Tutorial 10: Regresión lineal simple.

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. Diagramas de dispersión y formato de los datos.

En el Capítulo 10 del libro se plantea el estudio de la relación entre dos variables cuantitativas X e Y. Los datos muestrales en los que se basa ese estudio consisten en una colección de pares de puntos:

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n),$$

Y una de las tareas esenciales en ese estudio es la de representar gráficamente estos puntos, porque esa gráfica contiene mucha información sobre la posible relación que tratamos de analizar. Vamos a empezar el tutorial viendo cómo podemos realizar esos diagramas de dispersión usando los programas que conocemos, al menos en los casos más sencillos.

1.1. En GeoGebra.

Empecemos con un ejemplo. La Tabla 1 muestra los datos que aparecen en una noticia titulada *Relación entre la renta per cápita y los resultados de PISA*, publicada por el periódico EL PAÍS en su edición on-line del 6 de diciembre de 2013. (El informe PISA es un análisis del rendimiento de los estudiantes, que la OCDE realiza periódicamente en muchos países del mundo). Los datos de la columna se refieren a la renta per cápita (rpc) en miles de euros para cada una de las comunidades autónomas que participaron en el estudio en 2012, mientras que la segunda columna (pisa) contiene las puntuaciones (promedio) obtenidas por los estudiantes de esa comunidad autónoma, en la prueba de Matemáticas. Este ejemplo es bastante sencillo, porque partimos de una cantidad muy moderada de puntos. Pero aún así nos va a servir para hacernos una idea de las dificultades que nos encontraremos en la práctica en este tipo de problemas. Por un lado, cuando pensamos en hacer que una persona pueda leer fácilmente la información de las coordenadas de los puntos, la

	rpc	pisa
Extremadura	15.394	461
Andalucía	19.960	472
Murcia	18.520	462
Galicia	20.723	489
Asturias	21.035	500
Castilla y León	22.289	509
Cantabria	22.341	491
ESPAÑA	22.772	484
Baleares	24.393	475
La Rioja	25.508	503
Aragón	25.540	496
Cataluña	27.248	493
Navarra	29.071	517
Madrid	29.385	504
País Vasco	30.829	505

Tabla 1: Datos de la noticia *Relación entre la renta per cápita y los resultados de PISA*, publicados por el periódico EL PAÍS en su edición on-line del 6 de diciembre de 2013.

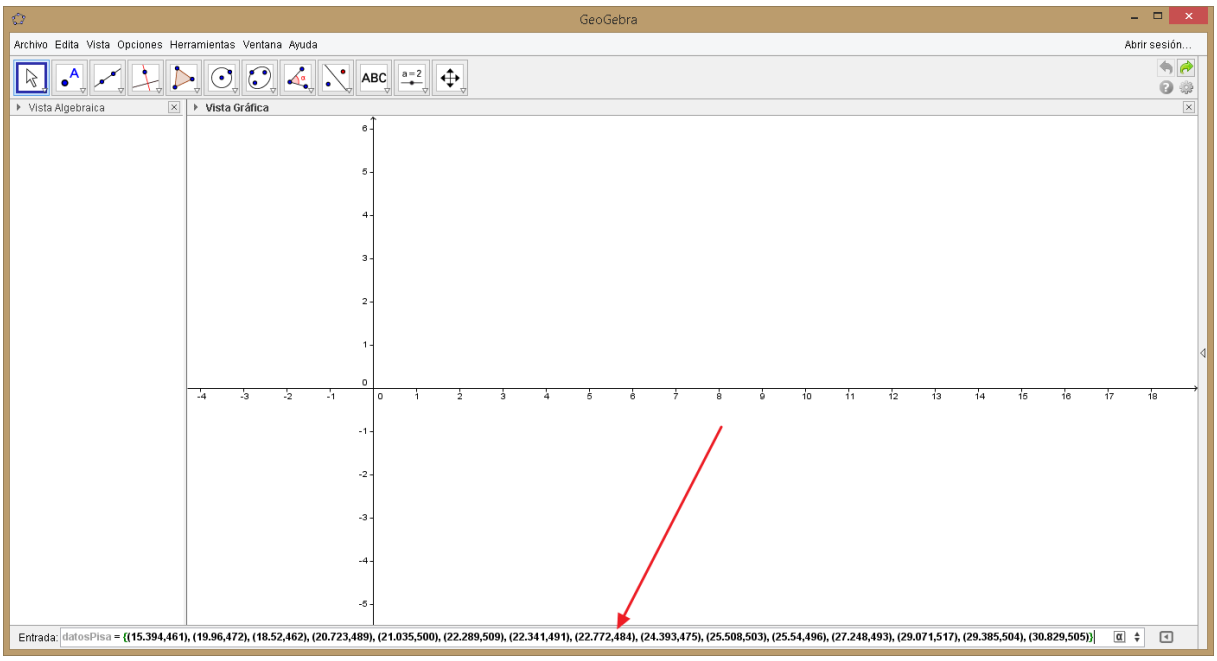
mejor manera es usar una tabla como la Tabla 1. Por otro lado, la forma más fácil de empezar a trabajar con estos datos en GeoGebra es partiendo de una *lista de puntos* como esta:

```
(15.394,461), (19.96,472), (18.52,462), (20.723,489), (21.035,500), (22.289,509),
(22.341,491), (22.772,484), (24.393,475), (25.508,503), (25.54,496), (27.248,493),
(29.071,517), (29.385,504), (30.829,505)
```

Puedes copiar esta lista de puntos y pegarla en la *Línea de Entrada* de GeoGebra rodeándola entre llaves así:

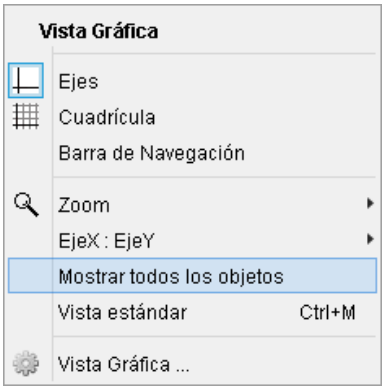
```
datosPisa = {(15.394,461), (19.96,472), (18.52,462), (20.723,489), (21.035,500),
(22.289,509), (22.341,491), (22.772,484), (24.393,475), (25.508,503), (25.54,496),
(27.248,493), (29.071,517), (29.385,504), (30.829,505)}
```

como se muestra en esta figura:

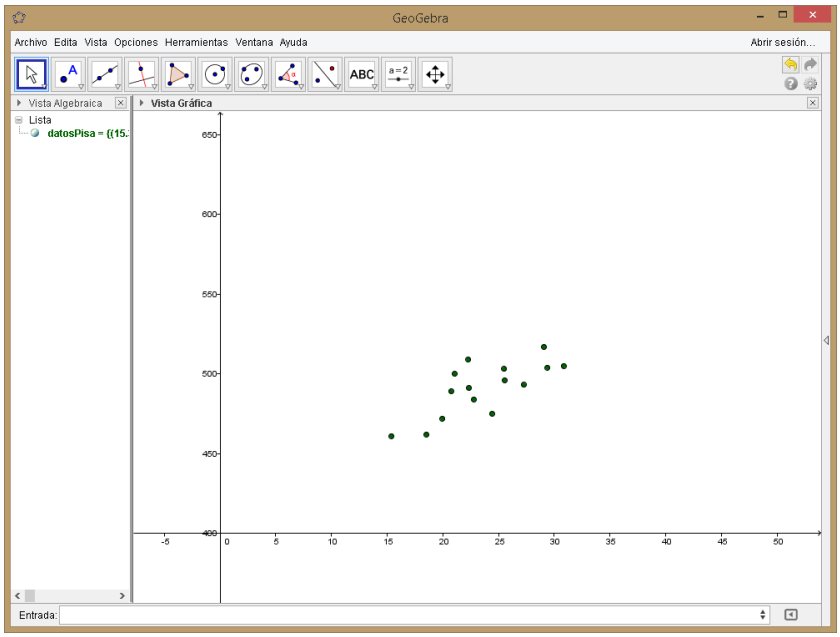


Cuando introduzcas estos datos en GeoGebra los verás aparecer en el panel de la izquierda, la *Vista Algebraica*, pero no en la *Vista gráfica*. Para hacerlos visibles tienes que pulsar sobre el botón circular situado a la izquierda de los datos en la *Vista Algebraica*, como muestra la figura:

Después de hacer esto lo más probable es que todavía no veas los puntos. Para verlos,haz click con el botón derecho del ratón en algún lugar de la *Vista gráfica*, para hacer aparecer el menú contextual de ese panel, y selecciona *Mostrar todos los objetos*, como se ve en esta figura:



Al hacerlo verás aparecer la nube de puntos. Todavía es posible que tengas que hacer zoom o cambiar la escala de los ejes para llegar a una visualización que nos parezca satisfactoria. Por ejemplo, yo he modificado la posición y escala de los ejes, para que el punto de corte ocurra en el valor 400 del eje vertical.



Puedes explorar el menú *Vista Gráfica* (identificado por la rueda dentada que aparece en el menú contextual que hemos usado antes), en el que puedes modificar estos y muchos otros parámetros de la visualización.

La limitación evidente de esta forma de trabajar es que en la mayoría de los casos partiremos de una tabla, o de un fichero csv con los datos. Por ejemplo, el fichero adjunto

Tut10-DatosPisaPais.csv,

cuyo contenido se muestra en la Figura 1 contiene los mismos datos de este ejemplo, en un formato mucho más común en la práctica. Una posibilidad es usar la *Hoja de Cálculo* que incorpora

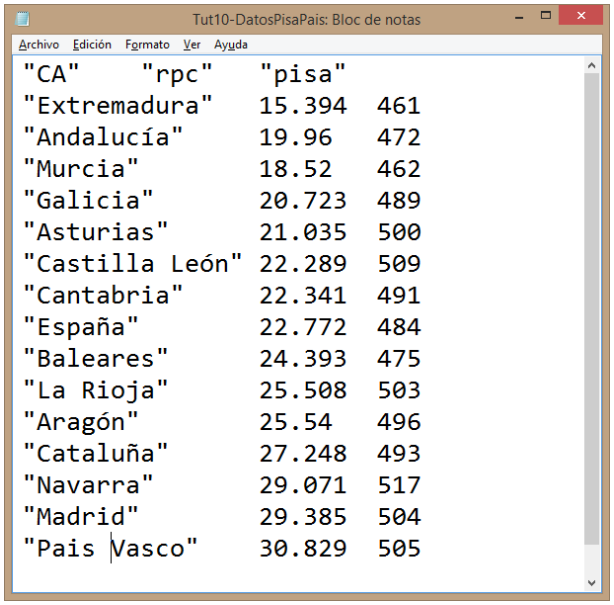


Figura 1: Fichero `csv` con los datos del ejemplo sobre el estudio PISA.

GeoGebra, e importar el fichero `csv` como hemos hecho en Calc. GeoGebra ofrece, en sus versiones recientes, muchas menos posibilidades de configuración de ese proceso de importación que, por ejemplo, Calc. Puedes encontrar más información en este enlace:

http://wiki.geogebra.org/en/Spreadsheet_View

Pero para trabajar cómodamente con ficheros de datos más complejos y, en general, para poder exprimir al máximo la información de estos datos, es sensiblemente mejor recurrir a un programa especializado como R.

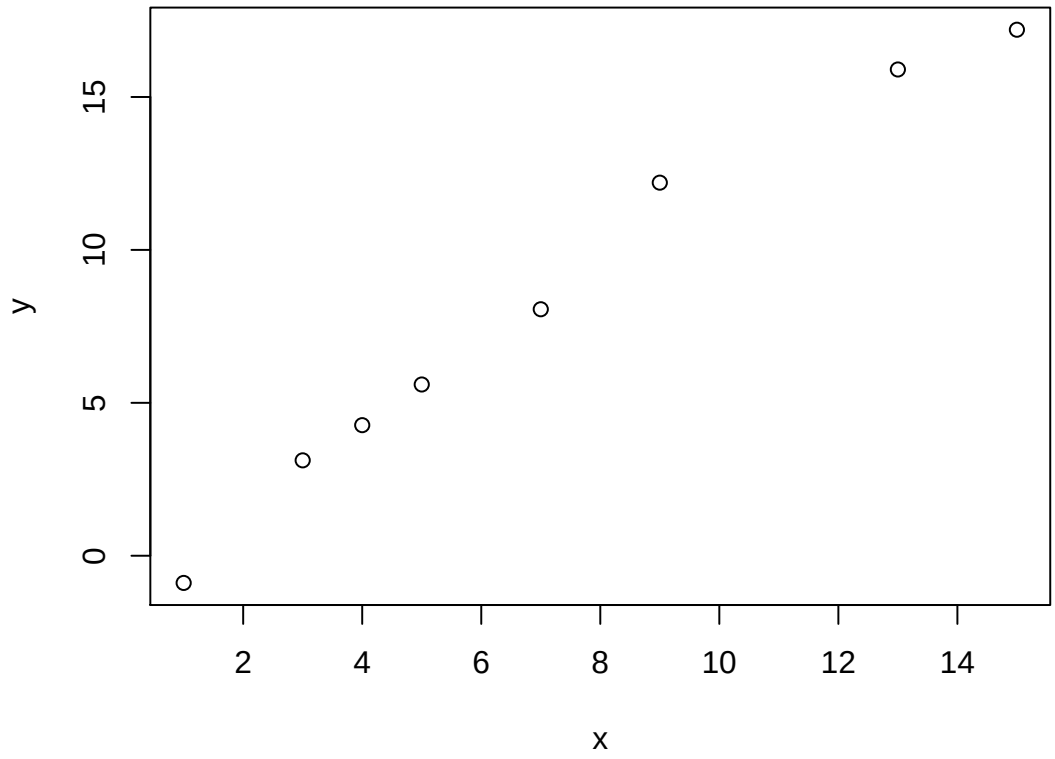
1.2. En R.

Vamos a empezar con un ejemplo muy elemental de cómo se dibuja un diagrama de dispersión en R. Partimos de dos vectores de coordenadas:

```
x = c(1, 3, 4, 5, 7, 9, 13, 15)
y = c(-0.888, 3.12, 4.27, 5.6, 8.06, 12.2, 15.9, 17.2)
```

Y para obtener el diagrama de dispersión basta con aplicar la función plot así:

```
plot(x, y)
```



El diagrama resultante es, desde el punto de vista gráfico, muy básico. Pero, como sucede siempre en R, las posibilidades para modificar este gráfico, adaptándolo a nuestras necesidades y deseos, son casi ilimitadas. Más adelante en este tutorial vamos a adentrarnos, apenas unos pasos, en el universo de los gráficos con R.

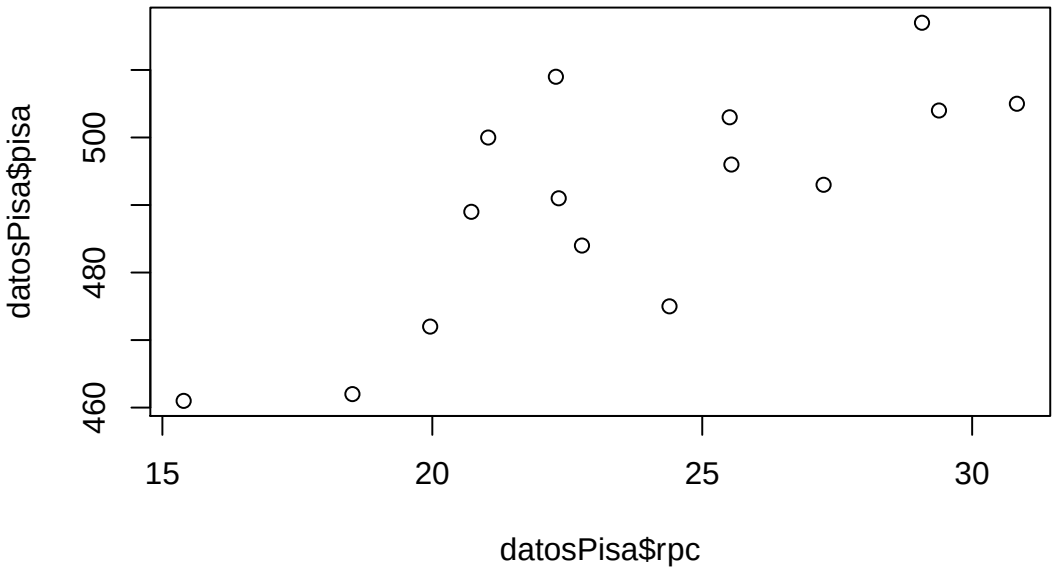
Lo que hemos hecho para obtener ese gráfico es el esquema básico. En ejemplos más complicados y realistas, como hemos dicho, el punto de partida será a menudo un fichero de datos. Vamos a usar como ejemplo el fichero `Tut10-DatosPisaPais.csv`, de datos del estudio PISA, que hemos incluido en la sección previa. Nuestro primer paso es leer los datos usando la función `read.table`, que ya hemos usado en otras ocasiones para esto. Fíjate en que usamos la opción `sep = "\t"`, porque las columnas de datos se han separado con tabuladores en el fichero `csv`. ¿Cómo hemos sabido que se había usado el tabulador? Probando a leer los datos con el espacio como separador y descubriendo que se producía un error de lectura. Recuerda además que siempre es bueno comprobar que la lectura ha sido correcta, usando por ejemplo la función `head`:

```
datosPisa = read.table(file="../datos/Tut10-DatosPisaPais.csv", header=TRUE, sep = "\t")
head(datosPisa)
```

```
##                     CA    rpc pisa
## 1          Extremadura 15.394  461
## 2     Andaluc\303\255a 19.960  472
## 3               Murcia 18.520  462
## 4              Galicia 20.723  489
## 5             Asturias 21.035  500
## 6 Castilla Le\303\263n 22.289  509
```

A partir de aquí, las cosas son sencillas. Usamos la notación habitual de los `data.frmaes` de R para trabajar con las dos columnas que contienen los datos que vamos a representar y usamos la función `plot` así:

```
plot(datosPisa$rpc, datosPisa$pisa)
```



Puedes probar también a usar

```
plot(pisa ~ rpc, data=datosPisa)
```

para ver que el resultado es el mismo. El símbolo `~`, que ya ha aparecido en algún tutorial previo, es la forma de decir en R que estamos estudiando la relación entre esas dos variables. En las próximas secciones volveremos a este ejemplo y seguiremos el análisis de regresión de esos datos. Pero antes, vamos a practicar en un ejercicio la lectura de datos a partir de ficheros `csv`.

Ejercicio 1. *En todos los casos se trata de leer un fichero* `csv` *que contiene pares de datos, para almacenar los valores de las variables x e y en sendos vectores de R, que llamaremos, claro está,* `x` *e* `y`*. Antes de empezar, guarda los ficheros en la carpeta* `datos` *de tu directorio de trabajo, y recuerda que conviene echarle un vistazo al fichero de datos, usando un editor de texto (como el* Bloc de Notas *de Windows).*

1. *El primer fichero es muy sencillo, no debe suponer problemas:*
   tut10-ejercicio01-1.csv
2. *Una variante del anterior, los mismos datos, con algunos cambios de formato:*
   tut10-ejercicio01-2.csv
3. *Y una tercera versión de los mismos datos:*
   tut10-ejercicio01-3.csv

Soluciones en la página 40. □

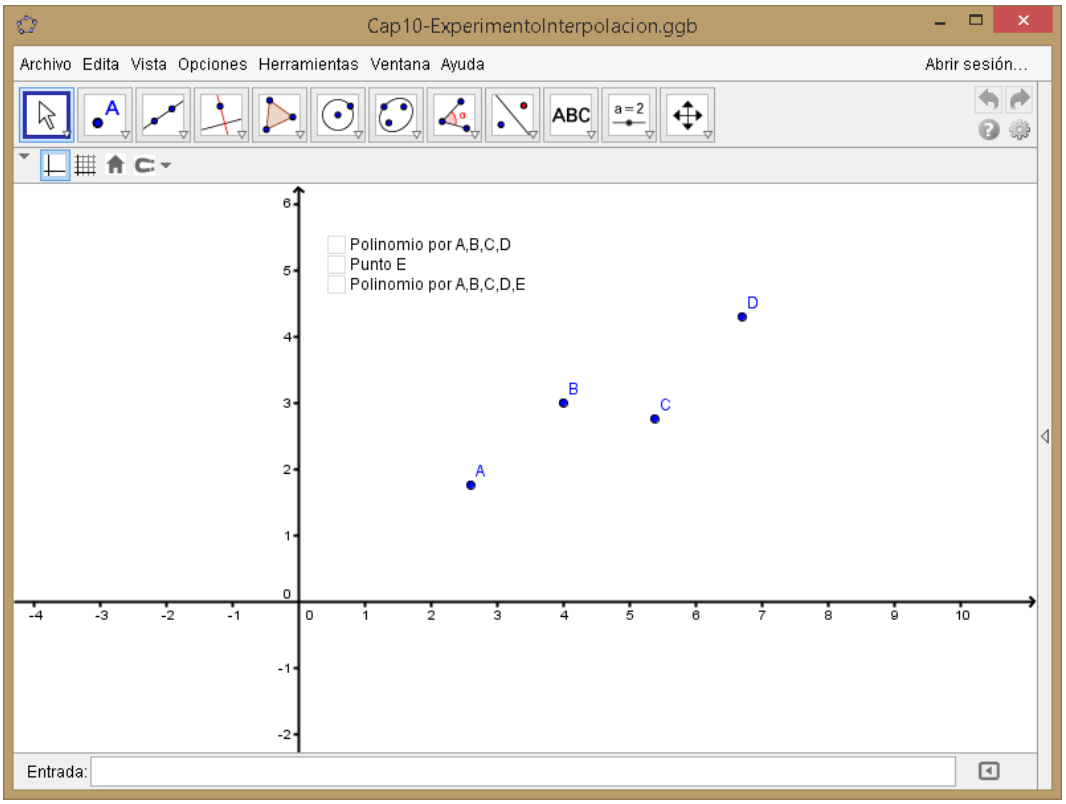
2. La recta de regresión lineal.

2.1. Interpolación en GeoGebra.

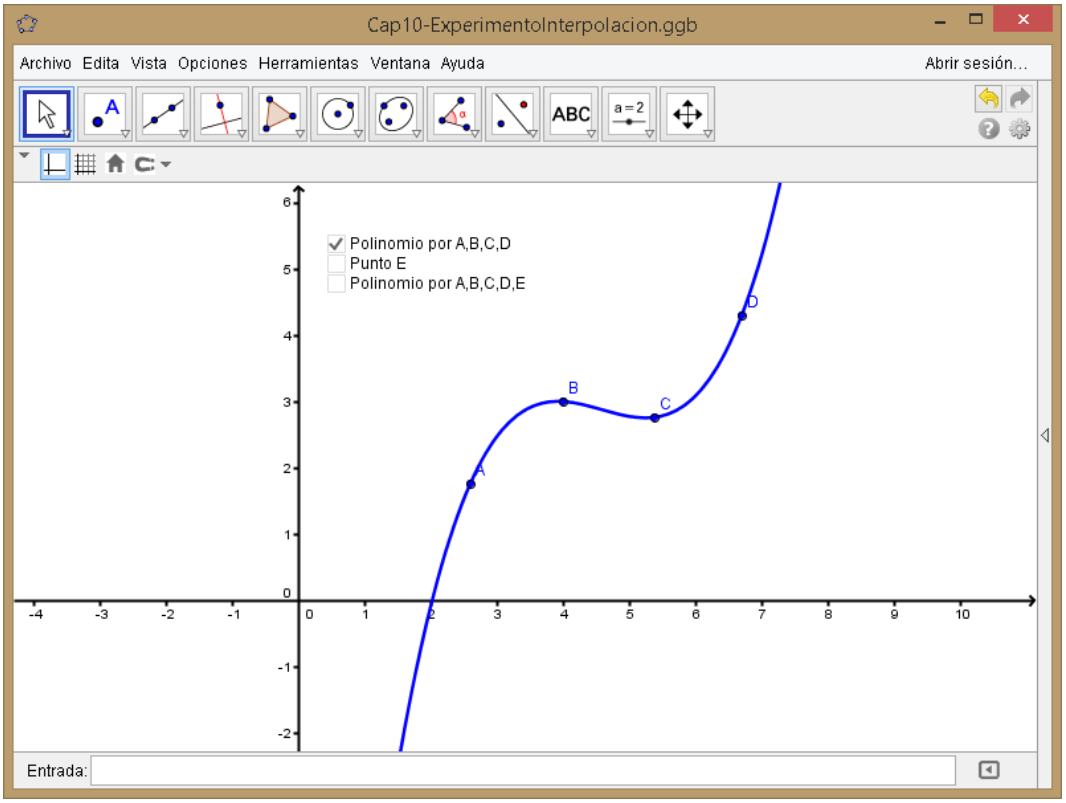
Dado un conjunto de puntos, representados mediante su diagrama de dispersión, nuestro primer paso en la Sección 10.1.1 del libro (pág. 348) ha sido la búsqueda de una curva que pasara por todos y cada uno de los puntos del diagrama de dispersión. La técnica llamada *interpolación* permite construir una curva polinómica con esas propiedades. Para que puedas experimentar con esa idea hemos incluido un fichero GeoGebra:

Cap10-ExperimentoInterpolacion.ggb

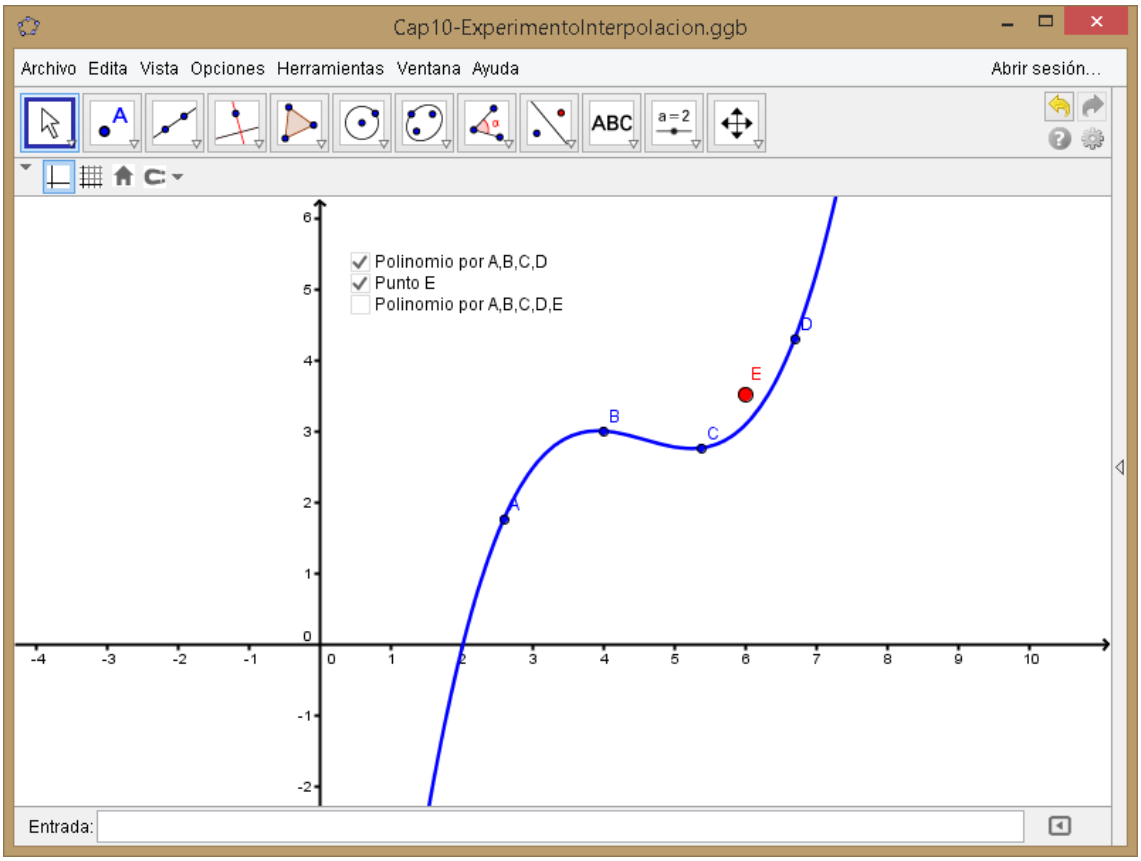
que al abrirlo te mostrará esta imagen:



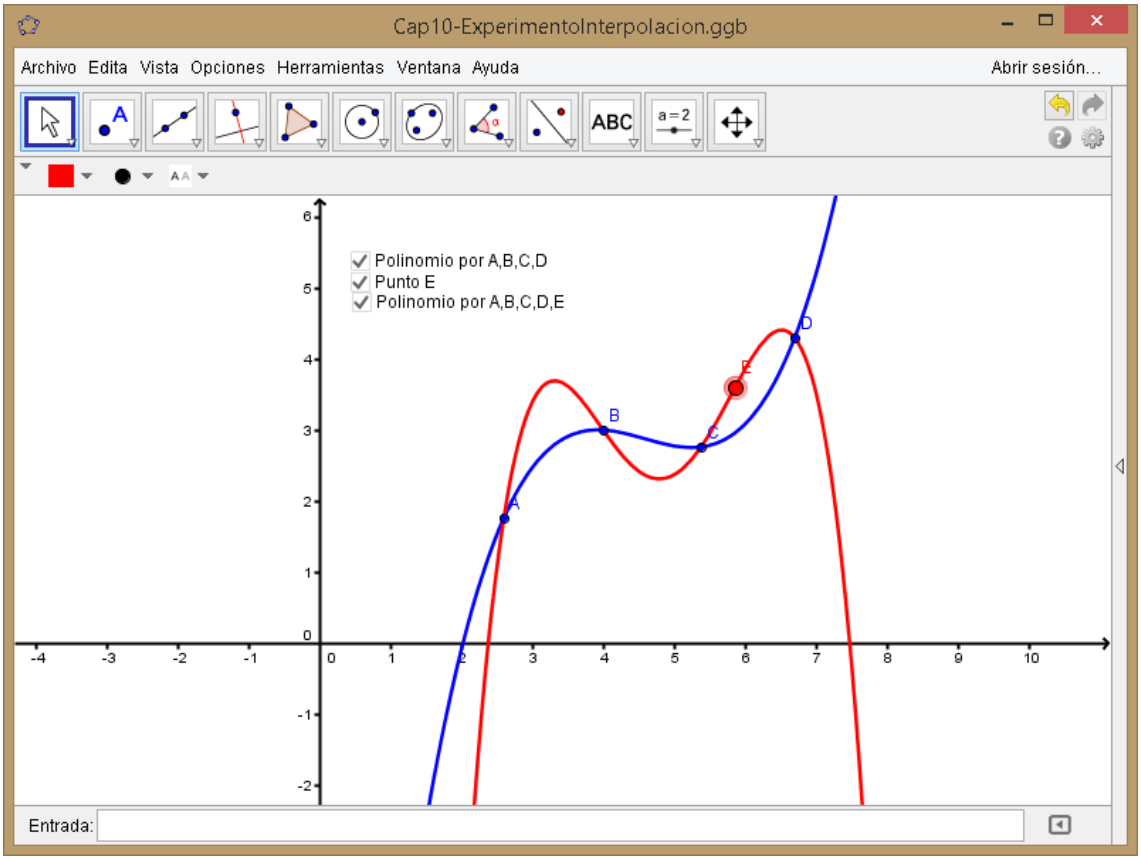
Empieza por marcar la casilla *Polinopmio por A, B, C, D* para ver aparecer la curva que pasa por esos puntos:



Si quieres, puedes probar a desplazar alguno de los puntos para ver cómo responde la curva. También puedes marcar la casilla para añadir un punto E adicional que, si no has movido los puntos, inicialmente aparecerá cerca de la curva:



Finalmente, marca la casilla restante para mostrar la curva de interpolación que pasa por los cinco puntos y prueba a mover el punto E. Verás como incluso pequeñas modificaciones de la posición del punto pueden tener una influencia muy importante en la posición de esa curva.



El comando de GeoGebra que nos permite construir la curva que pasa por los puntos A, B, C y D es: `Polinomio[A, B, C, D]`.

2.2. Recta de regresión en GeoGebra.

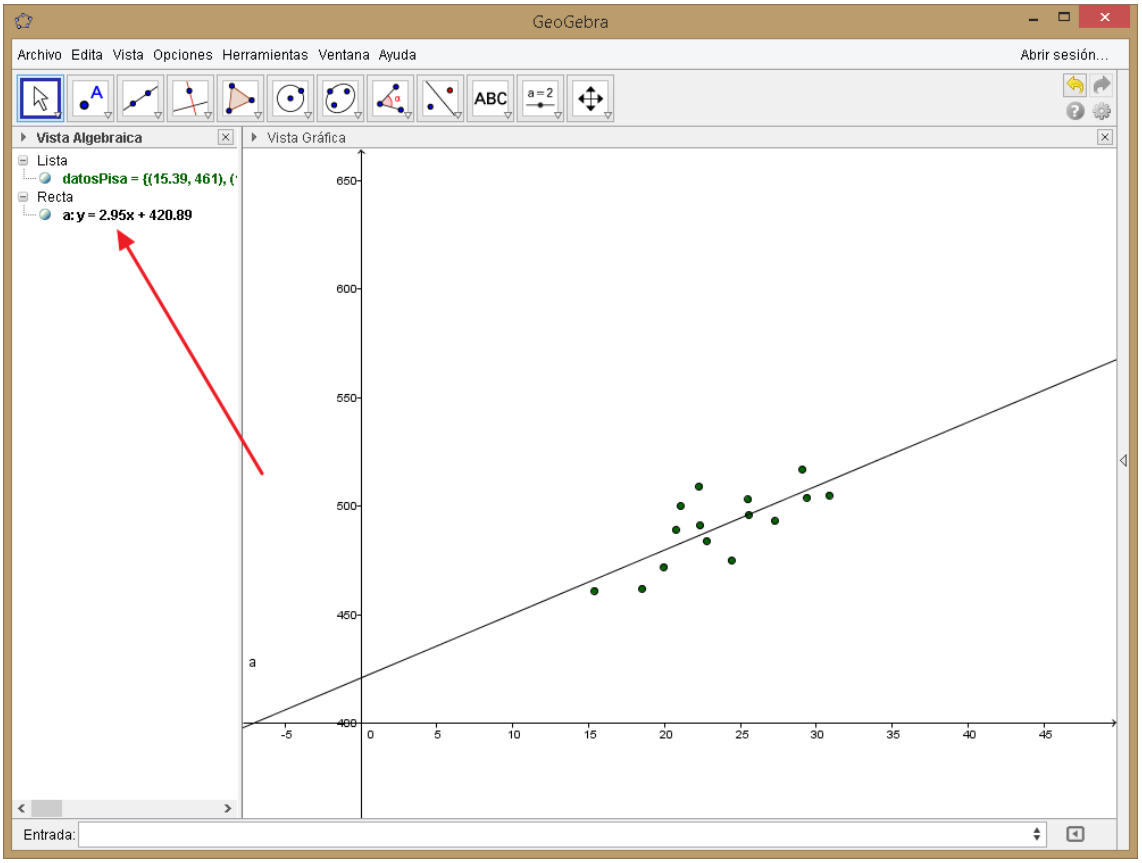
El siguiente paso, como hemos discutido en el libro, es renunciar a buscar una curva que pase por todos los puntos. En su lugar buscamos la mejor recta posible para representar ese conjunto de puntos. En GeoGebra, la recta de regresión lineal se obtiene con la función `AjusteLineal`, aplicada a una *lista* de puntos. Si todavía tienes abierto el fichero que hemos usado para la interpolación, puedes ejecutar este comando

```
AjusteLineal[{A, B, C, D}]
```

para obtener esa recta (si no usas las llaves el comando funcionará igualmente, GeoGebra las añade por ti). Y para los datos del estudio PISA que hemos usado al principio del tutorial, basta con hacer

```
AjusteLineal[datosPisa]
```

para obtener la recta:



Hemos destacado la ecuación de la recta en la *Vista Algebraica* de GeoGebra, para que veas que ahí aparecen la pendiente y la ordenada en el origen.

Ficheros de GeoGebra para visualizar las ideas asociadas a la recta de regresión.

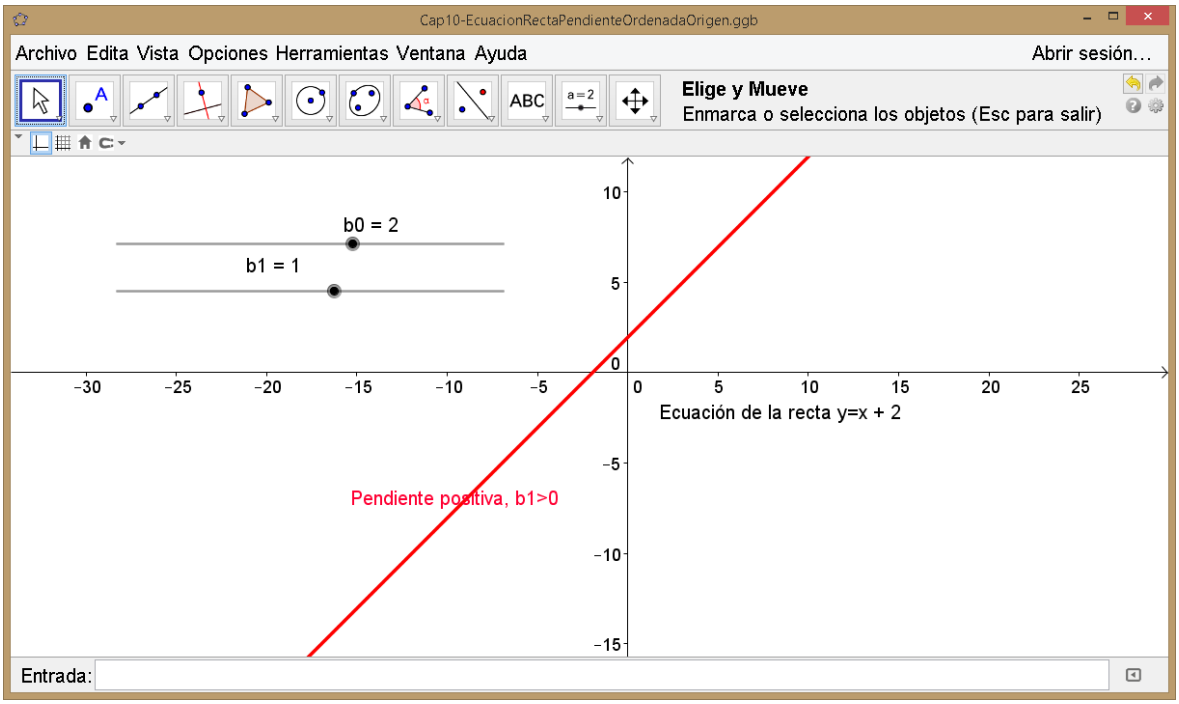
Vamos a aprovechar este apartado para incluir aquí varios ficheros de GeoGebra que acompañan la discusión de la Sección 10.2 del libro (pág. 352). Para empezar, el fichero

Cap10-EcuacionRectaPendienteOrdenadaOrigen.ggb

permite refrescar el significado de la pendiente b_1 y la ordenada en el origen b_0 de una recta cuya ecuación es

$$y = b_0 + b_1 x,$$

usando un par de deslizadores como se ilustra en la figura

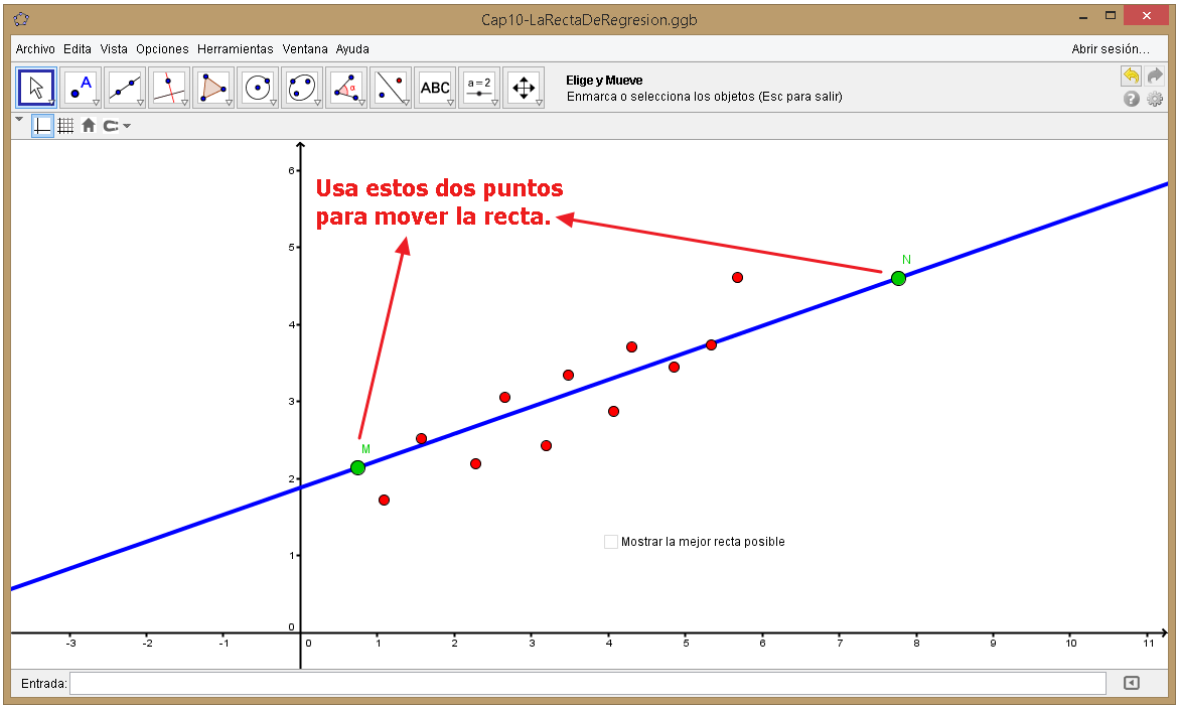


Prueba a mover esos deslizadores hasta asegurarte de que comprendes bien el efecto que cada uno de los coeficientes b_0 y b_1 tiene sobre la recta.

Una vez entendemos de las herramientas (los valores b_0, b_1) que nos van a permitir colocar la recta donde nosotros queramos, llega el momento de elegir la mejor recta para un conjunto dado de puntos. El fichero

Cap10-LaRectaDeRegresion.ggb

te presenta una colección o nube de puntos (de color rojo) y te permite usar los puntos M y N para tratar de colocar la recta azul de la manera que consideres más representativa de la nube de puntos dada.

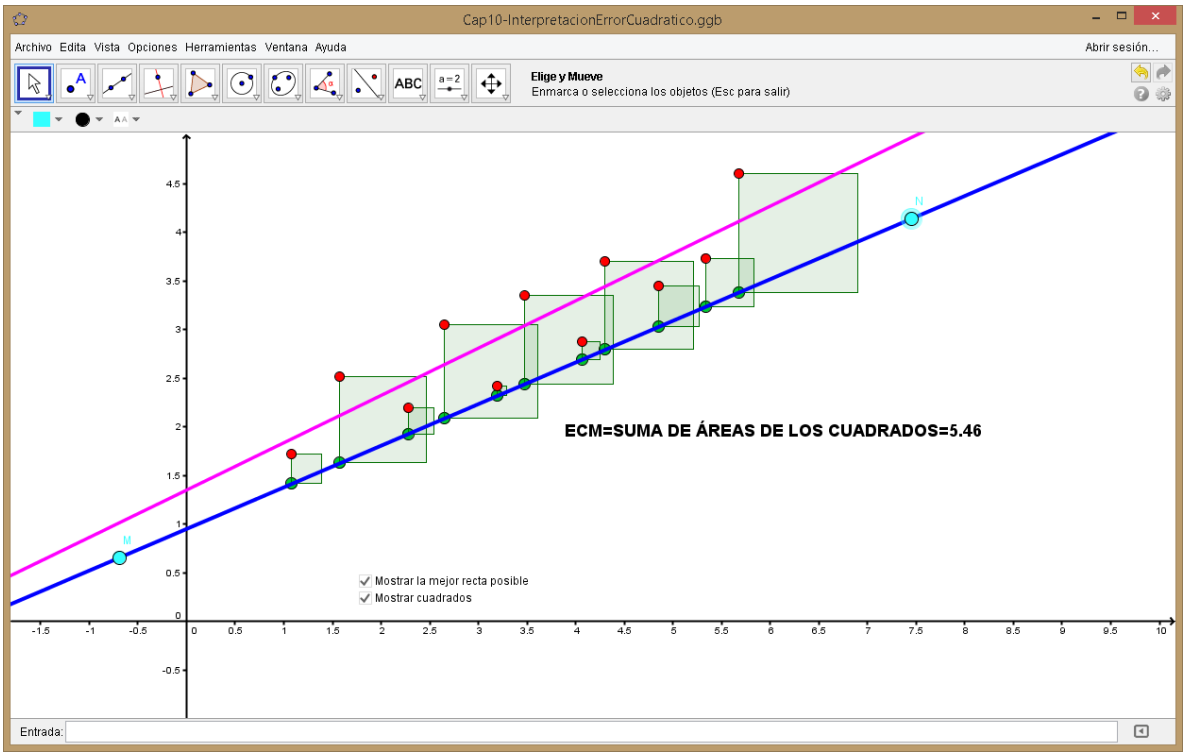


Una vez hecho esto, usa la casilla para mostrar la recta de regresión calculada por GeoGebra y comprobar si te has acercado a ese objetivo.

El siguiente paso consiste en cuantificar, para medirlo de forma precisa, el error que se comete al usar una recta como representante de la nube de puntos. El criterio que usamos es el error cuadrático, y el fichero Geogebra

Cap10-InterpretacionErrorCuadratico.ggb

trata de ilustrar la noción de error cuadrático. Como en el caso anterior, se muestra una nube de puntos y una recta que podemos tratar de colocar de la mejor manera posible. En este caso además de una casilla para mostrar la recta de regresión disponemos de otra que nos permite ver los cuadrados que determinan ese error *cuadrático*. Prueba a mover la recta hacia la posición que ocupa la recta de regresión y observa como la suma total de las áreas de esos cuadrados se reduce a un mínimo en esa posición.



Cuando mueves la recta hasta hacerla coincidir exactamente con la recta de regresión, este fichero también permite visualizar los valores de los *residuos*, que son los segmentos verticales que conectan los puntos de la nube inicial de datos (los puntos rojos) con los puntos correspondientes de la recta de regresión (los puntos verdes).

Vamos a cerrar este apartado con dos ficheros GeoGebra, que permiten explorar las ideas que aparecen en las Figuras 10.10 y 10.11 del libro (págs. 362 y 363, respectivamente). Los ficheros son:

Cap10-ZoomEnParabola.ggb
Cap10-RectasZoomHaciaFuera.ggb

En ambos casos, se trata de hacer zoom en la figura, hacia dentro en la primera y hacia fuera en la segunda. Puedes usar la rueda del ratón o las teclas `Ctrl +` y `Ctrl -` para hacer esto.

2.3. La recta de regresión en R.

En esta sección vamos a comenzar el estudio de los comandos de R necesarios para el análisis de regresión lineal. Las posibilidades de R van mucho más lejos de lo que vamos a ver aquí, pero, como siempre, es necesario empezar por lo más básico.

Para empezar a trabajar vamos a usar los datos del informe PISA con los que abríamos este tutorial. Supondremos que los datos están disponibles en el `data.frame` de R llamado `datosPisa` que hemos creado en la Sección 1.2 (pág. 4). En primer lugar, vamos a ver como calcular la covarianza de estos dos vectores, cosa que es extremadamente fácil de hacer en R:

```
cov(datosPisa$rpc, datosPisa$pisa)
```

```
## [1] 55.261
```

Recuerda que en R **se trata siempre de la covarianza muestral**, que usa $n-1$ en el denominador. Si deseas la covarianza que usa n, debes usar el truco habitual, multiplicando por $n-1$ y dividendo por n.

Con la función `cov` estamos listos para calcular la pendiente de la recta de regresión, que de acuerdo con la Ecuación 10.8 (pág. 359) del libro es:

```
(b1 = cov(datosPisa$pisa, datosPisa$rpc)/var(datosPisa$rpc))
```

```
## [1] 2.951
```

Y la ordenada en el origen es, entonces:

```
(b0 = mean(datosPisa$pisa) - b1 * mean(datosPisa$rpc))
```

```
## [1] 420.89
```

Así que la recta de regresión es, aproximadamente,

$$y = 420.9 + 2.951 \cdot x$$

Recuerda que ya habíamos obtenido esta recta en Geogebra, así que es un buen momento para que compruebes que el resultado es el mismo.

La función `attach`.

Si te has cansado de escribir todo el rato

`datosPisa$`,

puedes ejecutar

```
  attach(datosPisa)
```

Cuando lo ejecutes puede que veas algún mensaje de advertencia de R. No te preocupes por el momento. Con este comando le decimos a R que queremos acceder a las variables de ese `data.frame` sin necesidad de precederlas con `datosPisa$`. Por ejemplo, en lugar de `datosPisa$rpc` ahora puedes decir simplemente

```
rpc
```

```
##  [1] 15.394 19.960 18.520 20.723 21.035 22.289 22.341 22.772 24.393 25.508
## [11] 25.540 27.248 29.071 29.385 30.829
```

y, como ves, R lo entiende sin problemas. El problema de hacer esto es que en ocasiones se crean ambigüedades entre las variables del `data.frame` y otras posibles variables. Especialmente en programas más complicados y cuando se trabaja con varios `data.frames` a la vez. En un programa sencillo que escribimos para un cálculo corto, puede ser la solución más cómoda. Pero el uso generalizado de `attach` no es una práctica muy recomendable y la mayoría de programadores expertos de R lo desaconsejan. Cuando aprendas un poco más de R podrás evitarlo casi siempre, mediante herramientas como la función `with` y la opción `data` de muchas funciones. Además, el uso del tabulador en RStudio aligera mucho el trabajo adicional que esto supone. Nosotros vamos a seguir adelante en el Tutorial sin usar `attach`. La forma de deshacer sus efectos es mediante

```
detach(datosPisa)
rpc
```

```
## Error in eval(expr, envir, enclos): objeto 'rpc' no encontrado
```

y el mensaje de error nos confirma de que hemos vuelto a la situación previa al uso de `attach`.

Añadir la recta de regresión al diagrama de dispersión.

Naturalmente, una vez calculada la recta de regresión, queremos verla incluida en el diagrama de dispersión. Para ello, disponemos de la función `abline`, que puede dibujar cualquier recta. El nombre de esa función se debe a que tradicionalmente las rectas se han escrito así:

$$y = a + b \cdot x$$

a partir de su pendiente b y ordenada en el origen a. Para usarlo en nuestro ejemplo hacemos: y el resultado es la gráfica que se muestra en la Figura 2.

```
    plot(datosPisa$pisa ~ datosPisa$rpc)
    abline(a = b0, b = b1)
```

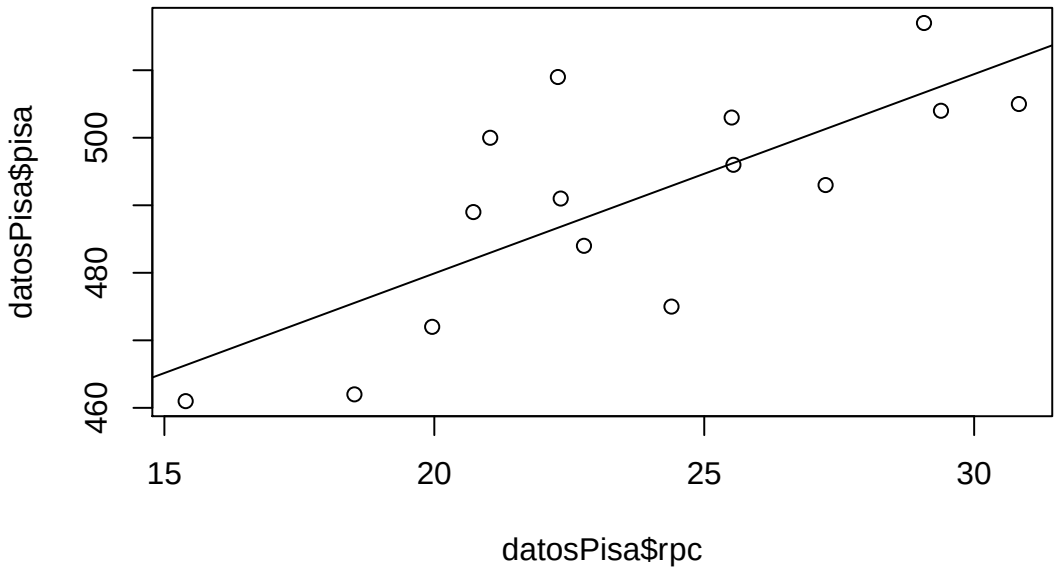


Figura 2: Recta de regresión en el diagrama de dispersión para el Ejemplo de los datos del informe PISA.

Está claro que, como hemos dicho antes, se trata de un gráfico muy sencillo, aunque suficiente para muchos de nuestros propósitos. Pronto aprenderemos a mejorarlo. Recuerda además que ya obtuvimos este gráfico con GeoGebra, así que es una buena idea que compares las dos versiones.

2.4. Coeficiente de correlación, Residuos y error cuadrático medio

El coeficiente de correlación r que aparece en la Ecuación 10.16 (pág. 380) del libro, se calcula en R con la función `cor`. Así que, para los datos del estudio PISA se obtiene

```
cor(datosPisa$pisa, datosPisa$rpc)

## [1] 0.75277
```

Ejercicio 2.

1. *Comprueba que el resultado de* `cor` *es el mismo que cuando se usa* `cov` *y la fórmula de la Ecuación 10.16 del libro (pág. 380). Recuerda que la función* `sd` *sirve para calcular las cuasidesviaciones típicas que necesitarás para el denominador de esa fórmula.*

2. *Averigua cuál es la función de GeoGebra que te permite calcular el coeficiente de correlación de la lista de puntos* `datosPisa` *que hemos creado al principio del tutorial. Comprueba que el resultado es el mismo que hemos obtenido en R.*

Soluciones en la página 41. □

Residuos y error cuadrático medio

Una vez que hemos obtenido la ecuación de la recta de regresión, también podemos usarla para calcular otros valores asociados al análisis de regresión como los residuos, o el error cuadrático, que son relevantes para el análisis de la varianza en el modelo de regresión. En la Sección 3 de este tutorial vamos a aprender a usar la función `lm` de R para hacer esta operación de una manera mucho más sencilla. Pero es bueno empezar por lo más básico, construyendo los resultados paso a paso.

Por ejemplo, para calcular los residuos con los datos del estudio PISA debemos primero calcular la diferencia entre los valores para las pruebas PISA que predice la recta, calculados a partir de `rpc`, y los valores de `pisa` que se han medido en la muestra. Para obtener los valores que predice la recta hacemos

```
(pisaRecta = b0 + b1 * datosPisa$rpc)

##  [1] 466.32 479.79 475.54 482.05 482.97 486.67 486.82 488.09 492.88 496.17
## [11] 496.26 501.30 506.68 507.61 511.87
```

que, como se ve, da como resultado el vector de valores predichos (*fitted values*, en inglés). Ahora sólo tenemos que restar estos de `pisa` para obtener los residuos:

```
(residuos = datosPisa$pisa - pisaRecta)

##  [1]  -5.31931  -7.79347 -13.54407   6.95493  17.03423  22.33370   4.18025
##  [8]  -4.09162 -17.87515   6.83451  -0.25992  -8.30019  10.32018  -3.60643
## [15]  -6.86764
```

Si lo que queremos es el error cuadrático EC, basta con hacer la suma de los cuadrados de los residuos:

```
(EC = sum(residuos^2))

## [1] 1745.9
```

Como hemos dicho, en la Sección 3 de este tutorial veremos como se pueden obtener muchos de estos valores usando `lm`. Pero antes, y para que puedas hacer fácilmente los siguientes ejercicios, vamos a incluir aquí un fichero de comandos R que sirve para realizar todos los pasos anteriores:

Tut10-RectaRegresion.R

Ejercicio 3. *Vamos a usar ese fichero de R para comprobar los valores que aparecen en algunos ejemplos del Capítulo 10. Hemos incluido ficheros* `csv` *para facilitarte las operaciones. En todos los casos, lo más recomendable es que repitas un análisis completo, ejecutando todos los comandos del fichero para cada conjunto de datos.*

1. *Ejemplo 10.2.1 (pág 359). Datos en el fichero:*
   cap10-EjemploRegresion01.csv
2. *Ejemplo 10.3.1 (pág 368). Datos en el fichero:*
   Cap10-EjemploRectaMalaAproximacion01.csv.
3. *Ejemplo 10.3.2 (pág 369). Datos en el fichero:*
   Cap10-EjemploRectaMalaAproximacion02.csv.
4. *Ejemplo 10.3.3 (pág 372).*
   Primero los puntos "no ruidosos". Datos en el fichero:
   Cap10-Ejemplo-Anova01.csv.
5. *Y del mismo ejemplo 10.3.3, ahora los puntos "ruidosos". Datos en el fichero:*
   Cap10-Ejemplo-Anova02.csv.

□

2.5. Un gráfico de regresión más expresivo

Opcional: esta sección puede omitirse en una primera lectura.

El gráfico de regresión que hemos mostrado en la Figura es bastante elemental, incluso rudimentario. Aquí vamos a ver algunos comandos y opciones de R que ayudan a conseguir un resultado visualmente más atractivo. Puedes considerar esta sección como una invitación a explorar más en las (muy amplias) capacidades gráficas de R, de las que aquí sólo mostramos una ínfima parte.

Para empezar, hemos cambiado el comando

```
plot(datosPisa$rpc, datosPisa$pisa)
```

por una versión más detallada. Ocupa varias líneas que, como siempre en estos casos, deben ejecutarse conjuntamente:

```
plot(datosPisa$rpc, datosPisa$pisa,
    lwd=2, col="red", cex=1.1, cex.lab=1.1, cex.axis=1.1, bty="n",
    xlab="Renta per capita, a<U+00F1>o 2012, en miles de euros",
    ylab="Puntos PISA2012 en Matem<U+00E1>ticas")
```

y que produce el resultado que aparece en la Figura 3. Vamos a comentar las modificaciones que hemos hecho:

- La opción `lwd=2` sirve para que los puntos (x_i, y_i) se dibujen con un trazo más grueso (de hecho, `lwd` proviene de *line width*, grosor de la línea).
- La opción `col="red"` ya ha aparecido antes en los tutoriales, y sirve para cambiar el color de los símbolos que usa R, en este caso para que los puntos (x_i, y_i) sean de color rojo.
- Las opciones que empiezan por `cex` se refieren al tamaño de fuentes tipográficas y de algunos símbolos empleados en el gráfico. El nombre `cex` proviene de *character expansion*, y su valor es un factor multiplicativo que nos indica cuántas veces más grande es el tipo de fuente que se usa, tomando como referencia el tipo base por defecto que usaría R. Así pues, `cex=1.5` significa un símbolo 1.5 veces más grande que el valor por defecto, etc. Las variantes de `cex` que aparecen se refieren a distintas partes del gráfico. Así, por ejemplo, `cex.lab` se refiere al tamaño del texto en las frases que se usan como etiquetas de los ejes (en inglés *labels*, de ahí el nombre). En cambio `cex.axis` se refiere a los números que aparecen en las escalas de los ejes (en inglés, *axis*).
- La opción `bty="n"` (de *box type*, tipo de caja) sirve para decirle a R el tipo de marco que queremos que dibuje alrededor del gráfico. En este caso hemos optado por `n`, de *none*, ninguno, porque vamos a añadirlo después.
- Finalmente, los argumentos `xlab` e `ylab` sirven para añadir las etiquetas (de nuevo, *labels*) o frases que acompañan al eje x y al eje y, respectivamente.

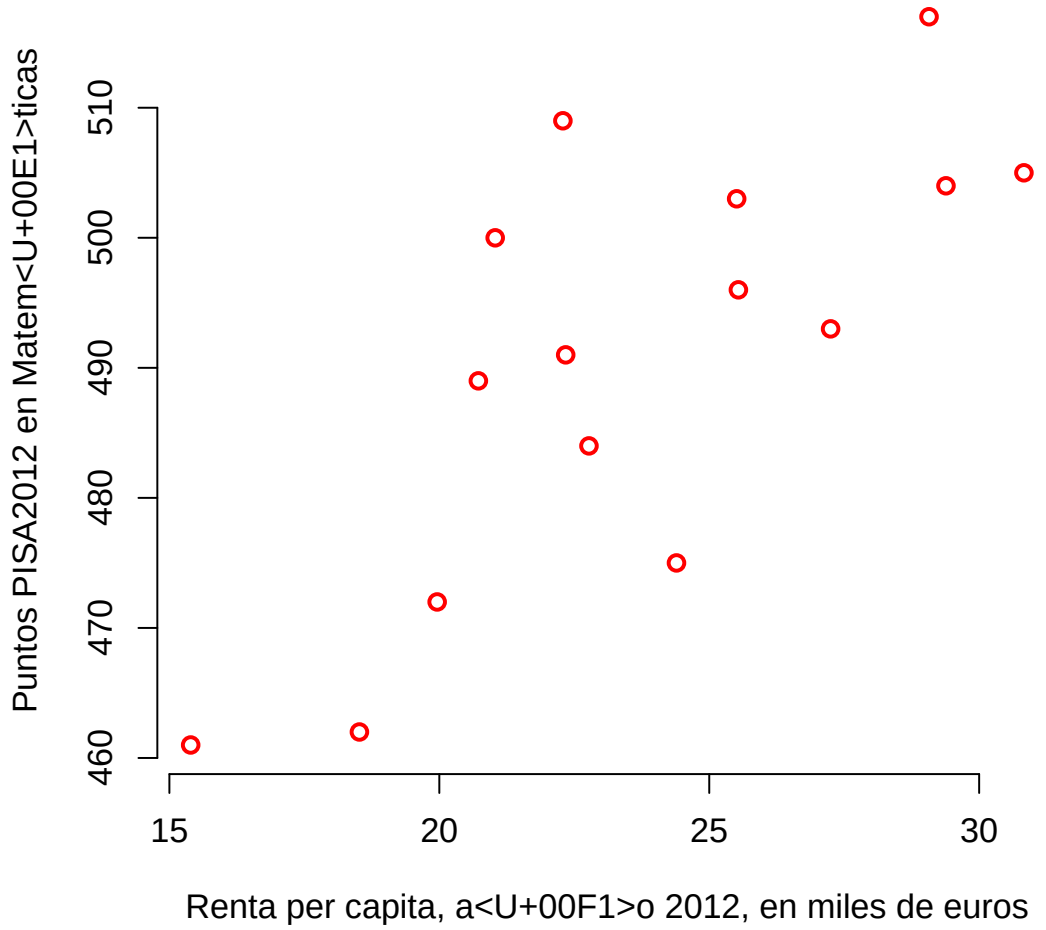


Figura 3: Primer paso para obtener un diagrama de dispersión mejorado para el ejemplo de los datos PISA.

Ejercicio 4.

1. *Prueba a usar la opción* `main` *para añadir un título general al gráfico.*
2. *¿Qué opción usarías para cambiar el tamaño de ese título?*
3. *Prueba a añadir la opción* `pch=18`*. ¿Qué ha cambiado? Busca información sobre los posibles valores de* `pch` *(por ejemplo pidiéndole a R que ejecute* `?pch`*).*

□

A continuación vamos a rotular cada punto del gráfico con el nombre de la región a la que corresponde. Esos nombres están disponibles en la primera columna del `data.frame datosPisa`. Para usarlos, ejecutamos este código , que enseguida comentaremos:

```
par(xpd=TRUE)
text(datosPisa$rpc, datosPisa$pisa, labels=datosPisa$CA,
    pos=3, offset=0.6,font=3)
par(xpd=FALSE)
```

Vamos a empezar por el comando `text`. Este comando sirve para colocar texto en cualquier posición del gráfico que estamos dibujando. Como de costumbre, podemos utilizar vectores como

argumentos. Así que en la opción `labels` usamos el vector `namesPisa` que contiene las etiquetas que queremos colocar junto a cada punto del gráfico. Los dos primeros argumentos de `text` son dos vectores, con las coordenadas x e y, respectivamente, de los puntos en los que se coloca cada una de esas etiquetas; en nuestro caso, se trata de los vectores `rpc` y `pisa`. El argumento `pos` (de *position*) sirve para colocar el texto debajo, a la izquierda, encima o a la derecha del punto, según que `pos` valga 1, 2, 3 o 4. La opción `offset` controla la separación entre el punto y el texto. Por último, la opción `font` permite elegir entre un tipo normal (1), negrita (2), itálica (3) o negrita itálica (4).

¿Para qué sirve el comando `par(xpd=TRUE)`? La función `par` (de *parameter*, parámetro) es una de las funciones más importantes cuando se trata de modificar un gráfico de R. En este caso la usamos porque algunos de los nombres del vector `namesPisa` son demasiado largos, y R "recorta" esos textos de las etiquetas para que no sobresalgan del marco del gráfico. Nosotros preferimos que los textos se muestren completos, aunque sobresalgan un poco, así que hemos usado la opción `xpd=FALSE` que le dice a R que no haga ese recorte. Una vez colocadas las etiquetas volvemos a activar esa opción con `par(xpd=FALSE)`. El resultado de estas operaciones es el gráfico de la Figura 4.

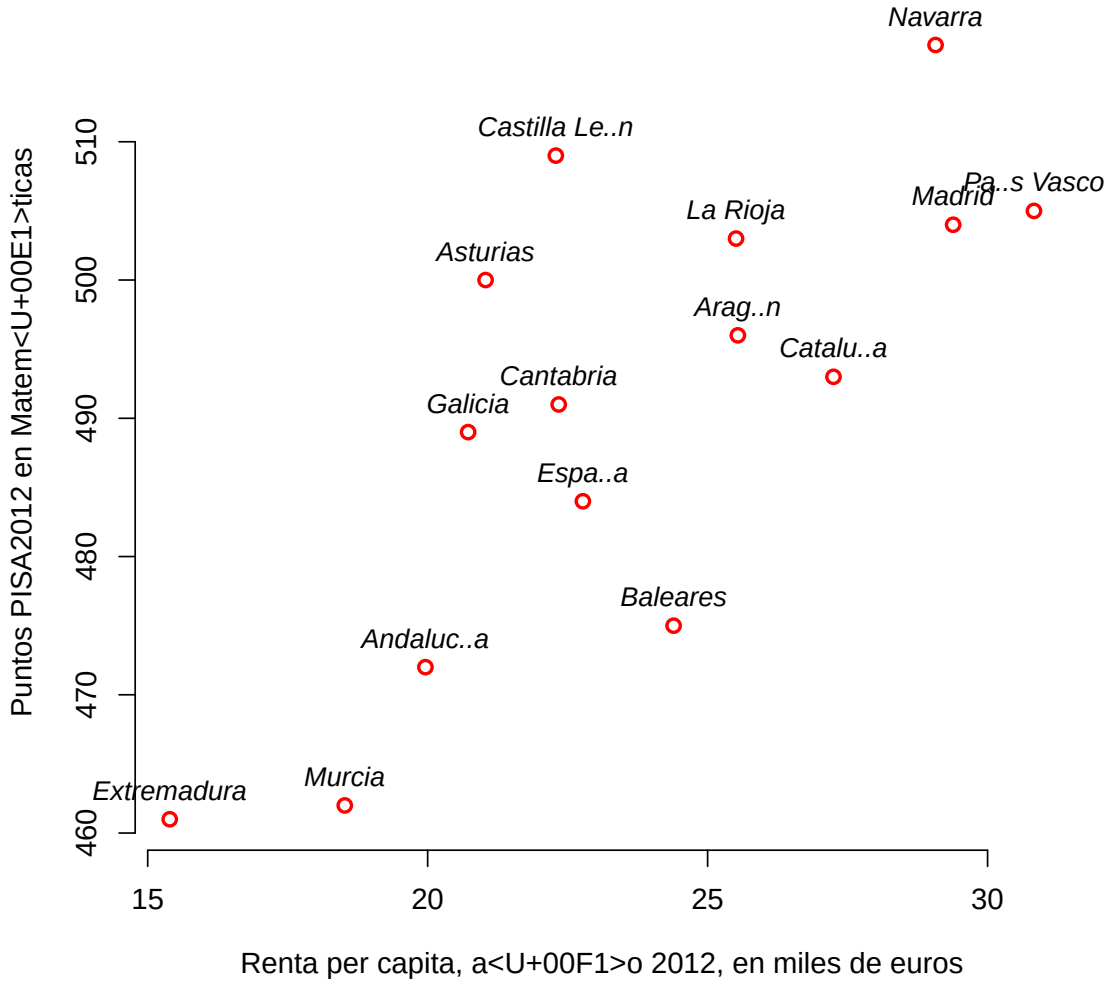


Figura 4: Segundo paso para obtener un diagrama de dispersión mejorado.

A continuación añadimos la recta con

```
abline(b0, b1, lwd=3, col="blue")
```

Las únicas modificaciones que hemos hecho son sobre el grosor y color del trazo. Ahora, para hacer visible la idea de residuo, vamos a añadir unos segmentos que conecten cada punto (x_i, y_i) de la muestra con el correspondiente punto de la recta, que es $(x_i, \hat{y}_i)$. Para eso usamos el comando:

```
segments(datosPisa$rpc, pisaRecta, datosPisa$rpc, datosPisa$pisa,
     lty=2,lwd=3)
```

cuyos dos primeros argumentos son las coordenadas x e y de los puntos iniciales de los segmentos, y cuyos argumentos tercero y cuarto son las coordenadas de los puntos finales de los segmentos. Fíjate en que los valores $\hat{y}_i$ están en el vector `pisaRecta` que hemos calculando antes. Hemos ajustado además el grosor de la línea con `lwd`, y el tipo de trazo, para que sea discontinuo, mediante la opción `lty` (de *line type*).

Ejercicio 5. *Busca qué otros tipos de trazo puedes usar con distintos valores de* `lty`.

□

Como última modificación hemos dibujado un marco o "caja" alrededor del gráfico, usando el comando

```
box(lwd=3,bty="l")
```

La opción `bty` (de *box type*, tipo de caja), le indica a R como queremos que sea la forma del marco. Los códigos que se usan en este caso, son curiosos: se usa una letra, cuya forma indica la de la caja. Por ejemplo con `bty="o"` le indicamos a R que queremos una caja con cuatro lados (la letra o es un círculo completo , mientras que con `bty="u"` indicamos una caja a la que le falta el lado de arriba. Ya vimos antes que la opción "n"suprime la caja. Las otras opciones disponibles son: `"l", "7", "c", "u", "[", "]"`

Ejercicio 6. *Pruébalas todas para ver el efecto que producen.* □

El resultado final es el gráfico de la Figura 5. Como hemos dicho, hay mucho más que aprender sobre los gráficos de R, y aquí apenas hemos esbozado algunas ideas básicas. El lector interesado encontrará mucha información en la red, y en la abundante bibliografía, de la que citamos como ejemplo el libro (¡de más de 400 páginas!) *R Graphics Cookbook*, de Winston Chang, editado por O'Reilly (ISBN: 978-1-449-31695-2).

3. Introducción a la función `lm` de R

Opcional: esta sección puede omitirse en una primera lectura. Pero su contenido es muy importante para seguir avanzando en el aprendizaje de R.

La función `lm`, de *linear model* (modelo lineal) es, sin exageración alguna, una de las funciones más importantes de R. Una de las tareas más importantes de la Estadística y el Análisis de Datos es precisamente la construcción de modelos para describir la relación entre variables. Y, dentro de los posibles modelos, la familia de modelos lineales es la más importante. Esos modelos lineales se construyen en R usando la función `lm`, así que podemos decir que esta función abre la puerta a algunas de las posibilidades más avanzadas del uso de R para la modelización estadística.

Pero empecemos por el modelo más sencillo, el modelo de regresión lineal simple que hemos descrito en el Capítulo 10. Vamos a usar como ejemplo los puntos con "ruido" del Ejemplo 10.3.3 del libro (pág. 372), que tienes en el fichero `csv` llamada `Cap10-Ejemplo-Anova02.csv`, que ya hemos usado antes en los ejercicios de la página 14 de este tutorial. De hecho, puedes usar los resultados del ejercicio 5 de esa página para compararlos con los que obtendremos aquí.

Empezamos leyendo los puntos de la muestra contenidos en ese fichero:

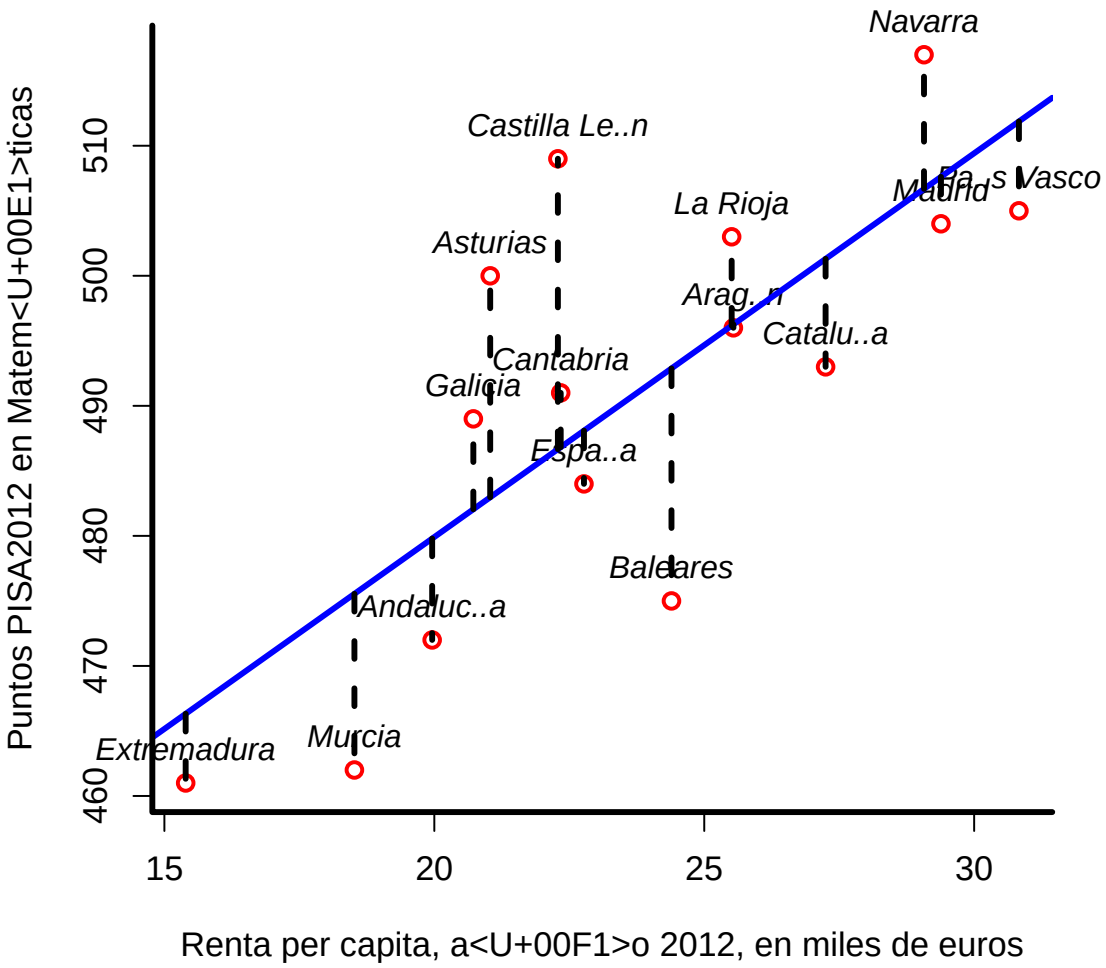


Figura 5: El diagrama de dispersión mejorado para el Ejemplo **??**.

```
datos = read.table("../datos/Cap10-Ejemplo-Anova02.csv", header=TRUE)
```

A continuación para explicarle a R que queremos usar el modelo de regresión lineal correspondiente a esos datos basta con ejecutar este comando:

```
(lmXY = lm(y ~ x, data=datos))

##
## Call:
## lm(formula = y ~ x, data = datos)
##
## Coefficients:
## (Intercept)            x
##       0.983       -0.481
```

La sintaxis `y ~ x` es la forma que tiene R de expresar que `x` es la variable explicativa, e `y` la variable respuesta en este modelo. Hemos almacenado el modelo en la variable `lmXY` (el nombre podría ser cualquier otro) porque así será más fácil acceder a las propiedades de este modelo, como vamos a ver enseguida. La salida de este comando contiene, bajo el nombre `coefficients`, los valores de b_0 y b_1. Concretamente, el valor de b_1, la pendiente, aparece bajo `x`, porque b_0 es el coeficiente que

acompaña a la x en la ecuación

$$y = b_0 + b_1 \cdot x$$

de la recta. El valor de b_0 aparece bajo `(Intercept)` porque ese es el nombre que se le da, en inglés, a la ordenada en el origen.

A primera vista, parece que no hemos ganado gran cosa, aparte de una forma rápida de llegar a los coeficientes de la recta. Pero en realidad el modelo, creado mediante `lm`, tiene asociadas muchas propiedades. Para ver algunas de ellas utiliza este comando:

```
summary(lmXY)
```

```
##
## Call:
## lm(formula = y ~ x, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.01363 -0.01164 -0.00158  0.00744  0.02287
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.9828     0.0160    61.6  5.4e-12 ***
## x            -0.4808     0.0208   -23.1  1.3e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0138 on 8 degrees of freedom
## Multiple R-squared:  0.985,Adjusted R-squared:  0.983
## F-statistic:  536 on 1 and 8 DF,  p-value: 1.29e-08
```

Como ves, R ha calculado muchas más propiedades de las que parecía. De hecho, insistimos, al usar `summary` sólo estamos viendo una parte de las propiedades disponibles.

La ventaja de haber guardado el modelo en la variable `lmXY` es que ahora podemos usar la notación con `lmXY$` para acceder a esas propiedades. Prueba a escribir precisamente eso, `lmXY$` en RStudio, y pulsa el tabulador. Aparecerá una lista de posibilidades, de la que vamos a ver el valor de la primera opción que R nos ofrece:

```
lmXY$coefficients
```

```
## (Intercept)           x
##     0.98276    -0.48082
```

Como ves, `lmXY$coefficients` es un vector cuyos valores son b_0 y b_1. Así que puedes hacer, por ejemplo

```
b0 = lmXY$coefficients[1]
```

para guardar la ordenada en el origen en la variable `b0`.

Con esta notación `lm` es muy fácil acceder, por ejemplo, a los valores predichos y a los residuos del modelo. Basta con usar, respectivamente, los comandos

```
lmXY$fitted.values
```

```
##       1       2       3       4       5       6       7       8       9
## 0.86255 0.76158 0.63176 0.61733 0.60772 0.59810 0.58849 0.54521 0.53560
##      10
## 0.52598
```

y

```
lmXY$residuals
```

```
##           1           2           3           4           5           6
## -0.0136288  0.0152918  0.0063964  0.0228680 -0.0119545  0.0031498
##           7           8           9          10
## -0.0129175 -0.0062996 -0.0106943  0.0077887
```

Los valores predichos que aparecen aquí corresponden a los valores de la variable x en los puntos de la muestra. Enseguida veremos como usar `lm` para hacer predicciones en valores de x *que no aparecen en la muestra.*

Otra ventaja de usar `lm` es que muchas funciones de R reconocen un objeto de tipo *modelo lineal*, y responden de forma interesante ante ese tipo de objetos.

Ejercicio 7.

1. *¿Qué tipo de objeto de R es* `lmXY`*? Indicación: No es un* `vector`*, ni un* `data.frame`*, ni ninguno de los tipos con los que nos hemos encontrado en los tutoriales previos. Recuerda, ¿cuál es la función que se usa en R para preguntar por el tipo de objeto?*

2. *Para ver un ejemplo de lo que decimos, prueba a ejecutar estos comandos:*

   ```
   plot(x, y)
   abline(lmXY)
   ```

   Lo interesante, en este caso, es el segundo comando, que nos permite añadir la recta de regresión al gráfico sin necesidad de especificar los coeficientes.

□

A la vista del segundo apartado de este ejercicio, puede que te hayas preguntado si no se puede hacer, directamente,

```
plot(lmXY)
```

Y la respuesta es que sí se puede, pero el significado es otro. Lo que se obtiene no es lo que seguramente esperabas, sino una *serie de gráficos*, que son muy útiles para hacer el diagnóstico del modelo de regresión lineal simple, en el sentido que se discute en la Sección 10.4.2 (pág. 389) del libro. Los vamos a discutir con más detalle en la Sección

Usando `lm` para predecir valores.

Otro uso muy frecuente de la función `lm` es la predicción de valores usando el modelo de regresión lineal. Por ejemplo, para predecir el valor correspondiente a $x = 0.74$ en el ejemplo que estamos usando haríamos:

```
predict(lmXY, newdata = data.frame(x=0.74))
```

```
##       1
## 0.62695
```

El resultado es el mismo que si sustituyeras ese valor en la recta de regresión:

```
lmXY$coefficients[1] + lmXY$coefficients[2] * 0.74
```

```
## (Intercept)
##     0.62695
```

En este caso hemos usado `predict` para predecir un único valor, pero no hay nada que nos impida obtener la predicción para todo un vector de valores de x. Por ejemplo, para predecir usando todos los valores de x de 0.7 a 0.75, de décima en décima, haríamos:

```
predict(lmXY, newdata = data.frame(x=seq(0.70, 0.75, length.out = 6)))
```

```
##       1       2       3       4       5       6
## 0.64618 0.64138 0.63657 0.63176 0.62695 0.62214
```

Y te adelantamos que cuando aprendas más sobre R comprobarás que se puede usar `predict` con muchos otros tipos de modelos estadísticos, más allá del modelo de regresión lineal simple que estamos usando aquí.

Anova y `lm`

Para ver otro ejemplo de la función `lm` en acción, podemos usarla, en combinación con la función `anova` de R, para obtener el análisis de la varianza (y varios resultados más, asociados con ese análisis de la varianza, en los que no podemos entrar porque aún no tenemos el lenguaje):

```
(anovaLMxy = anova(lmXY))
```

```
## Analysis of Variance Table
##
## Response: y
##           Df Sum Sq Mean Sq F value  Pr(>F)
## x          1 0.1017  0.1017     536 1.3e-08 ***
## Residuals  8 0.0015  0.0002
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Como hemos dicho, en esta sección no queremos sino hacer una primera breve visita a la función `lm`. A medida que el lector vaya aprendiendo más Estadística y más sobre R, podemos asegurar que `lm` será un buen compañero de viaje, que iremos usando para operaciones cada vez más sofisticadas.

3.1. Regresión ortogonal con R.

En la Sección 10.2.2 del libro (pág. 364) hemos descrito otras posibilidades a la hora de elegir la mejor recta para representar a una colección de puntos (x_i, y_i), distintas de la que proporciona el método de mínimos cuadrados. En particular, hemos hablado de regresión ortogonal y de RMA. En R disponemos de la librería `lmodel2` con la que es muy fácil obtener las rectas de regresión correspondientes a estos modelos. Recuerda que debes instalar la librería antes de poder usarla.

Vamos a ilustrar el uso de esa librería con los datos del Ejemplo 10.2.2 del libro (pág. 366). Esos puntos se han fabricado con este código:

```
set.seed(2014)
(x = sort(rnorm(10, mean=5, sd=3)))
```

```
##  [1]  1.1368  3.3030  5.3758  5.8008  5.9631  5.9677  6.1991  6.3765
##  [9]  9.0597 11.4506
```

```
(y = x/2 + 1 + rnorm(10, mean=0, sd=3))
```

```
##  [1] 4.8271 2.7622 4.8517 3.4673 1.8192 6.7416 3.6033 9.7035 5.9516 8.7183
```

Y podemos usar la librería `lmodel2` para obtener las rectas de regresión correspondientes a los distintos modelos:

```
library(lmodel2)
(lm2XY = lmodel2(y~x,range.y="relative", range.x="relative" ))
```

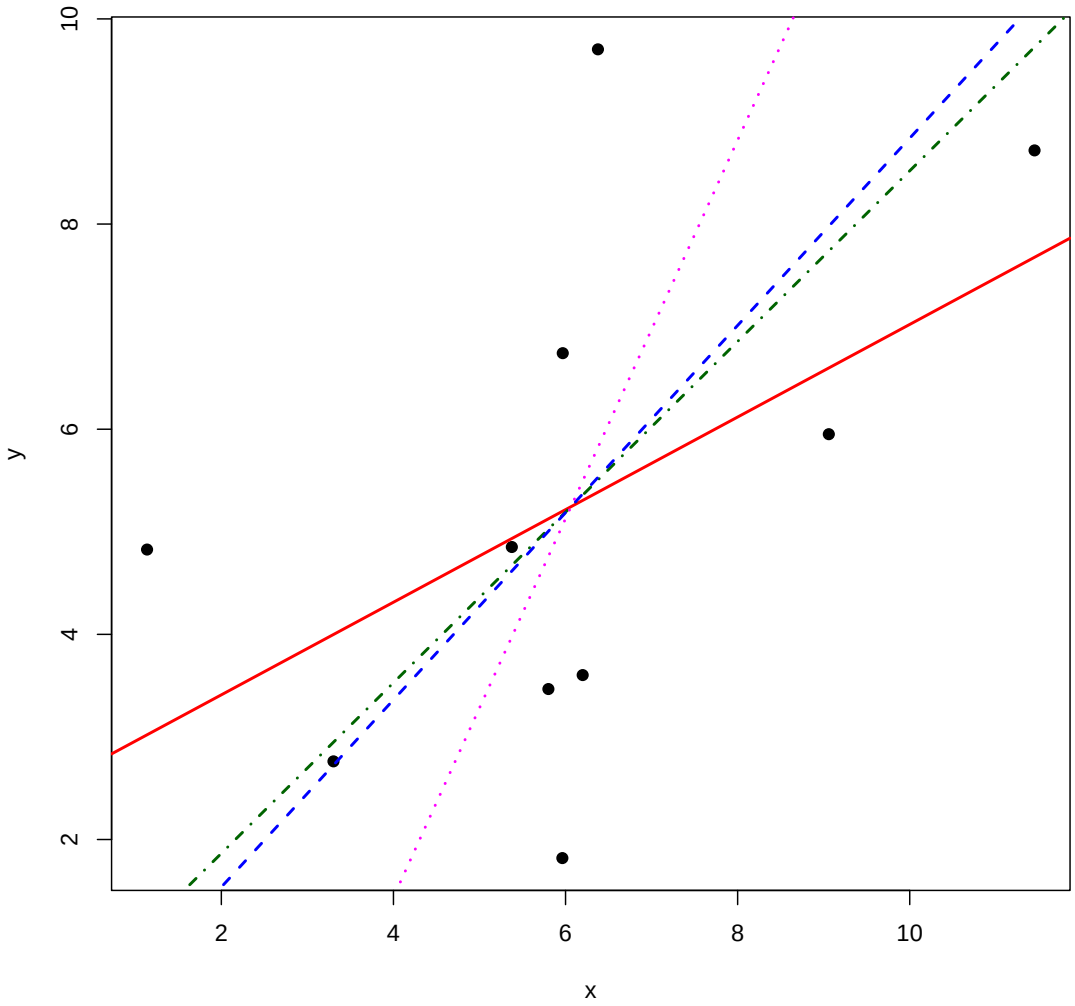
```
## No permutation test will be performed
```

```
##
## Model II regression
##
## Call: lmodel2(formula = y ~ x, range.y = "relative", range.x =
## "relative")
##
## n = 10   r = 0.49461   r-square = 0.24464
## Parametric P-values:   2-tailed = 0.14614    1-tailed = 0.073069
## Angle between the two OLS regression lines = 37.249 degrees
##
## Regression results
##   Method Intercept   Slope Angle (degrees) P-perm (1-tailed)
## 1    OLS   2.50782 0.45136          24.293                NA
## 2     MA   0.20139 0.83175          39.752                NA
## 3    SMA  -0.28854 0.91256          42.382                NA
## 4    RMA  -0.72097 0.98387          44.534                NA
##
## Confidence intervals
##   Method 2.5%-Intercept 97.5%-Intercept 2.5%-Slope 97.5%-Slope
## 1    OLS        -1.7726          6.7883  -0.195259       1.098
## 2     MA       -58.0294          5.7774  -0.087878      10.436
## 3    SMA        -5.4575          2.3839   0.471804       1.765
## 4    RMA        72.5505          4.8711   0.061598     -11.101
##
## Eigenvalues: 10.785 3.5859
##
## H statistic used for computing C.I. of MA: 0.49606
```

Y una vez obtenidos los modelos vamos a compararlos gráficamente para constatar que, al menos en este ejemplo, los resultado son claramente distintos:

```
plot(x,y, pch=20, cex=1.5)
(MinCuad = lm2XY$regression.results[1, ])

##   Method Intercept   Slope Angle (degrees) P-perm (1-tailed)
## 1    OLS    2.5078 0.45136          24.293                NA

abline(MinCuad$Intercept, MinCuad$Slope, col="red", lwd=2, lty=1)
(MAR = lm2XY$regression.results[2, ])

##   Method Intercept   Slope Angle (degrees) P-perm (1-tailed)
## 2     MA   0.20139 0.83175          39.752                NA

abline(MAR$Intercept, MAR$Slope, col="darkgreen", lwd=2, lty=4)
(SMA = lm2XY$regression.results[3, ])

##   Method Intercept   Slope Angle (degrees) P-perm (1-tailed)
## 3    SMA  -0.28854 0.91256          42.382                NA

abline(SMA$Intercept, SMA$Slope, col="blue", lwd=2, lty=2)
(lmXY = lm(x~y))

##
## Call:
## lm(formula = x ~ y)
##
## Coefficients:
## (Intercept)            y
##       3.221        0.542
```

```r
(blmXY1 = 1/lmXY$coefficients[2])

##     y
## 1.845

(blmXY0 = -lmXY$coefficients[1]/lmXY$coefficients[2])

## (Intercept)
##     -5.9422

abline(blmXY0, blmXY1, col="magenta", lwd=2, lty=3)
```



Desde luego, en ejemplos en los que la correlación es muy alta, la diferencia entre los disitntos modelos no es, ni mucho menos, tan acusada.

4. Inferencia en la regresión, usando R.

Opcional: esta sección puede omitirse en una primera lectura.

4.1. Simulando un modelo de regresión lineal simple en R

Recordemos la Ecuación 10.20(pág. 384) del libro, que define el modelo de regresión lineal simple:

$$y = \beta_0 + \beta_1 \cdot x + \epsilon$$

siendo $\epsilon \sim N(0, \sigma)$. Es fácil hacer una simulación de un modelo como este en R. Para obtener los puntos ruidosos del Ejemplo 10.3.3 del libro basta con ejecutar este código, que explicaremos a continuación:

```
rm(list=ls())
set.seed(2013)
n = 10
(x = sort(signif(runif(n, min=0, max=1 ), digits=2) )  )

##  [1] 0.25 0.46 0.73 0.76 0.78 0.80 0.82 0.91 0.93 0.95

beta0 = 1
beta1 = -1/2
(y = beta0 + beta1 * x + rnorm(n,sd=0.01))

##  [1] 0.84892 0.77687 0.63816 0.64020 0.59576 0.60125 0.57557 0.53891
##  [9] 0.52490 0.53377
```

Como ya sabemos, las dos primeras líneas sirven, respectivamente, para hacer limpieza de las variables de R, y para hacer que el generador de números pseudoaleatorios de R produzca siempre los mismos valores. En las líneas 3 y 4 hemos usado `runif` para generar 10 valores de la variable x, usando la distribución uniforme en el intervalo $(0, 1)$. La parte más importante del código está en las líneas 5 a 7, en las que hemos fijado los valores de $beta_0$, β_1 y hemos construido una muestra del modelo de regresión normal simple, por el procedimiento de sumar la parte que corresponde a la recta a una componente aleatoria normal. En el libro los valores de y aparecen redondeados.

Esquemáticamente, para ver la correspondencia entre la teoría y el código en R:

$$\underbrace{y}_{\texttt{y}} = \underbrace{\beta_0 + \beta_1 \cdot x}_{\texttt{beta0 + beta1 * x}} + \underbrace{\epsilon}_{\texttt{rnorm(n,sd=0.01)}} .$$

Es **muy importante** entender que lo que obtenemos, en el vector `yRuido` de R, es una muestra del modelo teórico. Así que, aunque la recta teórica (poblacional) es

$$y = \beta_0 + \beta_1 \cdot x,$$

y en este ejemplo

$$\beta_0 = 1, \qquad \beta_1 = \frac{-1}{2},$$

cuando calculemos la recta de regresión a partir de los vectores x e y, obtendremos una recta

$$y = b_0 + b_1 \cdot x,$$

en la que, desde luego,

$$\beta_0 \neq b_0, \qquad \beta_1 \neq b_1.$$

Concretamente, usando lo que hemos aprendido en las secciones previas de este tutorial, esa recta se obtiene con:

```
(lmXY= lm(y~x) )

##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##       0.983       -0.481
```

Como ves, los valores son

$$b_0 \approx 0.983, \qquad b_1 \approx -0.481,$$

que se parecen, a β_0 y β_1, pero ciertamente no coinciden, como queda de manifiesto en la Figura 10.16 (pág. 375) del libro.

4.2. Intervalos de confianza para los coeficientes de la recta de regresión

Vamos a ver otro ejemplo de una situación en la que la función `lm` nos hace la vida mucho más fácil. En la Sección 10.4.1, y concretamente en la Ecuación 10.23 (pág. 387) hemos visto como construir un intervalo de confianza para la pendiente β_1 de la recta poblacional. Si quisiéramos obtener además un intervalo para β_0, deberíamos usar el estadístico de la Ecuación 10.27 (pág. 389) del libro, y deducir a partir de este estadístico el intervalo. Hacer esos cálculos, aplicando esas fórmulas, y con ayuda de la función `qt`, no es demasiado complicado. Como muestra de lo que decimos, los valores del Ejemplo 10.4.1 (pág. 387) se pueden obtener así, después de aplicar el fichero `Tut10-RectaRegresion.R` a los datos de este Ejemplo:

```
(talfamedios = qt(0.975, df=n - 2))

## [1] 2.306

ECM = EC / (n-1)
(intConfBeta1 = b1 + c(-1,1) * talfamedios * sqrt(ECM /((n-2) * var(x))))

## [1] -0.52872 -0.43292
```

Es muy fácil hacer esto, después de calcular los coeficientes de la recta y el error cuadrático medio. Pero es que hay una manera mucho más sencilla, en una sola instrucción y usando las funciones `lm` y `confint` (de *confidence interval*). Sería así:

```
confint(lmXY, level=0.95)

##                2.5 %   97.5 %
## (Intercept)  0.94596  1.01956
## x           -0.52872 -0.43292
```

Como ves, el resultado son los dos intervalos de confianza, en la primera línea (identificado por `intercept`) el de la ordenada en el origen β_0 , y en la segunda el de la pendiente (identificado por `x`).

4.3. Verificando las condiciones del modelo

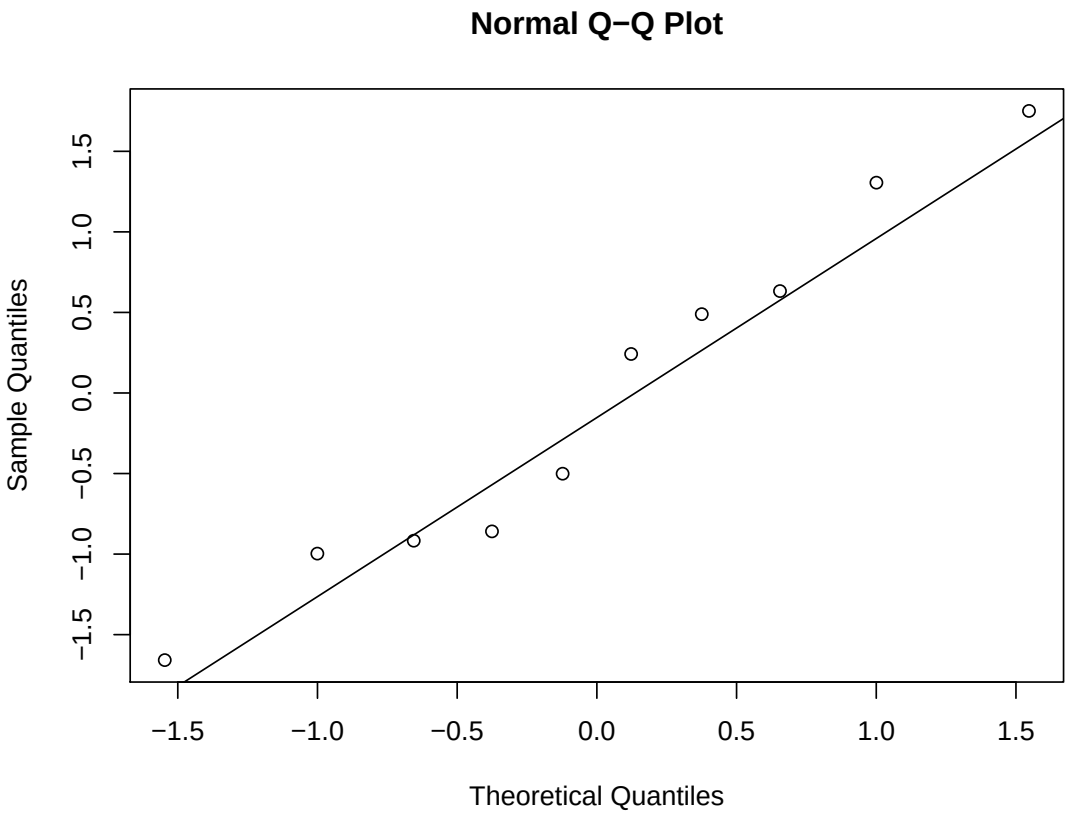
En la Sección 10.4.2 (pág. 389) del libro hemos visto que el análisis de los residuos era la clave para verificar que se cumplen las condiciones necesarias para que la inferencia basada en la regresión sea válida. Vamos a ver, en esta sección, como usar R para ese análisis de los residuos.

Una de las primeras cosas que hemos mencionado es que en el análisis lo habitual es trabajar con los residuos estudentizados. No queremos entrar en una discusión técnica de las razones que hay detrás de esto, ni de cómo se definen esos residuos estudentizados. Lo que sí queremos hacer es mostrar al lector lo fácil que es obtener estos residuos con R (debes haber instalado la librería `MASS`, que ya hemos usado en algún otro tutorial):

```
library(MASS)
(residuosSt = stdres(lmXY))

##        1        2        3        4        5        6        7        8
## -1.65793  1.30566  0.48954  1.75098 -0.91678  0.24219 -0.99682 -0.50095
##        9       10
## -0.85894  0.63269
```

Y una vez hecho esto, basta con usar las funciones `hist` y `boxplot` para obtener las partes (a) y (b) de la Figura 10.19 (pág. 391) del libro. En la parte (c) de esa figura hemos incluido un gráfico de tipo *qq-plot*, que se obtiene en R haciendo, simplemente

```
qqnorm(residuosSt)
qqline(residuosSt)
```



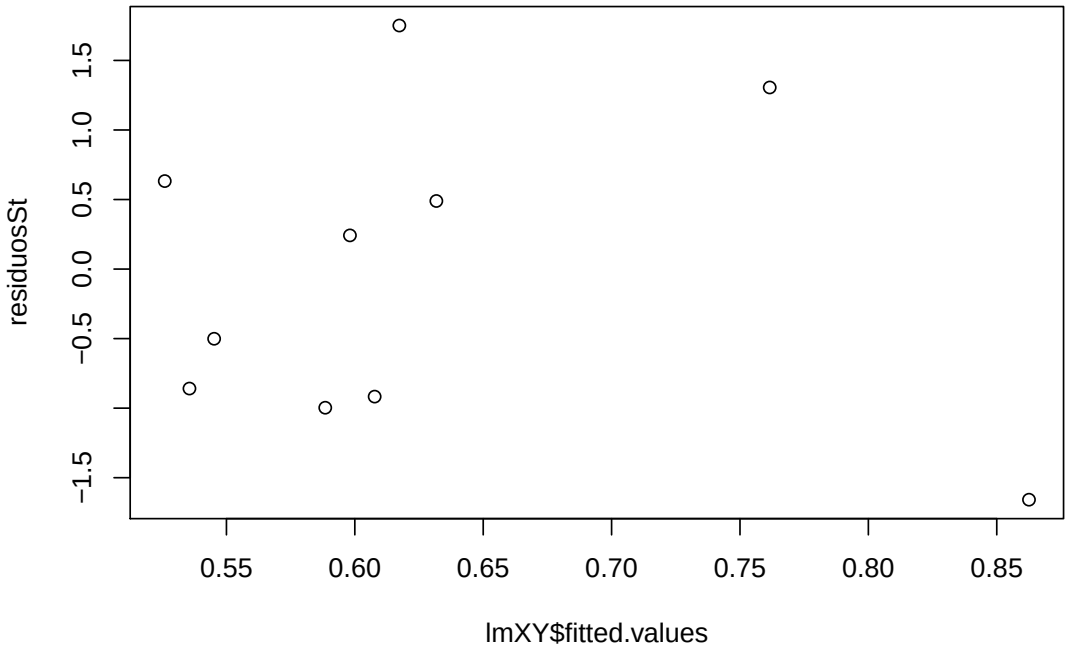
La primera instrucción (con `qqnorm`) dibuja los puntos del *qq-plot*, mientras que la segunda (con `qqline`) añade la recta para que sea más fácil comprobar si el gráfico se ajusta a lo que esperamos en caso de una distribución normal de los residuos. En esta, y en otras figuras que se incluyen en el curso, hemos añadido "decoración" (colores, etiquetas, etc.) al gráfico, usando instrucciones como las que hemos visto en la Sección 2.5 de este tutorial.

Ya hemos dicho, en esa sección, que aunque estos análisis gráficos son muy útiles, a veces es conveniente complementarlos con un contraste de normalidad más formal. Existe una librería en R, llamada `gvlma` (de *Global Validation of Linear Models Assumptions*) que, actuando sobre un modelo producido con `lm`, realiza una serie de contrastes sobre las hipótesis del modelo, y nos resume en la respuesta si esas condiciones se verifican. Recuerda que, para usarla, es necesario primero instalar esa librería. Una vez hecho eso, los resultados se obtienen así:

```
library(gvlma)

## Error in library(gvlma): there is no package called 'gvlma'

gvlma(lmXY)

## Error in eval(expr, envir, enclos): no se pudo encontrar la funci'on "gvlma"
```

Fíjate, en particular, en que también se ha comprobado la hipótesis de homocedasticidad (homogeneidad de las varianzas). Para comprobar esta condición, pero usando métodos gráficos, se acude a menudo a representar los residuos estudentizados frente a los valores que predice el modelo. Ese gráfico se obtiene simplemente haciendo:

```
plot(lmXY$fitted.values, residuosSt)
```



y luego le podemos añadir toda la decoración necesaria, claro. Hay que tener en cuenta, en cualquier caso, que el análisis de una muestra tan pequeña siempre resulta difícil.

Siguiendo con el tema de los diagnósticos gráficos del modelo de regresión, en su momento dijimos que la función `plot`, aplicada directamente a un modelo lineal como el objeto `lmXY` no produciría como resultado el diagrama de dispersión de ese modelo, como tal vez esperaríamos. El resultado de `plot(lmXY)` es, no un gráfico, sino la serie de cuatro gráficos que (para el Ejemplo 10.3.3 del libro que estamos usando) se muestra en la Figura 6 (pág. 29), y que tienen como objeto ayudar al diagnóstico de las condiciones de aplicabilidad del modelo de regresión lineal. En RStudio, al ejecutar `plot(lmXY)`, en la consola de comandos aparece este mensaje:

```
Hit <Return> to see next plot:
```

y debemos situarnos en la consola y pulsar la tecla `Entrar` (o `Return`, o ↵) para avanzar por estos gráficos.

El primero de esos gráficos es precisamente una versión más elaborada del diagrama *residuos vs. predichos* que acabamos de aprender a construir. El siguiente es el *qq-plot* de los residuos estudentizados que también hemos comentado. El tercero es otra versión de *residuos vs. predichos*, que no vamos a discutir, y el cuarto tiene que ver con el contenido del próximo apartado. Así que volveremos allí sobre este gráfico.

4.4. Residuos atípicos y puntos influyentes

Vamos a empezar por el último de la serie de gráficos que se obtienen con `plot(lmXY)` (ver Figura 6, pág. 29). En este gráfico aparece representado el valor de la denominada distancia de Cook para cada uno de los puntos de la muestra. La medida de Cook es una forma habitual de medir la *influencia* de un punto sobre el modelo de regresión, en el sentido que hemos discutido en la Sección 10.4.3 del libro (página 392). No vamos a entrar en los detalles de cómo se define y calcula esa distancia, y nos limitaremos a decir que se suele utilizar, como referencia, el criterio de que un punto con un valor de la distancia de Cook mayor que uno es un *punto influyente*. En nuestro ejemplo, la figura muestra que uno de los puntos de la muestra es influyente. Además, para ayudarnos a localizarlo, R lo ha rotulado con la posición que ocupa dentro de la muestra, que es 1, la primera posición. El punto es $(0.25, 0.84892)$, que es el punto situado más a la izquierda en la Figura 10.16 del libro (pág. 375). ¿Puedes explicar, usando esa figura, la razón de que este punto sea influyente?

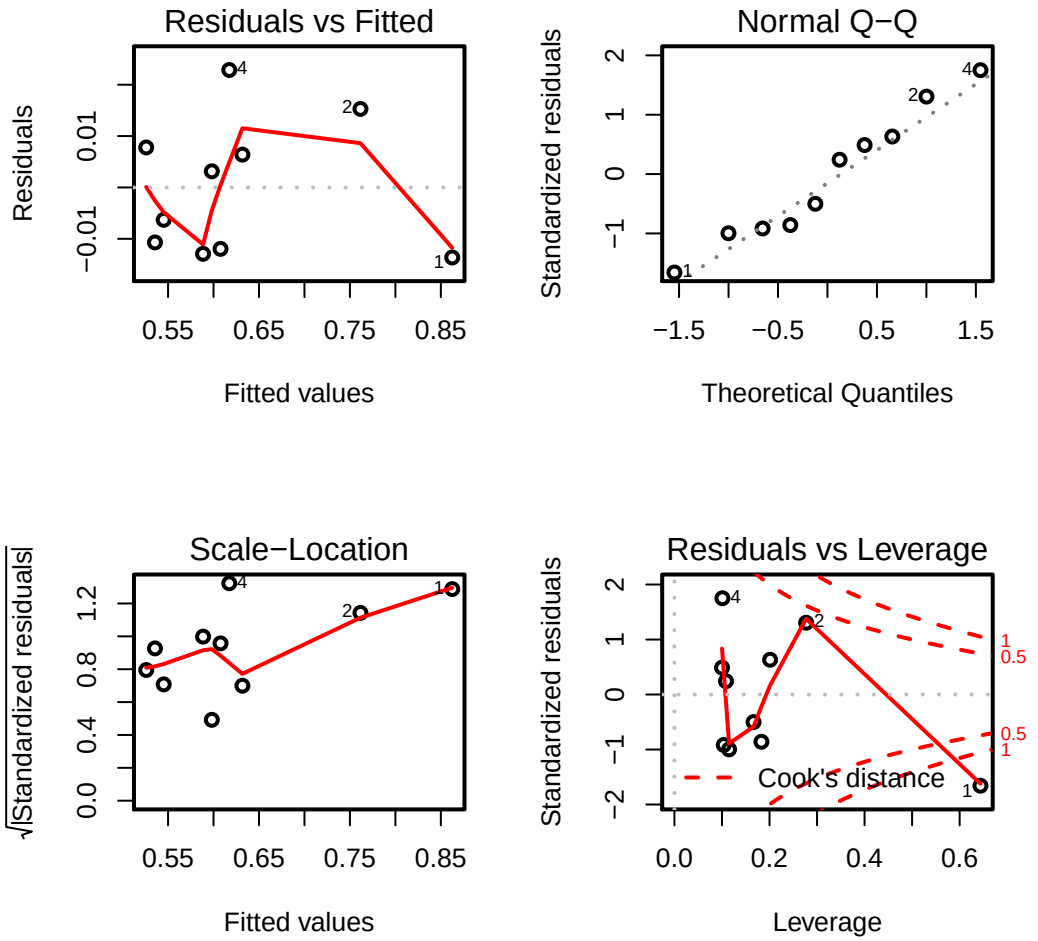
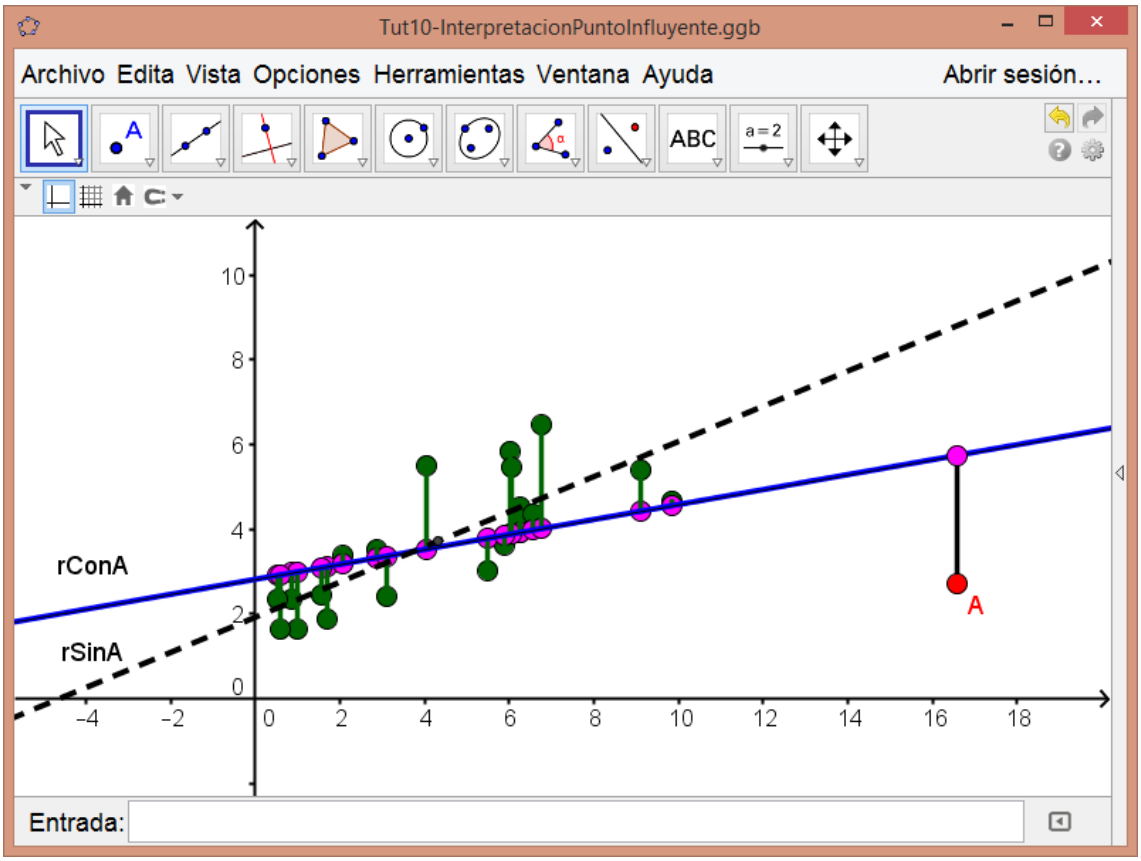


Figura 6: Serie de gráficos que se obtiene al usar `plot(lmXY)` en R.

En cualquier caso, para ayudarte a visualizar en general lo que significa la *influencia* hemos preparado un fichero GeoGebra,

Tut10-InterpretacionPuntoInfluyente.ggb

con el que puedes explorar las ideas que ilustra la Figura c (pág. 395. Cuando abras ese fichero verás una imagen similar a esta:



Como ves, se trata de un diagrama de dispersión con una nube de puntos (de color verde) y un punto adicional, el punto A (de color rojo), que es el punto que podemos mover para experimentar con la idea de punto influyente. Además, se muestran las rectas de regresión correspondientes a la nube de puntos sin A y también con A. La idea de influencia tiene que ver, precisamente, con la diferencia entre esas dos rectas. Para entenderlo lo mejor es que muevas el punto A con el ratón, para ver como cambia la recta que lo tiene en cuenta. Puedes usar los distintos apartados de la Figura 395 del libro como guía para cubrir todas las posibilidades que pueden darse.

En R disponemos de la función `cooks.distance` para obtener esa distancia de Cook. Aplicada al Ejemplo 10.3.3 de los puntos *ruidosos* del libro produce estos valores:

```
cooks.distance(lmXY)
```

```
##          1          2          3          4          5          6          7
## 2.4844717 0.3266243 0.0133412 0.1722289 0.0486853 0.0035679 0.0645098
##          8          9         10
## 0.0250655 0.0826111 0.0504299
```

que confirman lo que vemos en la última gráfica de las cuatro que componen la Figura 6: el primer punto de la muestra es el único punto influyente de este ejemplo.

Fíjate en que, en este ejemplo, aunque ese punto es influyente su residuo no es atípico (puedes usar de nuevo la Figura 10.16 del libro como ayuda para ver esto). De hecho, para comprobar si existe algún residuo atípico podemos utilizar la función `outlierTest` de la librería `car` de R. Esta función realiza un tipo de contraste de hipótesis (basado en la t de Student) para analizar la posible existencia de residuos atípicos. El resultado en este ejemplo es:

```
library(car)
```

```
## Warning: package 'car' was built under R version 3.3.2
```

```
##
## Attaching package: 'car'
```

```
## The following objects are masked from 'package:BSDA':
```

```
##
##    Vocab, Wool
## The following object is masked from 'package:gtools':
##
##    logit

outlierTest(lmXY)
```

En este caso, la frase `No Studentized residuals with Bonferonni p < 0.05` nos informa de que no existen residuos atípicos. Si hubiera residuos atípicos, la función nos los devolvería como resultado. En este caso, al no existir residuos atípicos la función se limita a indicar cuál es el punto cuyo residuo tiene el mayor valor (absoluto), sin llegar a ser atípico.

La matriz H

Volvamos a la medida de la influencia. En la Sección 10.4.3 del libro (pág. 392) hemos visto que la matriz H, la llamada *matriz sombrero*, permite medir fácilmente la influencia de cada punto de la muestra. En particular, nos interesan los valores de la diagonal de esta matriz, que en R se obtienen muy fácilmente así:

```
hatvalues(lmXY)
```

```
##       1       2       3       4       5       6       7       8       9
## 0.64384 0.27704 0.10018 0.10100 0.10382 0.10846 0.11492 0.16650 0.18297
##      10
## 0.20126
```

Para usar estos valores como medida de la influencia utilizamos el criterio de comparar los valores sombrero (*hatvalues)* con el valor $4/n$:

```
4/n
```

```
## [1] 0.4
```

```
hatvalues(lmXY) > (4 / n)
```

```
##     1     2     3     4     5     6     7     8     9    10
##  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

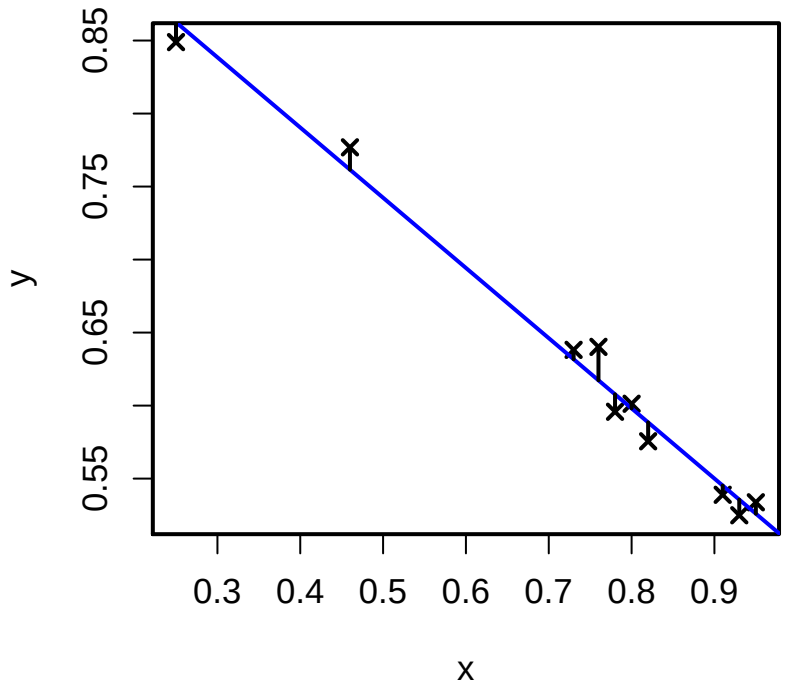
Como puedes ver, el resultado confirma lo que ya habíamos establecido usando la distancia de Cook: el único punto influyente de esta muestra es el primer punto.

4.5. Bandas de confianza y predicción en R.

En la Sección 10.4.4 del libro (pág. 398) hemos visto cómo se pueden definir las bandas de confianza y predicción para un modelo de regresión lineal simple que aparecen en la Figura 10.23 (pág. 401. Vamos a usar R para construir esas bandas, concretamente con la función `predict` que ya hemos encontrado en la página 21 de este tutorial.

Vamos a empezar por fabricar los datos del Ejemplo 10.3.3 del libro (pág. 372) y representarlos en un diagrama de dispersión, junto con la recta de regresión que obtenemos con `lm`. También hemos añadido unos segmentos que ilustran los residuos de cada uno de los puntos de la muestra:

```r
set.seed(2013)
n=10
x = sort(signif(runif(n, min=0, max=1 ), digits=2) )
y = 1 - (x/2) + rnorm(n,sd=0.01)
plot(x, y, pch=4, lwd=2, col="black", cex.lab=1.1, cex.axis=1.1)
box(lwd=2)
lmXY = lm(y ~ x)
abline(lmXY, lwd=2, col="blue")
segments(x,fitted(lmXY), x, y, lwd=2)
```



A continuación vamos a crear una sucesión de puntos que cubran todo el recorrido de valores de x en la muestra. Usaremos un total de 50 puntos, pero puedes usar más para conseguir mayor nivel de detalle.

```r
pred.x = seq(min(x), max(x), length.out=50)
```

Ahora viene el paso clave. Usamos `predict` para crear los intervalos de confianza y predicción para cada uno de los puntos de `pred.x`. La opción `int` de `predict` nos permite seleccionar el tipo de intervalo que se crea en cada caso. Usaremos un nivel de confianza del 95 %. El resultado son dos data.frames de los que mostramos las primeras líneas:

```r
predct.intrvl = predict(lmXY, int="p", newdata=data.frame(x = pred.x), level=0.95)
confd.intrvl = predict(lmXY, int="c", newdata=data.frame(x = pred.x), level=0.95)
head(predct.intrvl)
```

```
##       fit     lwr     upr
## 1 0.86255 0.82183 0.90328
## 2 0.85568 0.81535 0.89602
## 3 0.84882 0.80886 0.88877
## 4 0.84195 0.80237 0.88153
## 5 0.83508 0.79586 0.87429
## 6 0.82821 0.78935 0.86707
```

```
head(confd.intrvl)

##       fit     lwr     upr
## 1 0.86255 0.83707 0.88804
## 2 0.85568 0.83082 0.88054
## 3 0.84882 0.82458 0.87305
## 4 0.84195 0.81833 0.86556
## 5 0.83508 0.81208 0.85807
## 6 0.82821 0.80583 0.85059
```

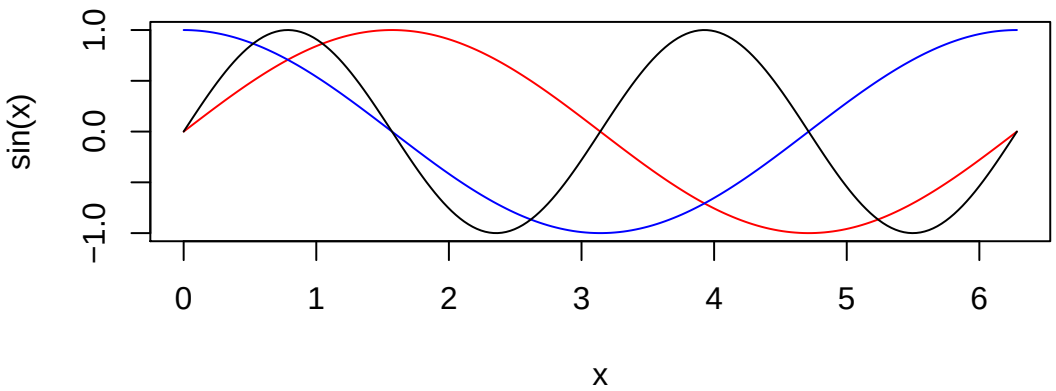
Como ves, en ambos casos se obtiene un `data.frame` con tres columnas que contienen, respectivamente el centro y los extremos inferior y superior del intervalo correspondiente, ya sea de predicción o de confianza.

El trabajo duro ya está hecho, ahora sólo hay que representar gráficamente esas bandas de predicción y confianza. Para eso lo más cómodo es usar la función `matlines` que nos va a permitir dibujar en un mismo gráfico simultáneamente los valores de varias columnas de una matriz. Para que lo entiendas en un ejemplo sencillo, vamos a usar `matlines` para dibujar en un mismo gráfico las curvas $y = \text{sen}(x)$, $y = \cos(x)$ e $y = \text{sen}(3x)$. Lo haremos creando primero una sucesión de 200 puntos x distribuidos uniformemente de 0 a 2π:

```
x = seq(0, 2 * pi, length.out=200)
```

Y ahora hacemos el gráfico con `matlines`. Primero dibujamos una de las curvas con `plot` y la opción `type="n"`. Este comando crea una ventana gráfica de las dimensiones adecuadas, pero no dibuja nada en ella (por el `tyype="n"`), en la que luego `matlines` se encargará de añadir las gráficas. Hemos elegido además los colores y tipo de trazo de las curvas:

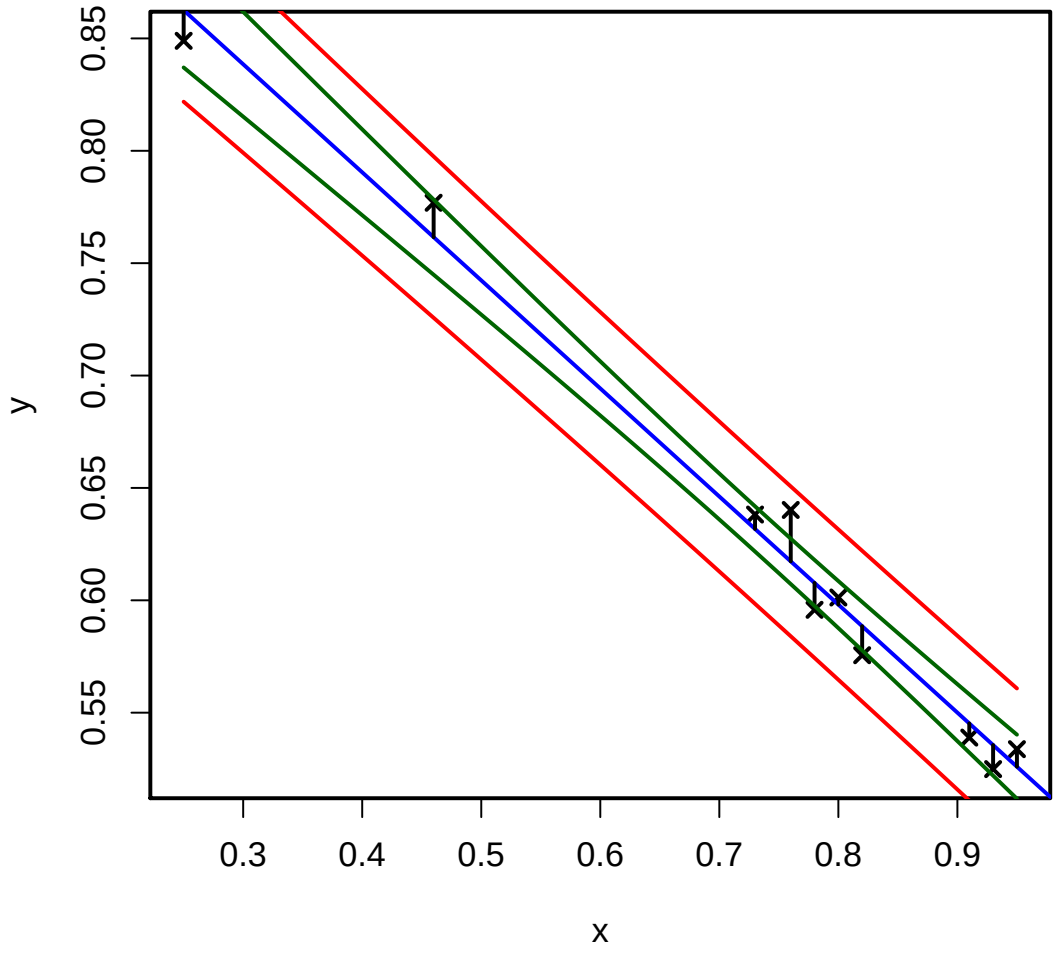
```
plot(x, sin(x), type="n")
matlines(x, cbind(sin(x), cos(x), sin(2*x), lwd=3),
    col=c("red", "blue", "black"), lty=1)
```



Volviendo al dibujo de las bandas de confianza y predicción, el límite superior de la banda de predicción es una curva que recorre los extremos superiores de los intervalos de predicción para cada punto del recorrido de la variable x en los puntos de la muestra. Y análogamente para el resto de curvas que definen las bandas de predicción y confianza. Podemos, por tanto, usar `matlines` para añadir esas bandas al gráfico de dispersión anterior, usando las columnas 2 y 3 de los `data.frames` que hemos construido para almacenar esas bandas. La única diferencia con el gráfico original está en las dos últimas instrucciones:

```
set.seed(2013)
n=10
```

```
x = sort(signif(runif(n, min=0, max=1 ), digits=2) )
y = 1 - (x/2) + rnorm(n,sd=0.01)
plot(x, y, pch=4, lwd=2, col="black", cex.lab=1.1, cex.axis=1.1)
box(lwd=2)
lmXY = lm(y ~ x)
abline(lmXY, lwd=2, col="blue")
segments(x,fitted(lmXY), x, y, lwd=2)
matlines(pred.x, predct.intrvl[ ,2:3], lty=c(1, 1, 1), col="red", lwd=2)
matlines(pred.x, confd.intrvl[ ,2:3], lty=c(1, 1, 1), col="darkgreen", lwd=2)
```



A diferencia de lo que hemos hecho en el libro, aquí hemos usado curvas de trazo continuo para el límite de las gráficas. Puedes cambiar los valores de `lty` para experimentar otras posibilidades.

4.6. El cuarteto de Anscombe en R.

En R, como parte de la instalación básica, disponemos de un `dataframe` llamado `anscombe`, con variables `x1, x2, x3, x4, y1, y2, y3, y4`, y que contiene los valores de estos ejemplos. Para verlos basta con que hagas:

```
anscombe
```

```
##    x1 x2 x3 x4    y1   y2    y3    y4
## 1  10 10 10  8  8.04 9.14  7.46  6.58
```

```
## 2   8  8  8  8  6.95 8.14  6.77  5.76
## 3  13 13 13  8  7.58 8.74 12.74  7.71
## 4   9  9  9  8  8.81 8.77  7.11  8.84
## 5  11 11 11  8  8.33 9.26  7.81  8.47
## 6  14 14 14  8  9.96 8.10  8.84  7.04
## 7   6  6  6  8  7.24 6.13  6.08  5.25
## 8   4  4  4 19  4.26 3.10  5.39 12.50
## 9  12 12 12  8 10.84 9.13  8.15  5.56
## 10  7  7  7  8  4.82 7.26  6.42  7.91
## 11  5  5  5  8  5.68 4.74  5.73  6.89
```

Vamos a comprobar lo que hemos dicho sobre los valores comunes a las cuatro muestras. Usaremos `apply` para calcular la media y cuasivarianza por columnas del `data.frame`:

```
apply(anscombe, 2, mean)
```

```
##     x1     x2     x3     x4     y1     y2     y3     y4
## 9.0000 9.0000 9.0000 9.0000 7.5009 7.5009 7.5000 7.5009
```

```
apply(anscombe, 2, var)
```

```
##      x1      x2      x3      x4      y1      y2      y3      y4
## 11.0000 11.0000 11.0000 11.0000  4.1273  4.1276  4.1226  4.1232
```

Para calcular la covarianza de cada una de las muestras hacemos un pequeño truco con `sapply` y una función anónima. Es un poco más avanzado que otras cosas que hemos hecho con R, pero si no lo entiendes siempre puedes hacer las cuentas más a mano:

```
sapply(1:4, function(x){cov(anscombe[ ,x], anscombe[ , x + 4])})
```

```
## [1] 5.501 5.500 5.497 5.499
```

Y para el coeficiente de correlación un truco análogo:

```
sapply(1:4, function(x){cor(anscombe[ ,x], anscombe[ , x + 4])})
```

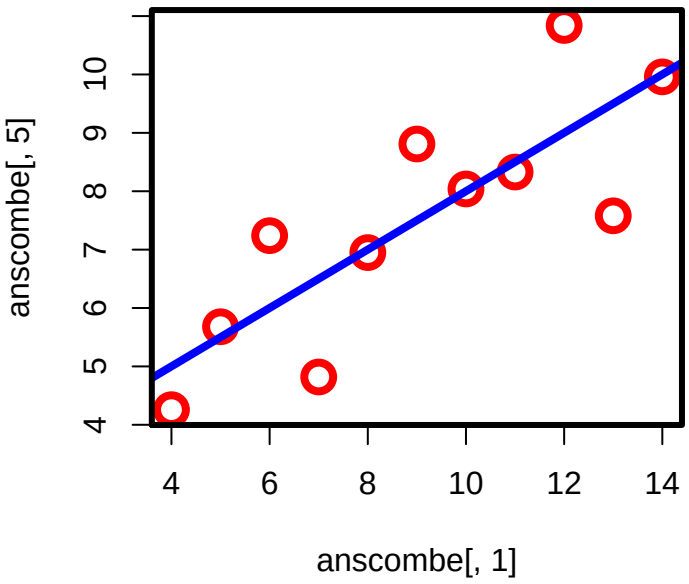
```
## [1] 0.81642 0.81624 0.81629 0.81652
```

Para dibujar de forma sencilla uno de estos ejemplos puedes hacer:

```
(lmA1=lm(anscombe[, 5] ~ anscombe[ , 1]))
```

```
##
## Call:
## lm(formula = anscombe[, 5] ~ anscombe[, 1])
##
## Coefficients:
##   (Intercept)  anscombe[, 1]
##           3.0             0.5
```

```
plot(anscombe[ , 1], anscombe[ , 5], col="red", lwd=4, cex=2)
abline(lmA1, col="blue", lwd=4)
box(lwd=3,bty="o")
```



Si lo que queremos es un diagnóstico sencillo de este modelo de regresión podemos hacer:

```
studres(lmA1)

##         1         2         3         4         5         6         7
##  0.031345 -0.040845 -2.081099  1.126800 -0.139801 -0.038196  1.116959
##         8         9        10        11
## -0.704581  1.838330 -1.568460  0.156809

outlierTest(lmA1)

##
## No Studentized residuals with Bonferonni p < 0.05
## Largest |rstudent|:
##   rstudent unadjusted p-value Bonferonni p
## 3  -2.0811           0.070994      0.78093

hatvalues(lmA1)

##        1        2        3        4        5        6        7        8
## 0.100000 0.100000 0.236364 0.090909 0.127273 0.318182 0.172727 0.318182
##        9       10       11
## 0.172727 0.127273 0.236364

hatvalues(lmA1) > 2 * mean(hatvalues(lmA1))

##     1     2     3     4     5     6     7     8     9    10    11
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

cooks.distance(lmA1)

##           1           2           3           4           5           6
## 0.000061398 0.000104247 0.489209276 0.061636999 0.001599342 0.000382900
##           7           8           9          10          11
## 0.126756485 0.122699896 0.279029593 0.154341222 0.004268011
```

```
gvlma(lmA1)

## Error in eval(expr, envir, enclos): no se pudo encontrar la funci'on "gvlma"
```

Te proponemos unos ejercicios adicionales para completar el estudio del *cuarteto de Anscombe*:

Ejercicio 8.

1. *Interpreta los resultados anteriores en términos de la validez del modelo, presencia de puntos influyentes, etc.*
2. *Usa la función* `plot` *para hacer un diagnóstico gráfico de este modelo de regresión.*
3. *Haz un diagnóstico de los modelos de regresión lineal simples correspondientes a los otros tres integrantes del* cuarteto.

□

La advertencia fundamental que hay que extraer de estos cuatro ejemplos es que el coeficiente de correlación r, no puede servir por si mismo como indicador de la calidad de un modelo de regresión lineal. Pero, abundando en esa dirección, también hay que observar que ningún análisis de regresión puede considerarse completo si no se incluye la exploración de los datos (y lo mismo sirve para cualquier análisis estadístico). La exploración gráfica, pero también un análisis minucioso de las condiciones del modelo, son herramientas imprescindibles, sin las cuales corremos el riesgo de que nuestras conclusiones carezcan de fundamento.

5. Modelos de regresión, más allá de las rectas.

Opcional: esta sección puede omitirse en una primera lectura.

La Sección 10.5 del libro (pág. 404) comienza con el Ejemplo 10.5.1, en el que mediante un cambio de variable reducimos la construcción de un modelo de la forma

$$y = a_0 \cdot x^{a_1}$$

a la de un modelo

$$\tilde{y} = b_0 + b_1 \cdot \tilde{x}.$$

donde

$$\begin{cases} \tilde{x} = \ln x, \\ \tilde{y} = \ln y, \\ b_0 = \ln a_0, \\ b_1 = a_1, \end{cases}$$

Los puntos de ese ejemplo se han construido en R con este código

```
set.seed(2014)
n=20
x=sort(signif(runif(n,min=10,max=25),digits=3))
y=round(3e-4 * x^(26/7) * exp(rnorm(n, 0, 0.1)),digits=1)
```

¿Puedes ver por qué ese código produce una colección de puntos adecuada para el modelo que nos planteamos en este ejemplo? Fíjate en particular en el término de error `exp(rnorm(n, 0, 0.1)` que nos garantiza que una vez aplicado el cambio de variable podremos aplicar con éxito un modelo lineal simple.

Ejercicio 9.

1. *Usa R para construir los puntos* $(\tilde{x}, \tilde{y})$ *de este ejemplo. Recuerda que el logaritmo neperiano se obtiene en R con* `log`.
2. *Construye (usando una plantilla o la función* `lm`*) el modelo lineal para esa muestra de puntos* $(\tilde{x}, \tilde{y})$*, y comprueba los resultados del Ejemplo 10.5.1 del libro.*

3. *Adicionalmente, puedes tratar de reproducir la Figura 10.26 del libro (pág.* **??***) usando la función* `curve` *para dibujar la función exponencial.*

□

Regresión polinómica en R.

Vamos a aprovechar el segundo miembro del cuarteto de Anscombe para introducirnos en el terreno de la regresión polinómica con R. Ese ejemplo en particular (que aparece en la parte superior derecha de la Figura 10.25 del libro, pág. 403) requiere el uso de una parábola en lugar de una recta. La ecuación de esa parábola será de la forma:

$$y = b_0 + b_1 \cdot x + b_2 \cdot x^2$$

para ciertos coeficientes b_0, b_1, b_2.

¿Cómo podemos usar R para averiguar cuál es la mejor parábola posible? Es decir, para localizar los valores adecuados de b_0, b_1, b_2. Pues usando, de nuevo, la función `lm`, pero con una sintaxis ligeramente más complicada. Para ilustrarlo, empezamos por crear dos vectores con las coordenadas x e y de este ejemplo:

```
 (x = anscombe$x2)
```

```
##  [1] 10  8 13  9 11 14  6  4 12  7  5
```

```
 (y = anscombe$y2)
```

```
##  [1] 9.14 8.14 8.74 8.77 9.26 8.10 6.13 3.10 9.13 7.26 4.74
```

Y ahora, para crear ese modelo polinómico de grado dos basta con usar este código:

```
(lmPolXY=lm(y ~ I(x) + I(x^2)))
```

```
##
## Call:
## lm(formula = y ~ I(x) + I(x^2))
##
## Coefficients:
## (Intercept)         I(x)       I(x^2)
##      -5.996        2.781       -0.127
```

La peculiar notación `I(x) + I(x^2)` se debe a que, por defecto, si escribiéramos simplemente:

```
lm(y ~ x + x^2)
```

```
##
## Call:
## lm(formula = y ~ x + x^2)
##
## Coefficients:
## (Intercept)            x
##         3.0          0.5
```

R usaría los denominados polinomios ortogonales, un tipo especial de polinomios que tienen propiedades que los hacen muy útiles en este contexto. Como de costumbre, R tiende a proporcionar la mejor solución, pero esa, a menudo, no es la que queremos ver cuando estamos aprendiendo a hacer las cosas por primera vez. El precio que pagaremos, en este caso, es esa notación un poco más complicada con la `I`.

Usando esa notación, en cualquier caso, obtenemos el modelo parabólico que andábamos buscando. En particular, los coeficientes b_0, b_1 y b_2 quedan almacenados en un vector, al que podemos acceder

usando la componente `coefficients` del resultado de la función `lm`. Para este ejemplo, y usando `b` como nombre del vector para mantener la notación que nos es familiar, hacemos:

```
(b = lmPolXY$coefficients)

## (Intercept)        I(x)      I(x^2)
##    -5.99573     2.78084    -0.12671
```

Pero cuidado al usar este vector, porque hay un desplazamiento en los índices: `b[1]`, `b[2]` y `b[3]` corresponden a b_0, b_1 y b_2 respectivamente.

En la Figura 7 (pág. 39) se muestra la parábola que hemos obtenido, que en este caso pasa exactamente por los puntos del segundo ejemplo de Anscombe. Se muestra además la recta de regresión que obtenemos con `lm(y~x)`. Como puedes ver, el modelo que proporciona la recta no es adecuado. El código R que hemos usado para producir esa figura es este:

```
plot(anscombe[ , 2], anscombe[ , 6], col="red", lwd=2, cex=1.1)
curve(b[1] + b[2] * x + b[3] * x^2,
    add = TRUE, from = 3, to = 15, col="blue", lwd=2)
abline(lm(anscombe[ , 6] ~ anscombe[ , 2]), lwd=2, lty=2)
```

Terminamos este apartado proponiendo al lector un ejercicio:

Ejercicio 10. *Vamos a buscar una parábola para los datos del Ejemplo 10.3.1 (pág 368) del libro. Los datos están en el fichero:*

`Cap10-EjemploRectaMalaAproximacion01.csv`

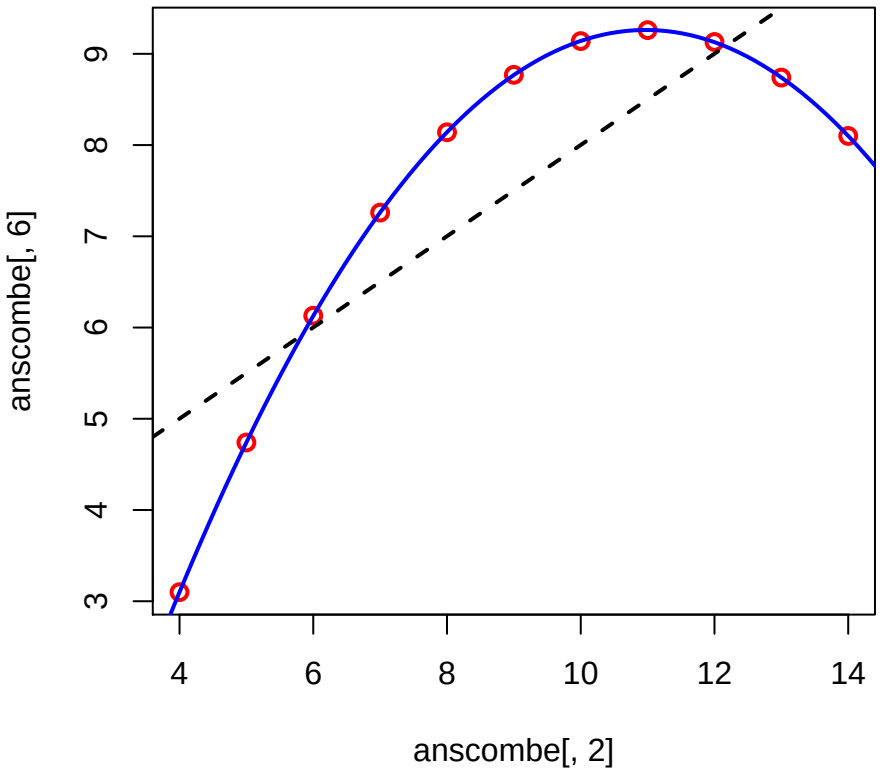


Figura 7: Ejemplo de regresión polinómica con R.

que puedes encontrar en el ejercicio 2(b) de la pág. 14 de este tutorial. Los valores de ese ejemplo se han generado en R con el código:

```
set.seed(2013)
(x = signif(runif(n=30, min=0, max=1 ), digits=3) )
(y = signif(x-x^2+rnorm(30,sd=0.001),digits=2 ) )
```

Eso significa que se basan en un modelo teórico como este:

$$y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \epsilon = x - x^2 + \epsilon$$

En el que hemos tomado

$$\beta_0 = 0, \beta_0 = 1, \beta_2 = -1,$$

y en el que el término de error cumple

$$\epsilon \sim N(30, 0.001).$$

Naturalmente, cuando obtengas b_0, b_1 y b_2, sus valores no coincidirán exactamente con los de $\beta_0 = 0$, $\beta_0 = 1$ y $\beta_2 = -1$, pero deberían ser bastante parecidos. *Para comprobar gráficamente el resultado, dibuja la parábola que obtienes sobre el diagrama de dispersión de los puntos (x, y).* □

□

6. Ejercicios adicionales y soluciones.

Ejercicios adicionales.

Ejercicio 11. *El Ejemplo 10.4.3 del libro (pág. 390) incluye una muestra de puntos para l que la condición de homogeneidad de la varianza claramente no se cumple. Esa muestra de puntos se ha fabricado en R con este código:*

```
set.seed(2013)
n=100
(x = sort(signif(runif(n, min=0, max=1 ), digits=2) )  )
(y = 1 - (x/2) + rnorm(n,sd=0.01*(1+50*x)))
```

1. *Dibuja el diagrama de dispersión de esa muestra de puntos.*
2. *Dibuja también el gráfico de los residuos estudentizados frente a los valores predichos que aparece en la parte (b) de la Figura 10.20 del libro (pág. 392). Para facilitarte el trabajo, los residuos estudentizados se pueden obtener con la función* `studres` *de la librería* `MASS`*, aplicada al modelo lineal que se obtiene con* `lm(y ~ x)`*. Y recuerda que los valores predichos son la componente* `fitted.values` *de la salida de la función* `lm`*.*
3. *Aplica la función* `gvlma` *(de la librería homónima) al modelo lineal para ver los diagnósticos que proporciona.*
4. *Por último, ¿por qué sucede esto? Busca una explicación en la forma en la que se han construido los valores y a partir de los valores x.*

□

Ejercicio 12. *Comprueba los resultados del Ejemplo 10.5.3 del libro (pág. 414).* □

Soluciones de algunos ejercicios.

- **Ejercicio 1, pág. 6**

1. Este es el caso más simple:

```
(datos = read.table("../datos/tut10-ejercicio01-1.csv", sep=",", header=TRUE))
```

```
##     x     y
## 1 0.9  0.63
## 2 1.4  4.38
## 3 1.8  5.86
## 4 1.9  6.43
## 5 2.3  7.81
## 6 2.5 10.05
## 7 2.6  9.02
## 8 2.9 11.23
```

2. Basta con modificar la opción `sep`:

```
(datos = read.table("../datos/tut10-ejercicio01-2.csv", sep="\t", dec=",", header=TRUE))
```

```
##     x     y
## 1 0.9  0.63
## 2 1.4  4.38
## 3 1.8  5.86
## 4 1.9  6.43
## 5 2.3  7.81
## 6 2.5 10.05
## 7 2.6  9.02
## 8 2.9 11.23
```

3. En este caso el fichero de datos está organizado de una manera especialmente molesta, con los datos dispuestos en filas en lugar de columnas, separados por punto y coma, y usando comas para los decimales. Pero con R podemos salir del paso sin demasiadas complicaciones. Los siguientes comandos te permitirán leer los datos y guardarlos en un `data.frame` idéntico al de los apartados anteriores:

```
datos = read.table("../datos/tut10-ejercicio01-3.csv", sep=";", dec=",")
datos = t(as.matrix(datos[ , -1]))
datos = data.frame(datos, row.names = NULL)
colnames(datos) = c("x", "y")
datos
```

```
##     x     y
## 1 0.9  0.63
## 2 1.4  4.38
## 3 1.8  5.86
## 4 1.9  6.43
## 5 2.3  7.81
## 6 2.5 10.05
## 7 2.6  9.02
## 8 2.9 11.23
```

El paso clave es el segundo, en el que leemos la parte numérica de los datos como una matriz usamos `datos[ , -1]` para *excluir* la primera columna), y la trasponemos con la función `t`.

- **Ejercicio 2, pág. 14**

1. La fórmula de la Ecuación 10.16 del libro se obtiene con:

```
with(datosPisa, {
  cov(rpc, pisa)/(sd(rpc) * sd(pisa))
  })
```

2. Ejecutando

```
CoeficienteCorrelación[datosPisa]
```

obtenemos 0.75277, de acuerdo con lo que se obtiene en R.

Fin del Tutorial10. ¡Gracias por la atención!