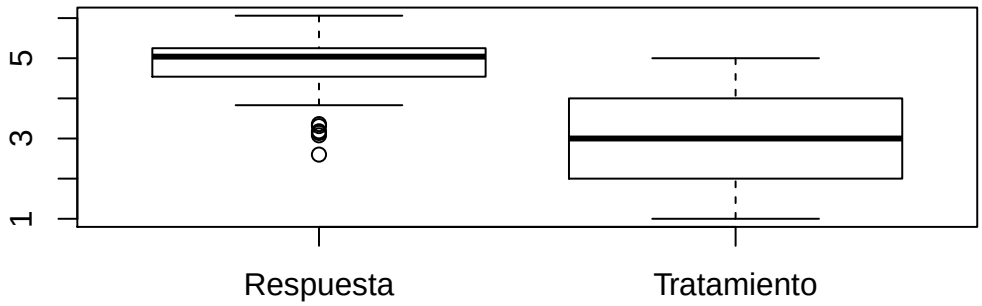
Esto funciona, pero como se puede apreciar, trabajar así es engorroso. Para obtener lo que queremos, hay una manera mucho más natural de proceder, usando la función `tapply`. Veamos como funciona:

```
tapply(experimento$Respuesta, experimento$Tratamiento, summary)

## $T1
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4.70    4.96    5.08    5.08    5.23    5.37
##
## $T2
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.60    3.11    3.18    3.21    3.34    3.83
##
## $T3
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4.52    5.01    5.29    5.23    5.48    5.71
##
## $T4
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4.40    4.60    4.97    5.06    5.40    6.06
##
## $T5
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4.56    4.94    5.02    5.04    5.18    5.39
```

Ahora sí: como ves, R ha reconocido esa estructura en niveles del factor `Tratamiento`, y no está describiendo la variable `Respuesta` para cada uno de los grupos de tratamiento que conforman nuestros datos. El resultado es una *tabla* (de ahí la `t` inicial de `tapply`) que contiene, ordenados por filas, los resultados de aplicar la función `summary` a los valores de `Respuesta`, agrupadas según los distintos niveles del factor `Tratamiento`.

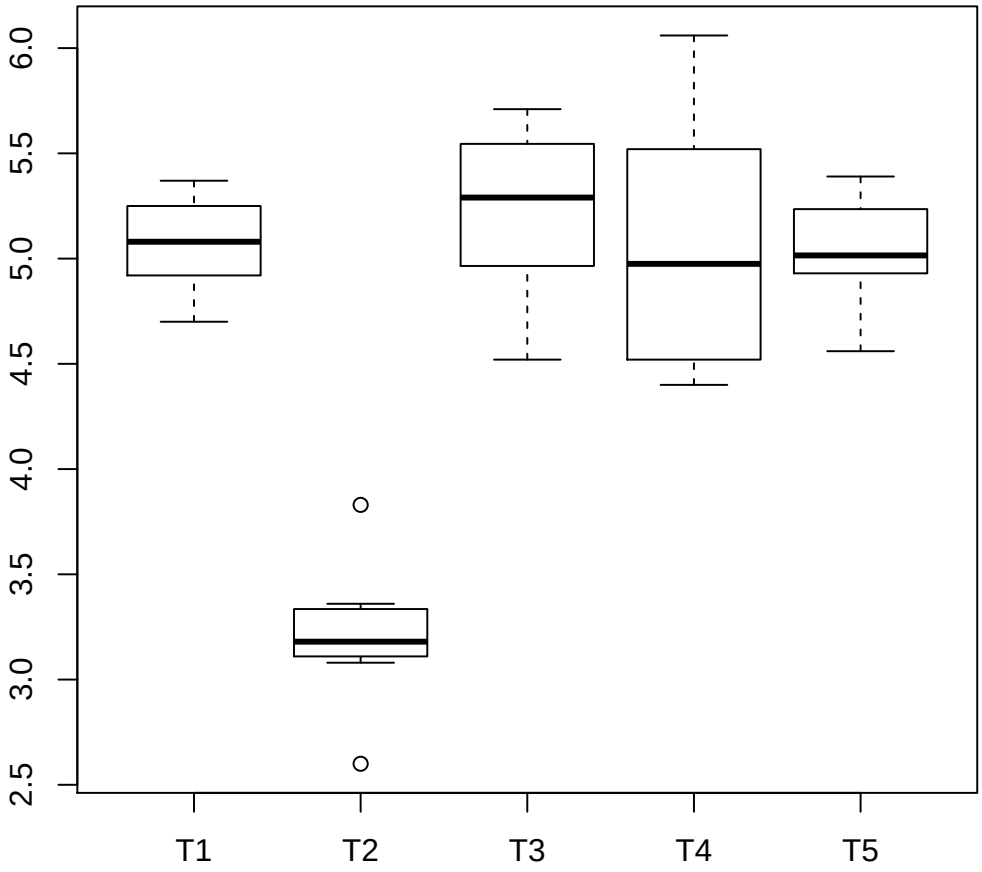
Otro ejemplo de las ventajas del lenguaje de factores se obtiene al pensar en la función `boxplot`. Si aplicamos esa función directamente al `data.frame`, ejecutando:

```
boxplot(experimento)
```



se obtiene un gráfico que es, de hecho un sinsentido. Fíjate en que se incluye un boxplot para la variable `Tratamiento` ¡qué es cualitativa! Evidentemente esto no es correcto. Para obtener lo que queremos basta con escribir:

```
boxplot(Respuesta ~ Tratamiento, data = experimento)
```



El resultado es un gráfico mucho más adecuado para comparar los resultados de los tratamientos.

La sintaxis `Respuesta ~ Tratamiento` que hemos usado por primera vez en esta llamada, es la que usa R para decir algo así como *"la forma en que los valores de* `Respuesta` *dependen de los tratamientos"*. Este tipo de expresiones se denominan fórmulas en R. Y en general, en R, el símbolo `~` se utiliza para indicar una dependencia o relación entre variables (lo obtienes, en un teclado estándar de PC, pulsando las teclas `AltGr` y 4 a la vez). En este caso, lo que estamos estudiando es si hay alguna relación entre la variable cualitativa (factor) `Tratamiento`, y la variable cuantitativa `Respuesta`. En particular, una pregunta que a menudo nos interesa consiste en saber si la respuesta media al tratamiento es distinta según el grupo de tratamiento que se considere. Este tipo de problemas, en los que se investiga la relación entre varias variables aleatorias (de distintos tipos), forman el contenido de la cuarta parte del curso, y por eso estamos empezando a preparar el terreno en este tutorial.

4.3. La función `cut`

Otra de las primeras funciones de R que se aprenden al comenzar a trabajar con factores es la función `cut`, cortar en inglés. Y el nombre describe muy bien lo que hace esta función. Se usa para cortar en piezas los datos de una variable cuantitativa, con el fin de agruparlos en clases o intervalos, como discutimos en la Sección 1.1.3 del Capítulo 1 del libro.

Por ejemplo, el fichero adjunto

Tut08-Edades.csv

contiene las edades, en años, de un grupo de 500 personas. Imagínate que queremos agrupar esos datos, según la edad, en estas categorías:

- *Menores*, con edades menores que 18 años,
- *Jovenes*, con edades entre 18 y 30 años,
- *MedianaEdad*, con edades entre 31 y 64 años,
- *Mayores*, con edades de 65 años en adelante.

Vamos a ver como usar `cut` para hacer eso en R. Primero leemos el fichero (recuerda que debe estar en la carpeta `datos` del *Directorio de Trabajo*) y lo guardamos en un vector de datos, llamado `edades`, al que aplicaremos la función `cut`.

```
edades = read.table(file="../datos/Tut08-Edades.csv")[ ,1]
```

Pero antes de cortar, tenemos que pensar bien dónde vamos a dar los cortes. Los intervalos que R usa, en la función `cut` son, por defecto, de la forma $(a, b]$. Es decir, incluyen el límite derecho pero no el izquierdo (esto se puede cambiar). Eso implica que so hacemos una elección poco cuidadosa de los puntos de corte, como en:

```
cortesEdad = c(0, 18, 30, 65, 100)
```

terminaremos con un intervalo de edades como $18 < edad \leq 30$, que no se corresponde con lo que queremos, porque no incluye a las personas de 18 años. Algo parecido sucede con el intervalo $30 < edad \leq 65$, que no es lo que queremos porque incluye a las personas de 65 años. Una elección algo más meditada nos conduce a:

```
cortesEdad = c(0, 17, 30, 64, 100)
```

Esto está casi bien, pero aún tenemos un problema. ¿Cuáles son los intervalos $(a, b]$ determinados por este segundo vector `cortesEdad`? El primer intervalo es $(0, 17]$. Eso significa que si alguna de las edades es 0 (como, de hecho, sucede), no quedará incluida en ese intervalo. Para evitar ese problema, la función `cut` dispone de una opción, `include.lowest=TRUE`, que permite cerrar a la izquierda el primero de los intervalos. Con eso estamos listos para usar la función `cut`:

```
factorEdad = cut(edades, breaks=cortesEdad, include.lowest=TRUE)
```

El resultado, que hemos guardado en la variable `factorEdad` es un vector, de tipo `factor` (puedes usar `class(factorEdad)` para comprobarlo), que contiene, para cada elemento de `edades` el intervalo de edades (nivel del factor) al que pertenece. Para que lo veas con más claridad, vamos a ver, juntos, los primeros elementos de los vectores `edades` y `factorEdad`, usando `head`:

```
head(edades)

## [1] 16 59 27 32 45 59

head(factorEdad)

## [1] [0,17]  (30,64] (17,30] (30,64] (30,64] (30,64]
## Levels: [0,17] (17,30] (30,64] (64,100]
```

Como ves, en cada posición de `factorEdad` está el intervalo de edades (nivel del factor) al que pertenece el correspondiente elemento de `edades`. Fíjate en que el intervalo del primer elemento es [0,17] , cerrado en ambos extremos.

Además, R nos recuerda al final el conjunto de nombres de los valores (niveles) que puede tomar el factor. Como hemos visto, los niveles de un factor se pueden obtener (y modificar) usando la función `levels` así:

```
levels(factorEdad)

## [1] "[0,17]"   "(17,30]"  "(30,64]"  "(64,100]"
```

Insistimos en que estos son *los nombres de los niveles*, para recordar que, internamente, R utiliza códigos numéricos para trabajar con los niveles. Por eso es bueno usar una palabra como etiquetas para referirse a esos nombres.

Y, como hemos visto, si queremos cambiar estas etiquetas, podemos reemplazarlas con otras más útiles para nosotros. La forma de hacerlo, en este ejemplo, es esta:

```
levels(factorEdad) = c("menor", "joven", "medianaEdad", "mayor")
```

Ahora, si pruebas a ejecutar de nuevo `head(factorEdad)`, verás el efecto que ha tenido ese cambio.

5. Ejercicios adicionales y soluciones.

Soluciones de algunos ejercicios.

- **Ejercicio 1, pág. 2**

1. Incluimos aquí el código del fichero con los datos de este ejercicio y el resultado que se obtiene.

```
#####################################################
# www.postdata-statistics.com
# POSTDATA. Introducción a la Estadísitica
# Tutorial-08.
#
# Fichero de instrucciones R para calcular
# un intervalo de confianza (1-alfa) para la proporcion p de
# una poblacion tipo Binomial (Bernouilli), a partir de una
# muestra con n datos.
```

```r
#
# Este fichero usa los valores de una muestra,
# previamente calculados (numero de datos, proporcion muestral)
#
################################################################

rm(list=ls()) #limpieza inicial

# Introduce el numero de datos de la muestra:

n = 456

# Introduce aqui el valor de pMuestral, la proporcion muestral .
# Recuerda que pMuestral es el numero de casos favorables en la muestra
# dividido por n.

pMuestral = 139 / 456

# LEE ESTAS INSTRUCCIONES ATENTAMENTE:
# SI LA MUESTRA TIENE < 30 ELEMENTOS O SI N*p<5 O SI n*q<5,
# NO USES ESTE FICHERO!!
# ASEGURATE DE HABER ENTENDIDO ESTAS INSTRUCCIONES

# Nivel de confianza deseado.

nc = 0.95

################################################
#NO CAMBIES NADA DE AQUI PARA ABAJO
################################################
(alfa = 1 - nc )

## [1] 0.05

# Calculamos el valor critico:

(z_alfa2 = qnorm( 1 - alfa / 2 ) )

## [1] 1.96

# Calculamos qMuestral

qMuestral = 1 - pMuestral

# Comprobamos la condicion sobre n
if((n<=30)|(n*pMuestral<5)|(n*qMuestral<5)){
  warning("NO SE CUMPLEN LAS CONDICIONES")
}


# Semianchura del intervalo
(semianchura=z_alfa2 * sqrt(pMuestral * qMuestral / n) )

## [1] 0.042251

# Y el intervalo de confianza (a,b) para mu es este:

(intervalo = pMuestral + c(-1, 1) * semianchura )

## [1] 0.26257 0.34708
```

2. Con `prop.test` el intervalo se obtiene así:

```
  prop.test(x = 139, n = 456, alternative = "two.sided", conf.level = 0.95)

##
##  1-sample proportions test with continuity correction
##
## data:  139 out of 456, null probability 0.5
## X-squared = 68.7, df = 1, p-value <2e-16
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.26330 0.34969
## sample estimates:
##       p
## 0.30482
```

La función `prop.test` también incluye un argumento `p` que es lo que nosotros llamamos p_0 en el contraste de hipótesis. Para este ejercicio, dado que sólo estamos interesados en el intervalo de confianza, no es necesario incluir el valor de p.

Para los datos de 2008 es:

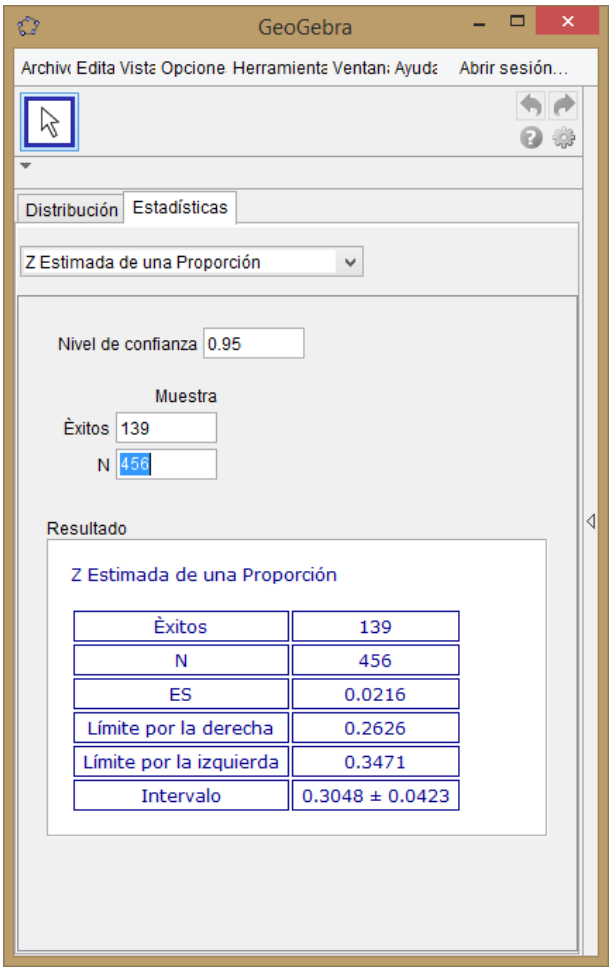
```
  prop.test(x = 138, n = (138 + 270), alternative = "two.sided", conf.level = 0.95)

##
##  1-sample proportions test with continuity correction
##
## data:  138 out of (138 + 270), null probability 0.5
## X-squared = 42.1, df = 1, p-value = 8.8e-11
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.29285 0.38672
## sample estimates:
##       p
## 0.33824
```

3. En el panel de *Cálculo Simbólico* de GeoGebra o en la *Línea de Entrada* puedes usar este comando:

```
IntervaloProporciónZ[139/456, 456, 0.95]
```

Recuerda ajustar el número de cifras en el menú opciones y recuerda también que la salida aparece en sitios distintos según utilices una u otra forma de entrada.

También tienes la opción de usar la *Calculadora de Probabilidades* como se muestra en la figura:



- **Ejercicio 2, pág. 4**

1. Código del fichero con los datos de este ejercicio y el resultado que se obtiene:

```
#########################################################
# www.postdata-statistics.com
# POSTDATA. Introducción a la Estadísitica
# Tutorial-08.
#
# Fichero de instrucciones R para calcular un contraste de
# hipotesis para la proporcion, a partir de una
#
#          MUESTRA GRANDE
#
# con n » 30 datos.
#
# El fichero no funcionara si no introduces todos los datos.
#
#####################################################################

    rm(list=ls())
# Numero de elementos en la muestra
    (n= 105) #SE SUPONE QUE LA MUESTRA ES GRANDE, n>30

## [1] 105

# Proporcion muestral (¡es una fraccion! Numero de exitos/n)
    (pMuestral= 39 / 105)

## [1] 0.37143
```

```r
# Valor a contrastar de la proporcion (aparece en la hipotesis nula)
    (p0= 0.35)

## [1] 0.35

# ¿Que tipo de contraste estamos haciendo?
    # Escribe 1 si la HIP. ALTERNATIVA es p > p0, 2 si es p < p0, 3 si es p distinto de p0
    TipoContraste = 1
#Nivel de significacion
    (nSig= 0.95)

## [1] 0.95

################################################
# NO CAMBIES NADA DE AQUÍ PARA ABAJO
################################################

# Calculo de alfa
    (alfa=1-nSig)

## [1] 0.05

# Calculo de q0
    q0 = 1- p0
# Comprobamos la condicion sobre n
if((n<=30)|(n*p0<5)|(n*q0<5)){
  warning("NO SE CUMPLEN LAS CONDICIONES")
}
# Calculo del estadistico del contraste
    (Estadistico=(pMuestral-p0) / sqrt( (p0 * q0) / n ))

## [1] 0.46036

# Funcion para el calculo del p-valor
    pValor=function(EstadCon,tipoCon){
      if(tipoCon==1){
        (pV=1-pnorm(EstadCon))
      }
      if(tipoCon==2){
        (pV=pnorm(EstadCon))
      }
      if(tipoCon==3){
        pV=2*(1-pnorm(abs(EstadCon)))
      }
      return(paste("El p-Valor es ",pV,sep="",collapse=""))
    }
# Funcion para el calculo del límite de la región de rechazo
    RegionRechazo=function(alfa,tipoCon){
      if(tipoCon==1){
        (regionRech=paste("valores del Estadistico mayores que ",qnorm(1-alfa)) )
      }
      if(tipoCon==2){
        (regionRech=paste("valores del Estadistico menores que ",qnorm(alfa)) )
      }
      if(tipoCon==3){
        (regionRech=paste("valores del Estadistico mas alejados del origen que ",qnorm(1-alfa/2)) )
      }
      regionRech=paste("La region de rechazo la forman los ",regionRech,sep="",collapse="")
      return(regionRech)
    }

# Y ahora se aplican ambas funciones para mostrar los resultados
    pValor(Estadistico,TipoContraste)

## [1] "El p-Valor es 0.322629077655666"
```

```
    Estadistico

## [1] 0.46036

    RegionRechazo(alfa,TipoContraste)

## [1] "La region de rechazo la forman los valores del Estadistico mayores que  1.64485362695147"
```

2. Código del fichero con los datos de este ejercicio y el resultado que se obtiene. Las conclusiones del contraste aparecen tras el código.

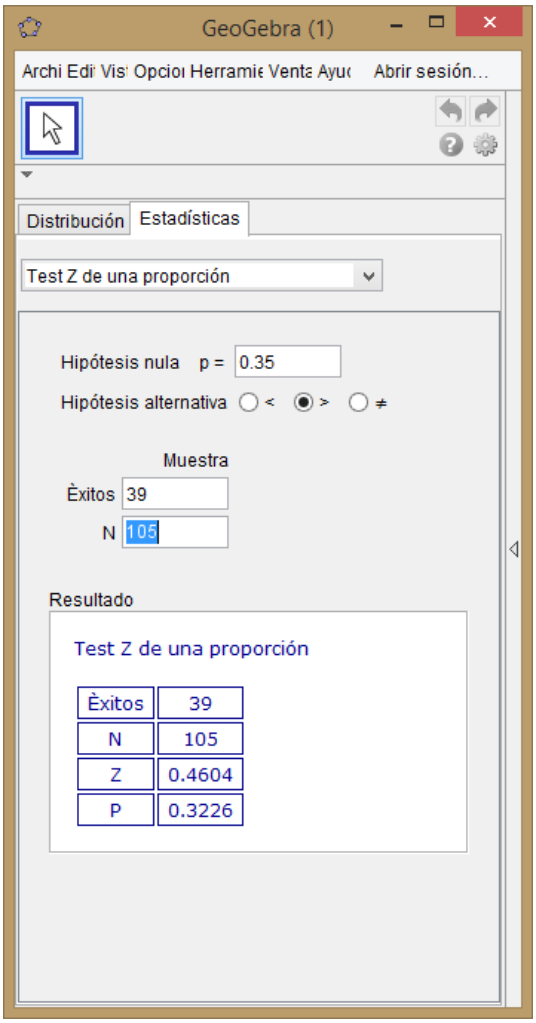
```
#####################################################
# www.postdata-statistics.com
# POSTDATA. Introducción a la Estadísitica
# Tutorial-08.
#
# Fichero de instrucciones R para calcular un contraste de
# hipotesis para la proporcion, a partir de una
#
#         MUESTRA GRANDE
#
# con n » 30 datos.
#
# El fichero no funcionara si no introduces todos los datos.
#
#####################################################################

    rm(list=ls())
# Numero de elementos en la muestra
    (n= 2452) #SE SUPONE QUE LA MUESTRA ES GRANDE, n>30

## [1] 2452

# Proporcion muestral (¡es una fraccion! Numero de exitos/n)
    (pMuestral= 568 / 2452)

## [1] 0.23165

# Valor a contrastar de la proporcion (aparece en la hipotesis nula)
    (p0= 0.25)

## [1] 0.25

# ¿Que tipo de contraste estamos haciendo?
    # Escribe 1 si la HIP. ALTERNATIVA es p > p0, 2 si es p < p0, 3 si es p distinto de p0
    TipoContraste = 2
#Nivel de significacion
    (nSig= 0.95)

## [1] 0.95

################################################
# NO CAMBIES NADA DE AQUÍ PARA ABAJO
################################################

# Calculo de alfa
    (alfa=1-nSig)

## [1] 0.05
```

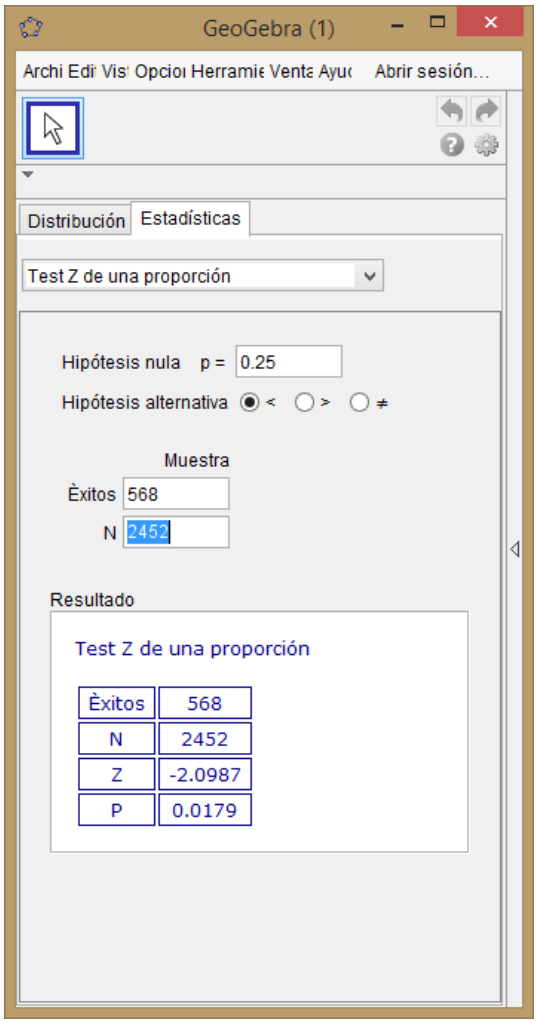
```r
# Calculo de q0
    q0 = 1- p0
# Comprobamos la condicion sobre n
if((n<=30)|(n*p0<5)|(n*q0<5)){
  warning("NO SE CUMPLEN LAS CONDICIONES")
}
# Calculo del estadistico del contraste
    (Estadistico=(pMuestral-p0) / sqrt( (p0 * q0) / n ))

## [1] -2.0987

# Funcion para el calculo del p-valor
    pValor=function(EstadCon,tipoCon){
      if(tipoCon==1){
        (pV=1-pnorm(EstadCon))
      }
      if(tipoCon==2){
        (pV=pnorm(EstadCon))
      }
      if(tipoCon==3){
        pV=2*(1-pnorm(abs(EstadCon)))
      }
      return(paste("El p-Valor es ",pV,sep="",collapse=""))
    }
# Funcion para el calculo del límite de la región de rechazo
    RegionRechazo=function(alfa,tipoCon){
      if(tipoCon==1){
        (regionRech=paste("valores del Estadistico mayores que ",qnorm(1-alfa)) )
      }
      if(tipoCon==2){
        (regionRech=paste("valores del Estadistico menores que ",qnorm(alfa)) )
      }
      if(tipoCon==3){
        (regionRech=paste("valores del Estadistico mas alejados del origen que ",qnorm(1-alfa/2)) )
      }
      regionRech=paste("La region de rechazo la forman los ",regionRech,sep="",collapse="")
      return(regionRech)
    }

# Y ahora se aplican ambas funciones para mostrar los resultados
    pValor(Estadistico,TipoContraste)

## [1] "El p-Valor es 0.0179214040476068"

    Estadistico

## [1] -2.0987

    RegionRechazo(alfa,TipoContraste)

## [1] "La region de rechazo la forman los valores del Estadistico menores que  -1.64485362695147"
```

Puesto que el p-valor es menor que 0.05, podemos rechazar la hipótesis nula (al 95 % pero no, por ejemplo, al 99 %) y concluir que lso datos respaldan la hipótesis de que la proporción de no creyentes es inferior al 25 %. También puedes usar el método de la región de rechazo, observando que el estadístico pertenece a esa región de rechazo.

- **Ejercicio 3, pág. 5**

Mostramos uno junto al otro los resultados obtenidos con la *Calculadora de Probabilidades* de GeoGebra para cada uno de los apartados de este ejercicio:





- **Ejercicio 4, pág. 11**

1. Con R sería

```
dpois(3, lambda=2)

## [1] 0.18045

dpois(10, lambda=2)

## [1] 0.00003819
```

En GeoGebra ejecutamos estos comandos:

```
Poisson[2, 3, false]

Poisson[2, 10, false]
```

Ten cuidado con el orden. En GeoGebra se coloca en primer lugar el valor de λ. Ten además en cuenta que si usas el panel de *Cálculo Simbólico* las respuestas serán (obviamente) simbólicas.

En Wolfram Alpha usaríamos los comandos:

```
P(X=3) for X~Poisson(2)

P(X=10) for X~Poisson(2)
```

2. El cálculo con R es:

```
ppois(2400, lambda=2489)

## [1] 0.037438
```

   Con GeoGebra el comando es:

```
Poisson[2489, 2400, true]
```

   La respuesta simbólica (al cabo de unos segundos) es excesivamente larga como para incluirla aquí. De hecho, en la versión 5.0.44 para Windows de GeoGebra este calculo simbólico funciona, pero produce un error en la versión 5.0.24 para Mac OSX.

   Puedes ver la respuesta simbólica completa más fácilmente usando este comando en Wolfram Alpha:

```
P(X<=2489) for X~Poisson(2400)
```

- **Ejercicio 5, pág. 13**

El comando necesario en Wolfram Alpha es:

```
sum((k - lambda)^2 * exp(-lambda) *  lambda^k / k!) from k=0 to oo
```

Fin del Tutorial08. ¡Gracias por la atención!