

Tutorial 10: Regresión

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: 29 de diciembre de 2013. Si este fichero tiene más de un año, puede resultar obsoleto. Busca si existe una versión más reciente.

Índice

1. Regresión lineal con R	1
1.1. Diagramas de dispersión	4
1.2. Calculando y dibujando la recta de regresión	5
1.3. Coeficiente de correlación, Residuos y error cuadrático medio	6
1.4. Un gráfico de regresión más expresivo	8
2. Regresión lineal con Calc, Wiris y Wolfram Alpha	11
3. Introducción a la función <code>lm</code> de R	12
4. Inferencia en la regresión, usando R.	13
4.1. Simulando un modelo de regresión lineal simple en R	14
4.2. Otro vistazo a la descomposición ANOVA	15
4.3. Intervalos de confianza para los coeficientes de la recta de regresión	15
4.4. Verificando las condiciones del modelo	16
4.5. Residuos atípicos y puntos influyentes	17
4.6. El cuarteto de Anscombe	20
5. Regresión polinómica con R	22

1. Regresión lineal con R

Vamos a presentar en esta sección los comandos de R necesarios para el análisis de regresión lineal que se describe en el Capítulo 10 del Curso. 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.

Formato de los datos y diagramas de dispersión

Si tenemos una muestra, formada por n pares de datos,

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n),$$

la forma más conveniente de representar esta situación en R es mediante dos vectores numéricos, a los que en este tutorial vamos a llamar en general \mathbf{x} e \mathbf{y} . De esa forma es fácil calcular las

medidas (longitud, media, cuasivarianza) de cada uno de esos vectores. Además, como veremos, los comandos de R específicos para la regresión se expresan con mucha sencillez en este formato de vectores separados.

Eso nos lleva a la pregunta de cómo introducir los datos en R. Si el número de datos es pequeño, bastará con crear los vectores directamente, como en este ejemplo:

Ejemplo 1.1. 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.

	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.

Para introducir estos datos en R basta con crear dos vectores, como hemos hecho aquí:

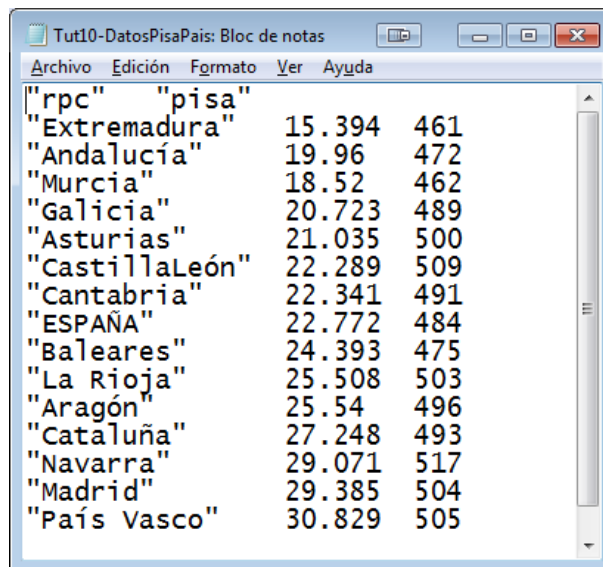
```
rpc=c(15.394,19.960,18.520,20.723,21.035,22.289,22.341,22.772,
      24.393,25.508,25.540,27.248,29.071,29.385,30.829)
pisa=c(461,472,462,489,500,509,491,484,475,503,496,493,517,504,505)
```

Pero, naturalmente, en otros casos, y especialmente si el número n de datos es grande, lo más seguro es que dispongamos de los datos en un fichero. Vamos a suponer que esos datos están almacenados en un fichero `csv`, con cada una de las variables en una columna. Como ya sabemos, los datos pueden estar separados por espacios, comas, tabuladores, etc., y los decimales se pueden indicar por comas o puntos. Además, en ocasiones el fichero incluye una primera línea de cabecera (*header*, en inglés) en la que aparecen los nombres de las variables.

El fichero adjunto [Tut10-DatosPisaPais.csv](#), cuyo contenido se muestra en la Figura 1 contiene los mismos datos de este ejemplo

Este es el típico fichero de datos que, tras leerlo con `read.table`, se almacena en R en un `data.frame`. Pero antes de lanzarnos a hacer eso, fijate en algunas peculiaridades de este fichero:

- La primera columna contiene, entrecomillados, los nombres de las comunidades autónomas. Esos nombres se van a usar en R como nombres de las filas del `data.frame`.
- La primera línea (cabecera) del fichero contiene los nombres de las variables. Son dos variables, aunque en el fichero veas tres columnas, porque la primera columna no se considera aquí una variable, sino un nombre, un mero identificador de fila. La función `read.table` es capaz



"rpc"	"pisa"
Extremadura	15.394 461
Andalucía	19.96 472
Murcia	18.52 462
Galicia	20.723 489
Asturias	21.035 500
CastillaLeón	22.289 509
Cantabria	22.341 491
ESPAÑA	22.772 484
Baleares	24.393 475
La Rioja	25.508 503
Aragón	25.54 496
Cataluña	27.248 493
Navarra	29.071 517
Madrid	29.385 504
País Vasco	30.829 505

Figura 1: Fichero `csv` con los datos del Ejemplo 1.1.

de detectar automáticamente esto: si la línea de cabecera tiene un elemento menos que las demás, `read.table` asume que la primera columna se considera como una columna de nombres (identificadores). Si el fichero no contiene una columna de nombres, R usará números en su lugar.

- Finalmente, el hecho de que los datos aparezcan tan nítidamente alineados en columnas apunta a que se han usado tabuladores para separar las columnas de datos. El tabulador se diseñó precisamente para facilitar visualmente la lectura de los datos. Puedes confirmar esto abriendo el fichero con Calc, como se ilustra en la Figura 2. El problema con la cabecera en Calc se puede solucionar manualmente tras abrir el fichero.

Volviendo a R, para que `read.table` reconozca esa separación mediante tabuladores, debemos incluir la opción `sep="\t"`, ya que `\t` es la forma en la que R se refiere al tabulador.

Con estas observaciones estamos listos para leer el fichero de datos, y almacenarlo en un `data.frame` que vamos a llamar `RpcPisa`. El comando necesario (asegúrate de que el directorio de trabajo es correcto) es:

```
RpcPisa=read.table(file="Tut10-DatosPisaPais.csv",header=TRUE,sep="\t")
```

Con este comando hemos creado un `data.frame`, con dos variables, a las que podemos acceder usando los nombres `RpcPisa$rpc` y `RpcPisa$pisa`. Para no tener que usar estos nombres largos, vamos a ejecutar el comando:

```
attach(RpcPisa)
```

que nos permite, a continuación, referirnos a las variables simplemente como `rpc` y `pisa`.

En la próxima sección volveremos a este ejemplo, y seguiremos el análisis de regresión de esos datos. Pero antes, en los siguientes ejercicios, puedes practicar la lectura de datos a partir de ficheros `csv`.

Ejercicios: 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, conviene echarle un vistazo al fichero de datos, usando un editor de texto (como el *Bloc de Notas* de Windows).

- El primer fichero es muy sencillo, no debe suponer problemas:
[tut10-ejercicio01-1.csv](#)

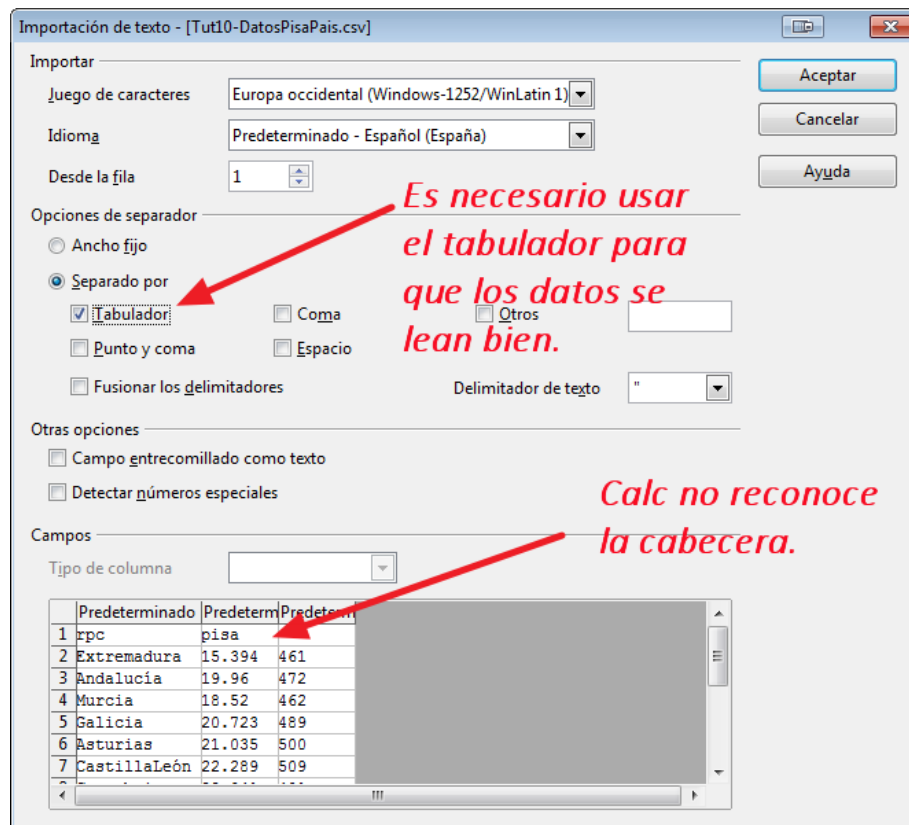


Figura 2: Abriendo el fichero Tut10-DatosPisaPais.csv con Calc.

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](#)

□

1.1. Diagramas de dispersión

Una vez que hemos leído los datos, el siguiente paso razonable es explorarlos, mediante un diagrama de dispersión. Si los datos están en los vectores `x` e `y`, esto se consigue en R con un comando muy sencillo:

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

Por ejemplo, para los datos del informe PISA y la renta per cápita por comunidades autónomas (Ejemplo 1.1), que habíamos leído en dos variables llamadas `rpc` y `pisa`. Así que haríamos:

```
plot(rpc, pisa)
```

Y el resultado es el gráfico, un tanto espartano, que aparece en la Figura 3. De momento, es suficiente para nuestros propósitos, porque parece indicar que, en efecto, hay una cierta correlación entre ambas variables. No nos vamos a entretener mucho en los aspectos técnicos necesarios para mejorar el aspecto de este gráfico. Entre otras cosas, porque las posibilidades son muchas. Pero en la Sección 1.4 daremos algunas indicaciones de por donde se puede empezar esa tarea.

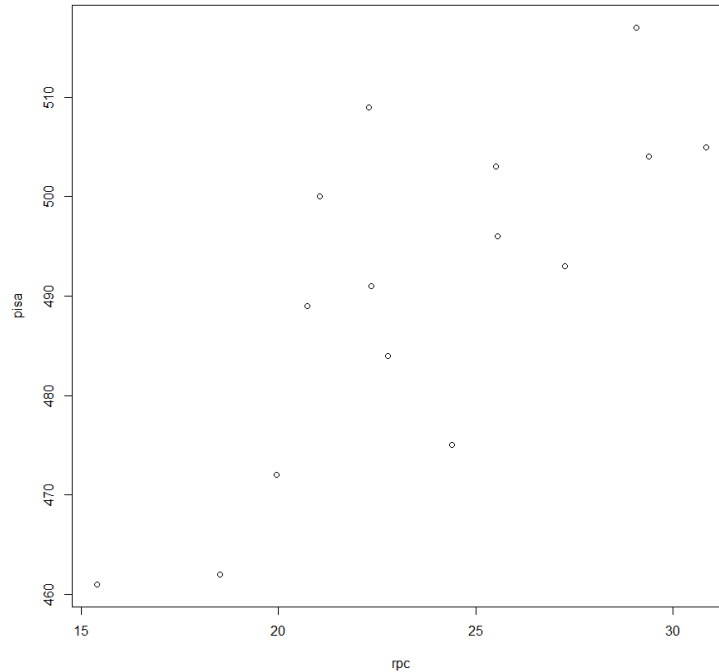


Figura 3: Diagrama de dispersión para el Ejemplo 1.1.

1.2. Calculando y dibujando la recta de regresión

En el Capítulo 10 hemos visto que la covarianza de los dos vectores x e y es un ingrediente esencial para obtener la recta de regresión de y frente a x . Y naturalmente, en R se incluye la función `cov` para calcularla. Por ejemplo, para los datos del Ejemplo 1.1, podemos obtener la covarianza de `rpc` y `pisa` haciendo (se muestra la salida)

```
> cov(pisa, rpc)
[1] 55.2612
```

Recuerda que en R **se trata de la covarianza muestral**, que usa $n - 1$ en el denominador. Si deseas la covarianza que usa n , debes usar el truco habitual, haciendo algo como esto:

```
> (n=length(pisa))
[1] 15
> (n-1)*cov(pisa, rpc)/n
[1] 51.57712
```

Con la función `cov` estamos listos para calcular la pendiente de la recta de regresión, que de acuerdo con la Ecuación 10.6 (pág. 263) del curso es (se muestra la salida):

```
> (b1=cov(pisa, rpc)/var(rpc))
[1] 2.950976
```

Y la ordenada en el origen es, entonces:

```
> (b0=mean(pisa) - b1 * mean(rpc))
[1] 420.892
```

Así que la recta de regresión es, aproximadamente,

$$y = 420.892 + 2.950976 \cdot x$$

Naturalmente, nos gustaría ver esta recta incluida en el diagrama de dispersión. Para ello, disponemos del comando `abline`, que puede dibujar cualquier recta

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

a partir de su pendiente b y ordenada en el origen a (esta notación es la que da origen al nombre `abline`). En nuestro ejemplo, debemos hacer:

```
abline(a = b0, b = b1)
```

y el resultado es la gráfica que se muestra en la Figura 4.

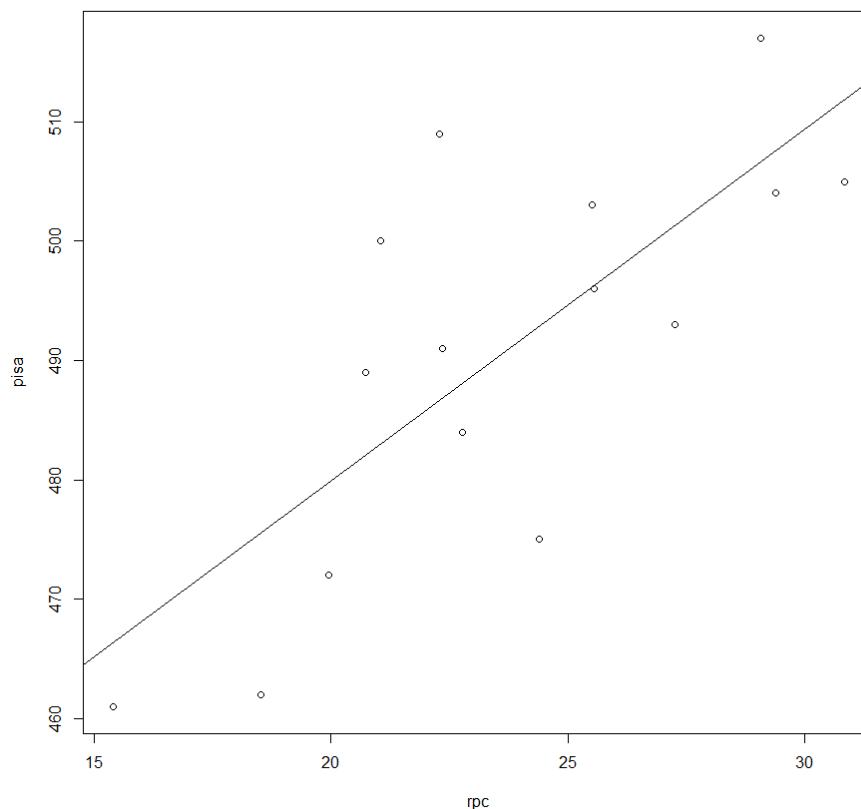


Figura 4: Recta de regresión en el diagrama de dispersión para el Ejemplo 1.1.

1.3. Coeficiente de correlación, Residuos y error cuadrático medio

El coeficiente de correlación r que aparece en la Ecuación 10.12 (pág. 279) del curso, se calcula en R con la función `cor`. Así que, para los datos del Ejemplo 1.1, se obtiene (se muestra la salida):

```
> cor(pisa, rpc)
[1] 0.7527695
```

Ejercicio: Comprueba que el resultado de `cor` es el mismo que cuando se usa `cov` y la fórmula de la Ecuación 10.12 del curso. Recuerda que `sd` calcula las cuasidesviaciones típicas que necesitarás para el denominador. □

Residuos y error cuadrático medio

Naturalmente, a partir de la recta, también podemos 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, en el Ejemplo1.1 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*rpc)
[1] 466.3193 479.7935 475.5441 482.0451 482.9658 486.6663 486.8197 488.0916 492.8752
[10] 496.1655 496.2599 501.3002 506.6798 507.6064 511.8676
```

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=pisa-pisaRecta)
[1] -5.3193148 -7.7934734 -13.5440672 6.9549316 17.0342269 22.3337024 4.1802517
[8] -4.0916192 -17.8751521 6.8345092 -0.2599221 -8.3001899 10.3201800 -3.6064266
[15] -6.8676366
```

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] 1745.89
```

Como hemos dicho, conviene hacer una visita a la Sección 3 de este tutorial, para ver 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](#)

Ejercicios:

1. Usando los datos del fichero `csv` del ejercicio de la página 3 (usa la primera versión, para simplificar), haz las siguientes operaciones:
 - a) Como hemos dicho en aquel ejercicio, asegúrate de que has leído el fichero `csv` y almacenado los datos de cada variable en dos vectores de R, llamados, `x` e `y`
 - b) Dibuja el diagrama de dispersión.
 - c) Calcula la media, las cuasivarianzas muestrales y la covarianza muestral de los vectores `x` e `y`.
 - d) Calcula los coeficientes `b0` y `b1` de la recta de dispersión.
 - e) Añade esa recta al diagrama de dispersión, y asegúrate de que tiene el aspecto que esperabas.
 - f) Calcula el coeficiente de correlación `r` de `x` e `y`.
 - g) Calcula los residuos y el error cuadrático EC de esos vectores, para la recta de regresión que has obtenido antes.
2. Comprueba 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, como el del ejercicio anterior.
 - a) Ejemplo 10.2.1 (pág 263). Datos en el fichero:
[cap10-EjemploRegresion01.csv](#)
 - b) Ejemplo 10.3.1 (pág 269). Datos en el fichero:
[Cap10-EjemploRectaMalaAproximacion01.csv](#).
 - c) Ejemplo 10.3.2 (pág 269). Datos en el fichero:
[Cap10-EjemploRectaMalaAproximacion02.csv](#).

- d) Ejemplo 10.3.3 (pág 273).
Primero los puntos “no ruidosos”. Datos en el fichero:
[Cap10-Ejemplo-ANOVA01.csv](#).
- e) Y del mismo ejemplo 10.3.3, ahora los puntos “ruidosos”. Datos en el fichero:
[Cap10-Ejemplo-ANOVA02.csv](#).

□

1.4. Un gráfico de regresión más expresivo

Atención: esta sección es opcional.

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(rpc, pisa)
```

por una versión más detallada (aquí ocupa dos líneas, que deben ejecutarse conjuntamente):

```
plot(rpc,pisa,lwd=2,col="red",cex=1.5,cex.lab=1.5,cex.axis=1.5,bty="n",  
     xlab="Renta per cápita, año 2012, en miles de €", ylab="Puntos PISA2012 en Matemáticas")
```

y que produce el resultado que aparece en la Figura 5. Vamos a comentar las modificaciones que

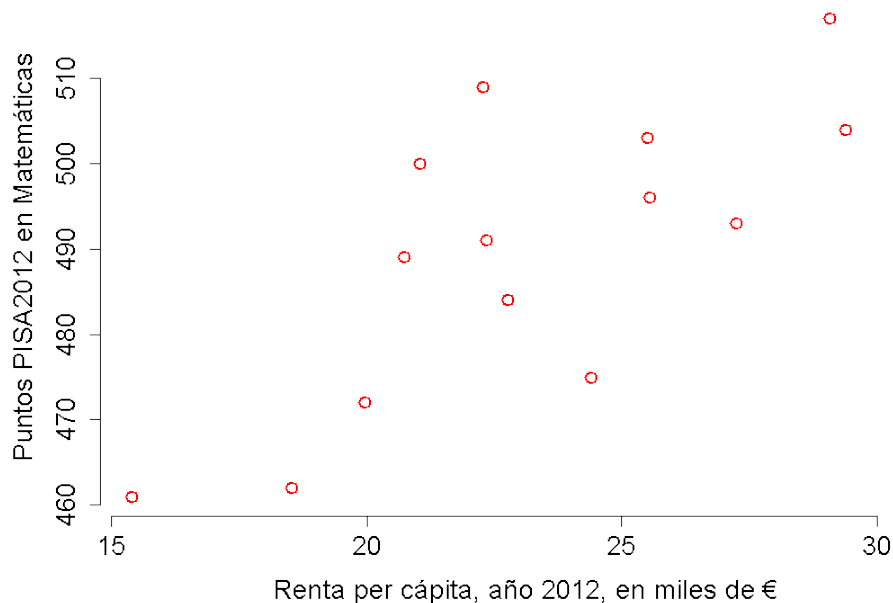


Figura 5: Primer paso para obtener un diagrama de dispersión mejorado para el Ejemplo 1.1.

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.

Ejercicios:

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. Para eso hemos creado un vector de nombres:

```
namesPisa=c("Extremadura","Andalucía","Murcia","Galicia","Asturias","CastillaLeón",
            "Cantabria","ESPAÑA","Balears","La Rioja","Aragón","Cataluña",
            "Navarra","Madrid","País Vasco")
```

Y luego hemos ejecutado el siguiente código, que enseguida comentaremos:

```
par(xpd=TRUE)
text(rpc, pisa, labels=namesPisa, pos=3, offset=0.6,font=2)
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 6. A continuación añadimos la recta con

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

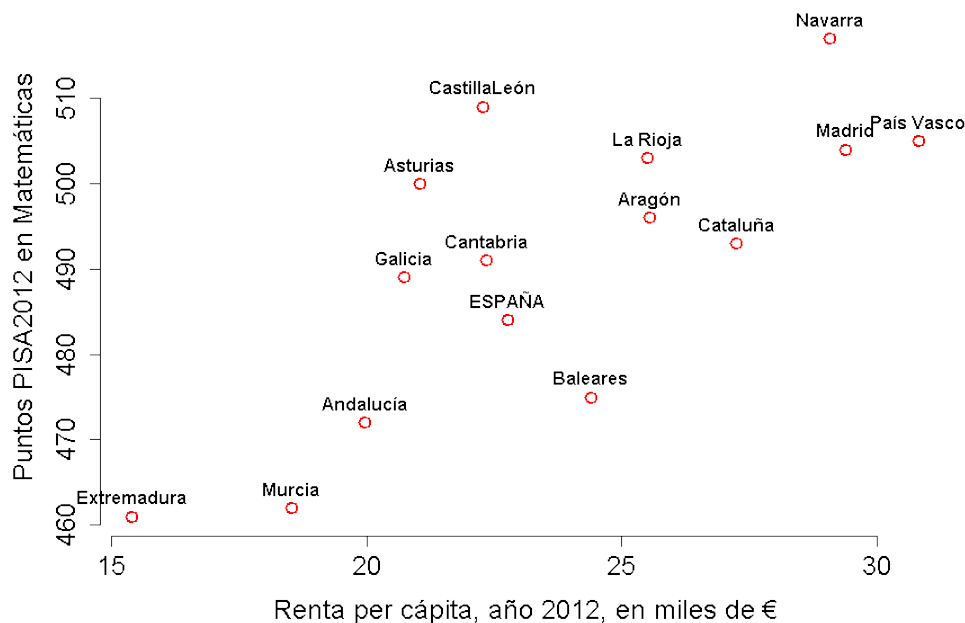


Figura 6: Segundo paso para obtener un diagrama de dispersión mejorado para el Ejemplo 1.1.

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(rpc,pisaRecta,rpc,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: 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="1")
```

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: `"1"`, `"7"`, `"c"`, `"u"`, `"["`, `"]"`

Ejercicio: Pruébalas todas para ver el efecto que producen. □

El resultado final es el gráfico de la Figura 7. 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).

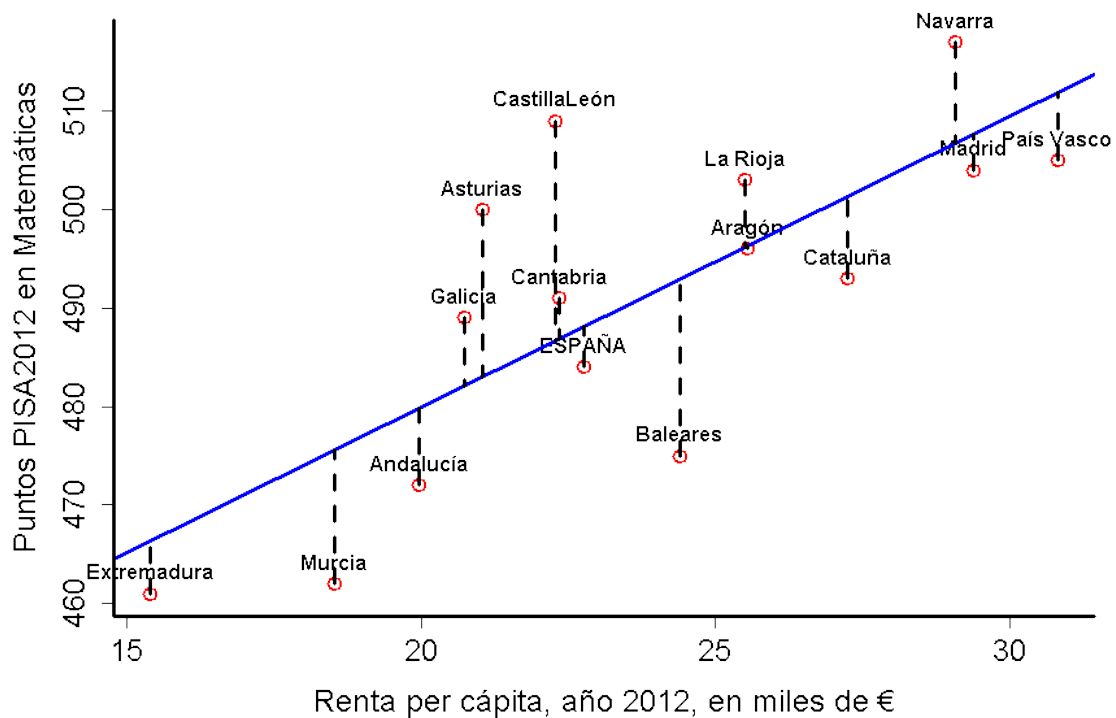


Figura 7: El diagrama de dispersión mejorado para el Ejemplo 1.1.

2. Regresión lineal con Calc, Wiris y Wolfram Alpha

Como no podía ser de otra manera, tratándose de una operación tan frecuente, Calc incluye funciones para obtener los valores necesarios para determinar la covarianza, la recta de regresión y el coeficiente de correlación.

En concreto, disponemos de las siguientes funciones:

1. `COVAR(Datos1;Datos2)`, que calcula la covarianza del conjunto de puntos.
¡ATENCIÓN! Calc usa n en lugar de $n - 1$. Así que la covarianza de Calc no coincide con la de R.
2. `PENDIENTE(Datos_y;Datos_x)`, que calcula la pendiente de la recta de regresión. *¡Atención al orden de las variables!*
3. `INTERSECCION.EJE(Datos_y;Datos_x)`, que calcula la pendiente de la recta de regresión. *¡Atención al orden de las variables!*
4. `COEF.DE.CORREL`, que calculan el valor de r

En esta hoja Calc

[Tut10-RectaRegresion.ods](#)

puedes ver como se aplican todos esos cálculos a una colección aleatoria de datos, y modificándola ligeramente puedes utilizarla para otros ejemplos.

También puedes usar este fichero de Wiris:

[Tut10-Wiris-RegresionLineal.html](#)

en el que se calcula la recta de regresión, covarianza y correlación para los mismos datos.

No vamos a decir gran cosa sobre Wolfram Alpha, porque no es muy cómodo de usar, salvo para conjuntos muy pequeños de datos. En cualquier caso, puedes hacerte una idea de las posibilidades que ofrece visitando esta galería de ejemplos:

<http://www.wolframalpha.com/examples/RegressionAnalysis.html>

3. Introducción a la función `lm` de R

Atención: esta sección es opcional en primera lectura, pero muy recomendable.

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, que tienes en el fichero `csv` [Cap10-Ejemplo-ANOVA02.csv](#), que ya hemos usado antes en los ejercicios de la página 7 de este tutorial. De hecho, puedes usar los resultados del ejercicio 2e de esa página para irlos comparando con los que obtendremos aquí.

Suponemos que el lector ha hecho los deberes, así que los datos de la muestra están almacenados en dos vectores `x` e `y`. Para explicarle a R que queremos usar el modelo de regresión lineal correspondiente a esos datos basta con ejecutar este comando (se muestra la salida):

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

Call:

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

Coefficients:

```
(Intercept)          x  
    0.9828      -0.4808
```

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)
```

con el que verás que R ha calculado muchas más propiedades de las que parecía.

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  
    0.9827585    -0.4808208
```

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 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
```

```
y
```

```
lmXY$residuals
```

Además, muchas funciones de R reconocen un objeto de tipo *modelo lineal*, y responden de forma interesante ante ese tipo de objetos.

Ejercicio: ¿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? \square

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. Puede que te estés preguntando si no se puede hacer, directamente,

```
plot(lmXY)
```

Y la respuesta es que 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. 286) del curso.

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  0.101651  0.10165   535.76 1.289e-08 ***
Residuals  8  0.001518  0.00019
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Como hemos dicho, en esta sección no queremos sino hacer una primera visita sucinta 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.

4. Inferencia en la regresión, usando R.

Atención: esta sección, que se corresponde con los resultados de la Sección 10.4 (páf. 281) del curso, es opcional, y sólo debe leerse si se ha leído aquella (que es, a su vez, opcional).

4.1. Simulando un modelo de regresión lineal simple en R

Recordemos la Ecuación 10.16(pág. 283) del curso, 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 curso 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) ) )
beta0 = 1
beta1 = -1/2
(y = beta0 + beta1 * x + rnorm(n,sd=0.01))
signif(y, digits = 2)
```

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. Para eso sirve el comando `set.seed(2013)` (puedes ver una explicación más detallada al final de este apartado, en la página 15).

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

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

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

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

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,

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

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

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

```
Call:
```

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

```
Coefficients:
```

```
(Intercept)          x
    0.9828      -0.4808
```

Como ves, los valores son

$$b_0 \approx 0.9828, \quad b_1 \approx -0.4808,$$

que se parecen, a β_0 y β_1 , pero ciertamente no coinciden, como queda de manifiesto en la Figura 10.15 (pág. 275) del curso.

Función `set.seed`. Unos comentarios sobre los números aleatorios en ejemplos y simulaciones. Parece un contrasentido generar unos números aleatorios que son siempre los mismos, pero esa es la forma de garantizar que nuestros ejemplos, o las simulaciones que hagamos, serán reproducibles por cualquiera (en este caso, por el lector). el valor que se coloca como argumento de `set.seed` es un número cualquiera. Podrá pensarse en que la mejor manera de hacer una simulación es haciendo que ese número sea un número cualquiera, por ejemplo eligiéndolo al azar para cada simulación. pero se desaconseja hacer eso, porque eso abre la puerta a que, en caso de querer manipular los datos, hayamos estado probando hasta elegir un número “aleatorio” que favorece a los resultados que queremos obtener. Así que lo mejor puede ser adoptar una política fija sobre el generador de números aleatorios, y aplicarla siempre. En mi caso, utilizo siempre el año en curso cuando escribo el código, para evitar ese tipo de suspicacias.

4.2. Otro vistazo a la descomposición ANOVA

Aunque ya hemos hecho una primera visita en la Sección 1.3 de este tutorial, ahora que hemos aprendido algo sobre la función `lm`, y hemos construido un ejemplo de modelo de regresión lineal simple, vamos a volver sobre la descomposición ANOVA, precisamente para ese ejemplo.

Primero vamos a obtener esa descomposición “a mano”, sin usar `lm`. El código necesario (con la salida) es:

```
> (TOTAL=(n-1)*var(y))
[1] 0.1031692
> (EC=sum((yRecta-y)^2))
[1] 0.001517854
> (RECTA=sum( (yRecta-mean(y))^2 ) )
[1] 0.1016513
> EC+RECTA
[1] 0.1031692
```

Tomando como referencia la notación de la Ecuación 10.9 (pág. 277), hemos llamado `TOTAL` a lo que allí se llama SST , `EC` al error cuadrático, que allí se llama $SS_{residual}$ y `RECTA` a lo que en el curso es SS_{modelo} . Como puede verse, la suma de los dos últimos coincide con el valor de `TOTAL`.

Ahora, usando la función `lm`, podemos obtener estos resultados de otra manera:

```
> (ANOVA=anova(lmXY))
Analysis of Variance Table

Response: y
      Df    Sum Sq Mean Sq F value    Pr(>F)
x       1 0.101651 0.10165   535.76 1.289e-08 ***
Residuals  8 0.001518 0.00019
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

La respuesta es el segundo ejemplo que vemos en este tutorial de lo que se conoce como una tabla ANOVA. Como puede verse, en la columna de la tabla de ANOVA titulada `Sum Sq` aparecen los valores que nosotros hemos llamado `RECTA` y `EC` respectivamente. Los valores de la columna `Mean Sq` son esos mismos valores, pero divididos por los grados de libertad adecuados (los que aparecen en la columna `Df`) (ver el Capítulo 11 para más detalles sobre las tablas ANOVA).

4.3. 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 10.18 (pág. 285) 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 10.20 (pág. 286) del curso, 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. 285) se han obtenido así (se muestra la salida):

```
> (talfamedios=qt(0.975,df=n-2))
[1] 2.306004
> (intConfBeta1=b1+c(-1,1)*talfamedios*sqrt(ECM/((n-2)*var(x))))
[1] -0.5287231 -0.4329184
```

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 (se muestra la salida):

```
> confint(lmXY,level=0.95)
                2.5 %      97.5 %
(Intercept)  0.9459612  1.0195558
x            -0.5287231 -0.4329184
```

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`).

4.4. Verificando las condiciones del modelo

En la Sección 10.4.2 (pág. 286) del curso 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:

```
(residuosSt=studres(lmXY))
```

Y una vez hecho esto, basta con usar las funciones `hist` y `boxplot` para obtener las partes (a) y (b) de la Figura 10.18 (pág. 288) del curso. 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 1.4 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)
> gvlma(lmXY)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)          x
      0.9828      -0.4808
```


ASSESSMENT OF THE LINEAR MODEL ASSUMPTIONS
 USING THE GLOBAL TEST ON 4 DEGREES-OF-FREEDOM:
 Level of Significance = 0.05

Call:

```
gvlma(x = lmXY)
```

	Value	p-value	Decision
Global Stat	3.6865	0.4501	Assumptions acceptable.
Skewness	0.3388	0.5605	Assumptions acceptable.
Kurtosis	0.5312	0.4661	Assumptions acceptable.
Link Function	2.1558	0.1420	Assumptions acceptable.
Heteroscedasticity	0.6606	0.4163	Assumptions acceptable.

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.

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 se muestra en la Figura 8 (pág. 18), 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.

4.5. Residuos atípicos y puntos influyentes

Vamos a empezar por el último de la serie de gráficos que se obtienen con `plot(lmXY)` (ver Figura 8, pág. 18). 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 página 290 del curso. 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. Para ayudar a visualizar lo que significa la *influencia* hemos preparado un fichero GeoGebra,

[Cap10-PuntosInfluyentes.html](#)

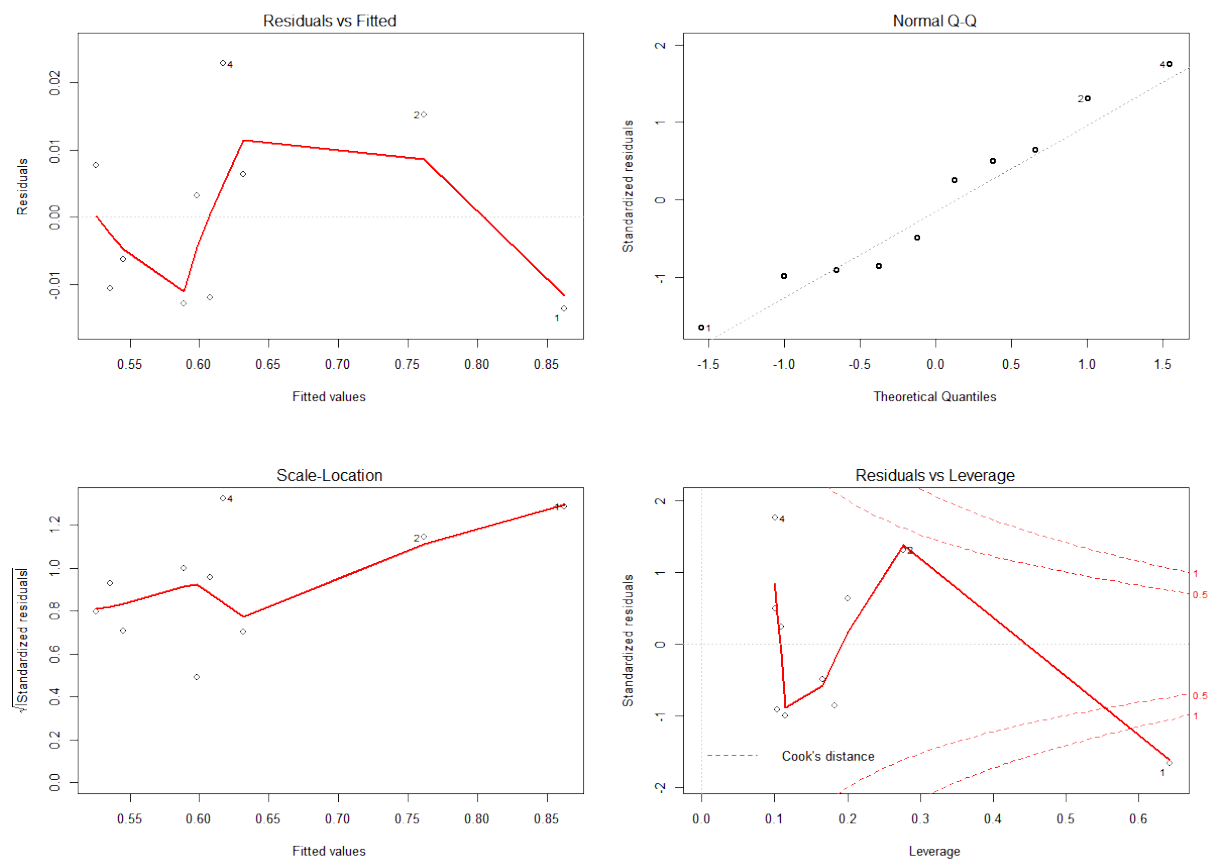


Figura 8: Serie de gráficos que se obtiene al usar `plot(lmXY)` en R.

que puedes usar para ver el efecto que tiene, sobre la recta de regresión, la eliminación de cada uno de los puntos de la muestra por separado. Además, en R, disponemos de la función `cooks.distance` para obtener esa distancia de Cook. Aplicada a nuestro ejemplo produce estos valores:

```
> cooks.distance(lmXY)
      1          2          3          4
2.484471697 0.326624332 0.013341172 0.172228877
      5          6          7          8
0.048685349 0.003567851 0.064509786 0.025065495
      9         10
0.082611091 0.050429876
```

que confirman lo que veíamos en la Figura 8: 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.

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:

```
> outlierTest(lmXY)

No Studentized residuals with Bonferonni p < 0.05
Largest |rstudent|:
      rstudent unadjusted p-value Bonferonni p
4 2.085574          0.075462          0.75462
```

y nos informa de que no existen residuos atípicos.

Volvamos a la medida de la influencia. Como hemos dicho en la página 290, la influencia de un punto tiene que ver con su “brazo de palanca”. En inglés, palanca (en este sentido) se dice *lever*, y por eso existe una medida de la influencia denominada *leverage*. En español no hay, por lo que sé, una traducción universalmente aceptada para este término, aunque he visto utilizar “efecto palanca” para referirse a esto (y me parece adecuado).

Para medir ese efecto palanca (leverage), se utiliza otro concepto relacionado, que son los (a falta de un nombre mejor) valores sombrero (en inglés, *hat values*). Estos valores, forman una matriz $n \cdot n$, la matriz sombrero H (hat matrix, en inglés), que se representa así:

$$H = \begin{pmatrix} h_{11} & \cdots & h_{1n} \\ & \ddots & \\ h_{n1} & \cdots & h_{nn} \end{pmatrix}$$

y que tiene la propiedad de que:

$$(\hat{y}_1, \dots, \hat{y}_n) = (\hat{y}_1, \dots, \hat{y}_n) \cdot H, \quad (\text{producto matricial}).$$

Es decir, que para cualquier $i = 1, \dots, n$ es:

$$\hat{y}_j = h_{1j} \cdot y_1 + h_{2j} \cdot y_2 + \cdots + h_{nj} \cdot y_n. \quad (1)$$

Esta relación muestra de donde proviene el nombre de la matriz H , y es porque transforma las y_j en las \hat{y}_j (H le pone el sombrero a las y_j).

¿Por qué son importantes estos valores sombrero h_{ij} al tratar de medir la influencia? Imagínate que, manteniendo los mismos valores de x_1, \dots, x_n , cambiásemos los valores y_i . Esta matriz nos diría cuáles serían los nuevos valores \hat{y}_i . Es decir, que esta matriz construye la recta de regresión. Además, la diagonal de esta matriz tiene una propiedad muy importante. Para cualquier elemento h_{jj} de la diagonal se tiene:

$$h_{ii} = h_{i1}^2 + h_{i2}^2 + \cdots + h_{in}^2. \quad (2)$$

Y además, el valor h_{ii} sólo depende de los x_i , como queda de manifiesto en esta relación:

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{j=1}^n (x_j - \bar{x})^2}$$

Los valores que aparecen elevados al cuadrado aquí (los de la fila i -ésima de H) son los que, de acuerdo con la Ecuación 1, determinan el peso que tiene el ingrediente y_i a la hora de calcular cada uno de los \hat{y}_j . Es decir, que determinan el peso que tiene el valor y_i , asociado con el i -ésimo valor x_i de la muestra. Puesto que además se cumple la Ecuación 2, cada uno de los valores

$$h_{11}, h_{12}, \dots, h_{nn}$$

puede utilizarse como un indicador de la influencia global sobre el modelo (sobre la recta) del valor x_i . Eso significa que podemos usar esos valores para medir el efecto palanca de los x_i .

Afortunadamente, calcular estos valores con R es muy fácil. Para obtenerlos en el ejemplo que estamos usando, basta con ejecutar (se muestra la salida)

```
> hatvalues(lmXY)
      1      2      3      4      5
0.6438400 0.2770361 0.1001842 0.1010030 0.1038231
      6      7      8      9     10
0.1084628 0.1149219 0.1665037 0.1829698 0.2012554
```

Estos valores confirman que el efecto palanca (leverage), y por tanto la influencia, del primer punto es grande. Como regla práctica se utiliza el criterio de considerar como influyentes aquellos puntos x_i cuya palanca (el valor h_{ii} correspondiente) es mayor que dos veces el valor palanca medio, que es sencillo ver que viene dado por:

$$\bar{h} = \frac{2}{n}.$$

En nuestro ejemplo, eso significa que $\bar{h} = \frac{2}{10} = 0.2$. Así que un punto influyente, según este criterio, sería aquel cuyo efecto palanca sea mayor que 0.4. Como se ve, en este ejemplo obtenemos el mismo resultado que cuando usábamos la distancia de Cook para medir la influencia. El segundo punto de la muestra tiene un valor mayor que la media, pero no llega a ser influyente.

4.6. El cuarteto de Anscombe

Ninguna discusión del modelo de regresión lineal simple estaría completo sin incluir esta colección de ejemplos, ya clásicos, debidos al estadístico inglés Frank Anscombe.

http://en.wikipedia.org/wiki/Francis_Anscombe

Se trata de cuatro muestras, cada una de ellas formada por 11 puntos (x_i, y_i) , que tienen muchas propiedades estadísticas prácticamente iguales. Por ejemplo, las cuatro muestras tienen los mismos valores de

- $\bar{x} = 9$
- $\bar{y} \approx 7.50$
- $s_x^2 = 11$
- $s_y^2 \approx 4.1$
- $\text{Cov}(x, y) \approx 5.5$
- $r \approx 0.816$

y en particular, en los cuatro casos la recta de regresión es aproximadamente (con tres cifras significativas):

$$y = 3 + 5 \cdot x.$$

Sin embargo, los diagramas de dispersión de los cuatro casos, que aparecen en la Figura 9 muestran que las cuatro situaciones son claramente distintas.

- En el primer caso, la recta de regresión es un buen modelo del conjunto de puntos.

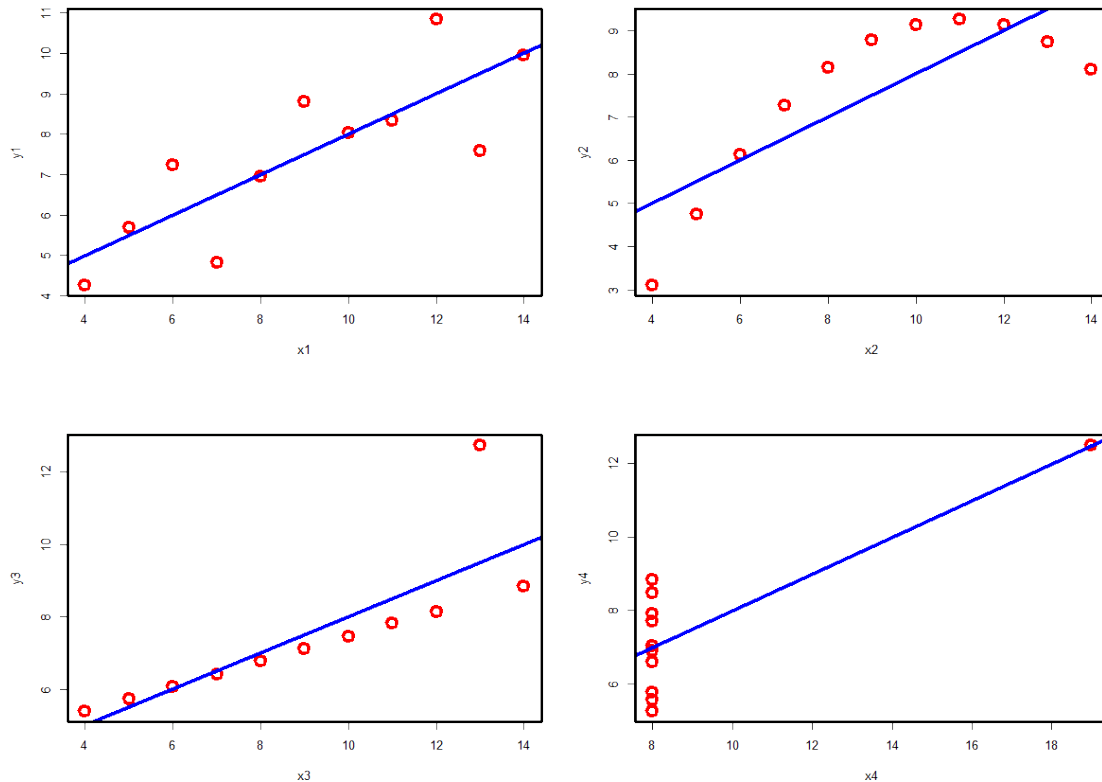


Figura 9: Diagramas de dispersión de los cuatro casos del *Cuarteto de Anscombe*.

- En el segundo caso, la relación entre las variables X e Y es, obviamente, no lineal, y lo que se necesita es un ajuste polinómico. Volveremos sobre este caso en el apartado 5 de este tutorial.
- El tercer caso contiene un punto (concretamente el (x_3, y_3)) con un residuo atípico, que además es influyente (su *efecto palanca* no es grande, pero su distancia de Cook es mayor que uno).
- El cuarto caso es, en algún sentido, el más patológico. Todos los valores x_i son iguales excepto uno (x_8 para ser precisos). Así que si se elimina el punto (x_8, y_8) , los restantes puntos están en una recta vertical (y el modelo de regresión que hemos visto en el Capítulo 10 no sirve, porque no es capaz de producir rectas verticales). Es interesante observar que el punto (x_8, y_8) es, obviamente, influyente, pero que su residuo no es atípico.

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 facilitar el trabajo, el fichero:

[Tut10-Anscombe.R](#)

contiene código R para analizar los ejemplos del *Cuarteto de Anscombe*.

Una primera conclusión, a la luz de estos cuatro ejemplos es que el coeficiente de correlación r , no puede servir por sí mismo como indicador de la calidad de un modelo de regresión lineal. Pero, abundando en esa dirección, la lección más importante que hay que extraer de estos ejemplos es que, sin explorar los datos, ningún análisis de regresión puede considerarse completo (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. Regresión polinómica con R

Para terminar, 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 9) 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 más complicada. Para ilustrarlo, empezamos por crear dos vectores con las coordenadas x e y de este ejemplo (se muestra la salida):

```
> (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 (se muestra la salida):

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

Call:

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

Coefficients:

(Intercept)	I(x)	I(x^2)
-5.9957	2.7808	-0.1267

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

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

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 con (se muestra la salida):

```
> (b=lmPolXY$coefficients)
(Intercept)      I(x)      I(x^2)
-5.9957343    2.7808392   -0.1267133
```

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 11 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)`. En el fichero

[Tut10-RegresionPolinomica.R](#)

está el código R que hemos usado para producir esa figura.

Terminamos este apartado proponiendo al lector un ejercicio:

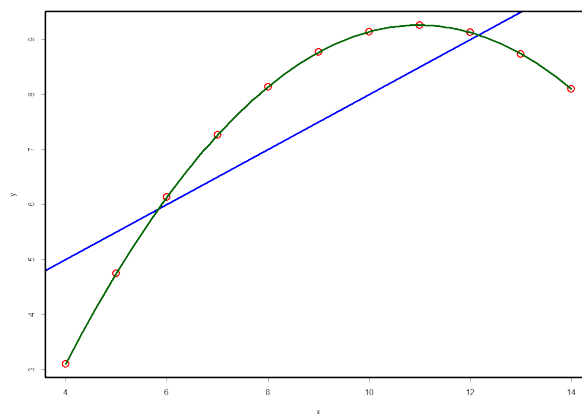


Figura 10: Ejemplo de regresión polinómica con R.

Ejercicio: Vamos a buscar una parábola para los datos del Ejemplo 10.3.1 (pág 269) del curso. Los datos están en el fichero:

Cap10-EjemploRectaMalaAproximacion01.csv

que puedes encontrar en el ejercicio 2(b) de la pág. 7 de este tutorial. Los valores de ese ejemplo se han generado en R con el código:

```
(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 $\beta_0 = 0, \beta_1 = 1, \beta_2 = -1$, y $\epsilon \sim N(0, 0.001)$. Naturalmente, cuando obtengas b_0, b_1 y b_2 , sus valores no coincidirán exactamente con los de $\beta_0 = 0, \beta_1 = 1$ 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) . La que he obtenido yo se muestra en la Figura 11.

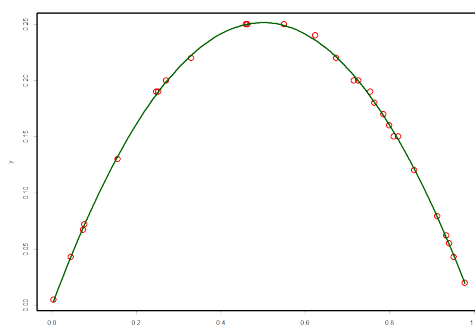


Figura 11: Parábola para los datos del Ejemplo 10.3.1 del curso.

□

Muchas gracias por la atención.