Notas de R para Econometría

Jesús Antonio Piñera Esquivel

Índice

1. Introducción a R

1.1. Instalación

Para instalar R, ingresar a la página https://cran.r-project.org/mirrors.html, elegir cualquier servidor (el ITAM tiene un servidor para R, como dato curioso), elegir el sistema operativo adecuado y seguir las instrucciones según la elección.[1]

Después de descargar R, vale la pena descargar RStudio, que es un entorno para R. El link para descargarlo es https://rstudio.com/products/rstudio/download/, la versión gratis. Una vez instalados R y RStudio, podrán abrir e interactuar con la interfaz de RStudio:

[1] Instalar R en Linux es más complicado y va más allá del alcance de este documento. Se recomienda acercarse al departamento de computación si se requiere dicha instalación.

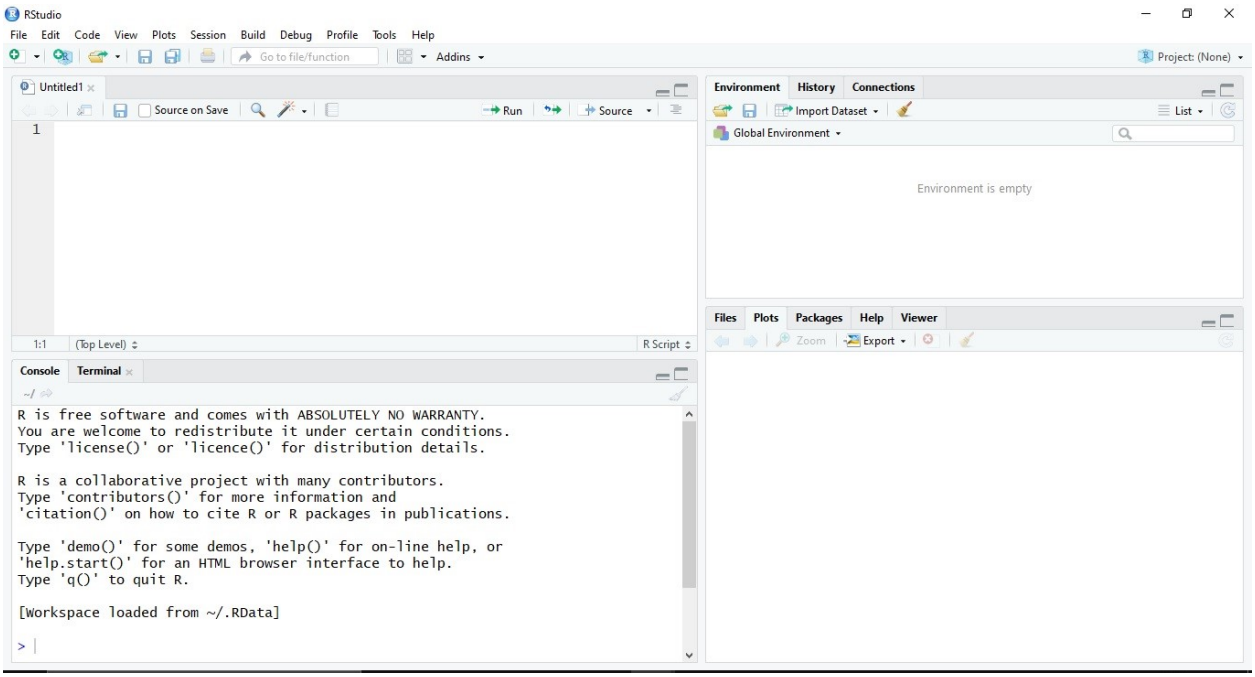


Figura 1: Interfaz de RStudio

Se pueden observar cuatro ventanas con múltiples pestañas. Entre ellas, las que se usarán son *Environment, plots, packages, help* y *console*. Asimismo, la ventana superior izquierda muestra el *script*, que es básicamente un editor de texto que permite guardar en formato .R, pero RStudio puede leer un archivo tipo .txt y correrlo. Se prefiere utilizar el ambiente interno de RStudio, pero eso queda a preferencia del usuario. Las pestañas mencionadas muestran lo siguiente:

- *Environment*: todos las variables definidas a lo largo de la sesión. Úil para visualizar matrices o *data frames*.[2]
- *Plots*: cualquier gráfica que se realice en un script o en la consola se mostrará en esta pestaña.
- *Packages*: los paquetes que R tiene instalados. Si se quieren instalar otros paquetes, basta con ir a esta pestaña, dar click en **Install** en la esquina superior izquierda, y en Packages escribir el nombre del paquete.
- *Help*: ayuda para cualquier función o paquete de R instalados. Sumamente útil para saber qué hace cada función y los argumentos que requiere.
- *Console*: donde se "corre" el programa.

1.2. Funciones básicas de R

Como cualquier lenguaje de programación, R posee su propia sintáxis, la cual es muy afable. A continuaciín se presentan algunas de las funciones más elementales de R.

1.2.1. Asignación de valores y operaciones elementales

Para asignar valores, se puede utilizar el signo de igual o una flecha, pero se prefiere la flecha. Las operaciones elementales se presentan a continuación:

[2]Cabe recalcar que un *data frame* no es una matriz, aspi que no se pueden utilizar las funciones matriciales de R con *data frames*.

```r
X <- 5
Y = 4
X
```

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

```r
Y
```

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

```r
X + Y
```

```
## [1] 9
```

```r
X * Y
```

```
## [1] 20
```

```r
X / Y
```

```
## [1] 1.25
```

```r
X ^ Y
```

```
## [1] 625
```

1.2.2. Vectores y matrices

Para crear un vector, se pueden realizar dos comandos. El primero es $c()$, el cual crea un vector con las entradas introducidas por el usuario. En este vector se puede introducir cualquier tipo de variable.

```r
x <- c(0,1,2,0==0,"Mexico", 1>2)
x
```

```
## [1] "0"      "1"      "2"      "TRUE"   "Mexico" "FALSE"
```

También se pueden crear vectores numéricos que se asemejan a una partición de un intervalo $[a, b]$ en n pedazos.

```r
a <- 0
b <- 1
n <- 10
x <- seq(a,b,length.out = n)
x
```

```
##  [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
##  [8] 0.7777778 0.8888889 1.0000000
```

Nótese que la función anterior es inclusiva de los extremos.

Por otro lado, las matrices se pueden crear con el siguiente comando:

```r
n <- 3
m <- 3
A <- matrix(nrow = n, ncol = m)
A
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
## [3,]   NA   NA   NA
```

Al crear una matriz de esta forma, las entradas son todas NA. Para llenar la matriz, se pueden realizar operaciones con ella o poner el argumento $data = a$ con a algún valor numérico.

```r
A <- matrix(data = 0, nrow = 3, ncol = 3)
A
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0
```

```r
(A+2)*8
```

```
##      [,1] [,2] [,3]
## [1,]   16   16   16
## [2,]   16   16   16
## [3,]   16   16   16
```

Más adelante se verá címo referenciar objetos para ingresar valores en ciertas entradas de un objeto. Por ejemplo,

```r
A[1,] <- c(1,2,3)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    0    0    0
## [3,]    0    0    0
```

```r
A[,2] <- c(pi, exp(1), sqrt(2))
A
```

```
##      [,1]     [,2] [,3]
## [1,]    1 3.141593    3
## [2,]    0 2.718282    0
## [3,]    0 1.414214    0
```

Nótese que en el código anterior se introdujo *pi* como π. Este es el valor predeterminado, así que hay que evitar utilizarlo como variable.

```
pi
```

```
## [1] 3.141593
```

```
pi <- 3
pi
```

```
## [1] 3
```

Finalmente, en R las operaciones entre objetos **siempre** son entrada a entrada. Por ejemplo

```
x <- matrix(nrow = 1, ncol = 3)
x[1,] <- c(1,2,3)
x*x
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    9
```

```
A*A
```

```
##      [,1]     [,2] [,3]
## [1,]    1 9.869604    9
## [2,]    0 7.389056    0
## [3,]    0 2.000000    0
```

Para realizar operaciones matriciales (ya sea entre vectores o matrices), el operador se tiene que poner entre los símbolos % %

```
x %*% t(x)
```

```
##      [,1]
## [1,]   14
```

```
t(x) %*% x
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    4    6
## [3,]    3    6    9
```

```
A %*% A
```

```
##      [,1]      [,2] [,3]
## [1,]    1 15.923968    3
## [2,]    0  7.389056    0
## [3,]    0  3.844231    0
```

1.2.3. Manejo de índices y referencias

Referenciar una variable es muy sencillo y se puede hacer de muchas maneras. Para ilustrar, creamos una matriz de 5×3 llena de ceros.

```
A <- matrix(data = 0, nrow = 5, ncol = 3)
```

Es decir, 5 renglones por 3 columnas. R automáticamente asume que los renglones son datos y las columnas son variables. Si la base de datos se encuentra al revés (i.e. las columnas son variables), habrá que transponer la matriz con la funciÃ³n $t()$.

Ahora, para referencias, digamos, la columna 2 completa, podemos hacer lo siguiente para llenarla con algunos valores

```
A[,2] <- c(1,2,3,4,5)
A
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    2    0
## [3,]    0    3    0
## [4,]    0    4    0
## [5,]    0    5    0
```

Es importante tener en cuenta las dimensiones de los objetos al darles valores, pues si las dimensiones no concuerdan, R marcará un error.

Se pueden referenciar renglones o columnas enteras, partes de renglones o partes de columnas, e incluso introducir cierta clase de «booleanos» (valores lógicos) como «-1» para quitar el primer elemento

```
A[1,]               ## Primer renglon completo
```

```
## [1] 0 1 0
```

```
A[,3]               ## Tercera columna completa
```

```
## [1] 0 0 0 0 0
```

```
A[c(1,2), c(1,2,3)] ## Primer y segundo renglon; primera, segunda y tercera columna
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    2    0
```

```
A[-1,]              ## Todo menos el primer renglon
```

```
##      [,1] [,2] [,3]
## [1,]    0    2    0
## [2,]    0    3    0
## [3,]    0    4    0
## [4,]    0    5    0
```

```
A[,-3]              ## Todo menos la tercera columna
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]    0    2
## [3,]    0    3
## [4,]    0    4
## [5,]    0    5
```

```
A[A[1,] == 0,]      ## Aquellos renglones donde el primer renglon sea igual a cero
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    3    0
## [3,]    0    4    0
```

Nótese que el booleano anterior se repite a lo largo de la matriz. Es decir

```
A[c(TRUE,FALSE),]       ## Salta cada segundo renglon y repite
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    3    0
## [3,]    0    5    0
```

```
A[c(FALSE,TRUE),]       ## Salta cada primer renglon y repite
```

```
##      [,1] [,2] [,3]
## [1,]    0    2    0
## [2,]    0    4    0
```

```
A[c(TRUE,TRUE,FALSE),]  ## Dos si, uno no
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    2    0
## [3,]    0    4    0
## [4,]    0    5    0
```

Existen formas más complejas para referenciar objetos en R, así como objetos aún más complejos como *arrays*, pero para motivos del curso, con lo visto es suficiente.

1.2.4. Lectura de datos

La funcián que más se utilizará para la lectura de datos es *read.csv()*, la cual lee archivos con formato .csv:

```
dat <- read.csv(file = "datos.csv")
```

Existen otras funciones para cargar datos en formatos .xlsx, .rds o .dbt, pero la mayoría de las bases de datos aún se guardan en Excel, lo cual hace guardar archivos en formato .csv más sencillo.

Caben mencionar ciertas puntualidades importantes a las que se podrían enfrentar al leer archivos:

- Los datos usualmente tienen un encabezado, así que se utiliza como pretederminado el valor de TRUE en *header*. En caso de que la base de datos no tenga encabezados, basta con poner *header* = FALSE.
- La base de datos puede tener una separación distinta a una coma (que es el predeterminado). Esto es muy usual en macOS, que a veces guarda archivos .csv separados por punto y coma (;). Para solucionar esto, simplemente hay que introducir el argumento *sep* a la función.
- Al leer la base de datos, es muy probable que tenga entradas vacías. Tienen que introducir en la función la variable *na.strings* de tal forma que sepa R qué formato tienen las celdas vacías y las guarde como N/A. Se pueden introducir múltiples formatos al mismo tiempo (e.g. si una misma base de datos pone N/A, NA y * en los datos faltantes).
- R es sensible a mayúsculas y minúsculas, así que distingue entre *pib*, *Pib* y *PIB*.
- La lectura de un archivo .csv es literal, es decir, si en el archivo .csv se encuentra «$500.00», se leerá un elemento de tipo *string* (texto) o *factor* y no del tipo *float* (punto flotante, o número). Si el objetivo es leer 500, conviene cambiar el formato del .csv en alguna plataforma como Excel para evitar confusiones por el estilo o limpiar la base de datos en R.
- Evitar alterar el .csv de tal forma que se dejen celdas vacías, pues R puede llegar a leer dichas celdas como objetos nulos, o *N/A*'s. Es preferible copiar y pegar en otro archivo.
- Es preferible que el archivo .csv con los datos se encuentre en la misma carpeta que el código, pues por defecto R toma el directorio de trabajo como la carpeta en donde se encuentre el *script*. En caso de que este no sea el caso, se puede utilizar el comando Ctrl+Shft+H para elegir el directorio de trabajo de manera manual.
- En caso de que se utilice Excel como el lector de archivos, evítese el uso de fórmulas dentro del .csv, pues estas no pueden ser leídas.

A manera de ejemplo de lo anterior, podemos volver a leer la base de datos

```
datos <- read.csv("datos.csv", header = TRUE, sep = ",", na.strings = c("N/A","NA","","*"))
```

El comando anterior guardará el archivo como clase *data frame*, que es una de las clases más utilizadas en el análisis de datos en R y en otros lenguajes como Python. Para ver el encabezado de un *data frame* se puede utilizar la función *head*().

```
head(datos)
```

```
##   X Persona Edad Sexo Ingreso
## 1 1       1   21    1  141080
## 2 2       2   27    1  205230
## 3 3       3   32    1  224710
## 4 4       4   37    1  260170
## 5 5       5   42    1  273140
## 6 6       6   47    1  280930
```

Como ya se vio, la referencia en forma *objeto*$[\ n, m\ ]$ es general y aplica a cualquier tipo de variable con mútiples entradas en R (e.g. matriz, vector, *array*), mientras que el comando $ se puede utilizar en *data frames*. Es decir, dada la matriz A, se puede recuperar la entrada en el renglón 3, columna 2 con el comando

```
A[3,2]
```

```
## [1] 3
```

De igual forma, dado el *data frame* **datos** definido anteriormente, se puede recuperar la entrada en al fila 121, en la columna 2 con el comando

```
datos[10,3]
```

```
## [1] 21
```

pero, también utilizando el símbolo $ y eligiendo la variable adecuada.

```
datos$Edad[10]
```

```
## [1] 21
```

Como se puede observar, ambos comandos arrojan el mismo valor. Esto es muy útil al analizar bases de datos, pues se les establecer nombres a las columnas de un *data frame* y en lugar de buscar el número de la columna de la variable deseada, basta con utilizar el simbolo $ y el nombre de la columna que se estableció.

Para cambiar el nombre de las columnas de un *data frame* se utiliza la función *colnames*()

```
colnames(datos)
```

```
## [1] "X"       "Persona" "Edad"    "Sexo"    "Ingreso"
```

```
colnames(datos) <- c("Z","W", "Edad", "Sexo", "Ingreso")

colnames(datos)
```

```
## [1] "Z"       "W"       "Edad"    "Sexo"    "Ingreso"
```

```
colnames(datos) <- c("X","Persona", "Edad", "Sexo", "Ingreso")
```

Al escribir nombres de las columnas hay que evitar acentos o símbolos fuera de punto y guión bajo (e.g. &, %, $, #) y hacerlo en formato *string*, i.e., entre comillas, para que R entienda que es texto.

1.2.5. Funciones básicas para el análisis de datos

Ahora que tenemos cargada nuestra base de datos, que llamamos **datos**, vamos a explorar distintas funciones útiles para su análisis. La primera, y la más sencilla, es *summary*(), la cual da un resumen estadístico simple de la distribución de los datos en las columnas del *data frame*.

```
summary(datos)
```

```
##        X            Persona           Edad            Sexo        Ingreso
##  Min.   : 1.00   Min.   : 1.00   Min.   :21.00   Min.   :0.0   Min.   :111540
##  1st Qu.: 5.25   1st Qu.: 5.25   1st Qu.:32.00   1st Qu.:0.0   1st Qu.:157605
##  Median : 9.50   Median : 9.50   Median :42.00   Median :0.5   Median :162160
##  Mean   : 9.50   Mean   : 9.50   Mean   :41.89   Mean   :0.5   Mean   :197473
##  3rd Qu.:13.75   3rd Qu.:13.75   3rd Qu.:52.00   3rd Qu.:1.0   3rd Qu.:260005
##  Max.   :18.00   Max.   :18.00   Max.   :62.00   Max.   :1.0   Max.   :280930
```

Otras funciones útiles son *mean*(), *var*() y *cor*(), las cuales obtienen la media muestral $\bar{X}$, la varianza muestral S^2 y la correlación muestral. Asimismo, la función *var*() calcula la covarianza muestral si se ingresan dos variables.

```
mean(datos$X)
```

```
## [1] 9.5
```

```
var(datos$Ingreso)
```

```
## [1] 3266546033
```

```
var(datos$Edad,datos$Ingreso)
```

```
## [1] 294842.7
```

```
cor(datos$Edad,datos$Ingreso)
```

```
## [1] 0.3831508
```

```
colSums(datos)
```

```
##       X Persona    Edad    Sexo Ingreso
##     171     171     754       9 3554510
```

Como las anteriores, existen muchísimas funciones como *max*, *min*, *quantile*, *median*, e incluso *colSums* y *colMeans* para obtener sumas y medias muestrales sobre columnas. Además, existe un gran número de paquetes, cada uno con funciones distintas. Realizar una lista exhaustiva en un documento es imposible, pues basta buscar en cualquier buscador en línea lo que se desea hacer en R para encontrar documentación exhaustiva al respecto. Incluso la interfaz de RStudio provee de la pestaña *Help*, la cual es de mucha ayuda para buscar y entender ciertas funciones.

2. Estimación por mínimos cuadrados ordinarios

2.1. Regresión lineal simple

Dada la ecuación

$$Y_i = \alpha + \beta X_i + \varepsilon_i \tag{1}$$

se busca estimar α y β con los datos $Y = (Y_1, Y_2, ..., Y_N)'$ y $X = (X_1, X_2, ..., X_N)'$. Tenemos la estimación de Y_i dada por

$$\hat{Y}_i = \hat{\alpha} + \hat{\beta} X_i \tag{2}$$

Definimos el residuo como

$$e_i = Y_i - \hat{Y}_i \tag{3}$$

El problema se reduce a minimizar la SCR dada por

$$\text{SCR} = \sum_{i=1}^{N} e_i^2 \tag{4}$$

eligiendo $\hat{\alpha}$ y $\hat{\beta}$. Así, el problema en cuestión es

$$\min_{\{\hat{\alpha},\hat{\beta}\}} \sum_{i=1}^{N} e_i^2$$

Realizar dicha estimación es sencilla con métodos numéricos y computacionales. En R se puede utilizar la función *lm()*. Para conocer todas las entradas de la función, se puede ingresar en la consola (*Console*, debajo del *script*) el comando **?lm**. En las que nos vamos a enfocar son *formula* y *data*. A continuación se presenta un código estándar de R para realizar una regresión lineal simple:

$$\text{modelo <- lm(formula = Y} \sim \text{X, data = datos)}$$

A manera de ejemplo, tomemos X una partición del segmento $[0, 1]$ de tamaño 100 y $Y = f(X) + \varepsilon_i$, donde $\varepsilon_i \sim N(0, \sigma^2)$. Tomemos el caso más simple con $f(x) = a + bX$, donde $a, b \in \mathbb{R}$ y $\sigma^2 = 1$. El código es el siguiente [3]:

```
X <- seq(0,1,length.out = 100) ## Esto es un comentario
a <- 3
b <- 10
Y <- a + b*X + rnorm(100,0,1)
modelo <- lm(formula = Y ~ X)
```

La variable **modelo** guarda distintos valores pertinentes al análisis de regresión. Por default, siempre se incluye el intercepto de la regresión. Si se quiere revisar un resumen general del modelo, se puede utilizar la función *summary()* sobre el objeto *lm*, la cual arroja: los cuartiles de los residuos; de los coeficientes, el valor estimado, el error estándar, el estadístico t y el *p-value* de dicho estadístico; códigos de significancia, el error estándar de la regresión (o de los residuos, S^2); el coeficiente de determinación R^2 y coeficiente de determinación ajustado por grados de libertad $\bar{R}^2$; y el estadístico F, con los grados de libertad y su *p-value*.

```
summary(modelo)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.21655 -0.62082  0.03984  0.60689  3.01724
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6850     0.2083   12.89   <2e-16 ***
## X            10.4016     0.3599   28.90   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.05 on 98 degrees of freedom
## Multiple R-squared:  0.895,  Adjusted R-squared:  0.8939
## F-statistic: 835.1 on 1 and 98 DF,  p-value: < 2.2e-16
```

[3] Los comentarios dentro del código se incluyen después de los símbolos ##, los cuales están escritos de manera incorrecta pues R no es capaz de leer acentos ni la letra *ñ*. Además, nótese que se prefiere utilizar la asignación «<-» en lugar de «=», pero son indistintas.

Asimismo, se pueden revisar valores específicos del modelo con el símbolo $. Es decir, si se quieren obtener los coeficientes, basta con el siguiente comando:

```
modelo$coefficients
```

```
## (Intercept)           X
##    2.685039   10.401602
```

Se puede escribir «modelo$» y revisar todos los valores que se obtienen con la función *lm()*, o bien utilizar *View*(modelo) para revisar la lista completa de valores que incluye **modelo**. Para mayor información, revísese la ayuda provista por R en la pestaña de *Help* (la cual, recalco, es sumamente útil).

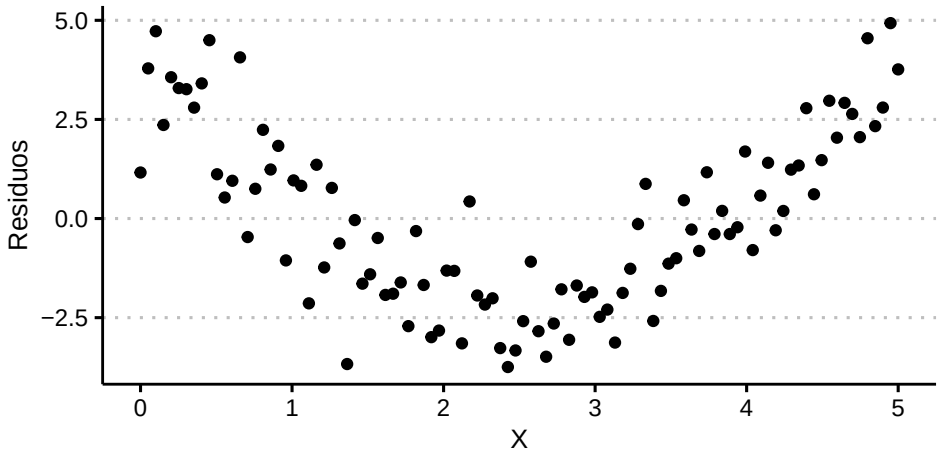
Continuando con el ejemplo anterior, ahora tomamos $f(x) = x^2$, $\sigma^2 = 1$ y X una partición de [0,5] de tamaño 100:

```
X <- seq(0,5,length.out = 100)
Y <-X^2 + rnorm(100,0,1)
modelo <- lm(Y ~ X)
summary(modelo)
```

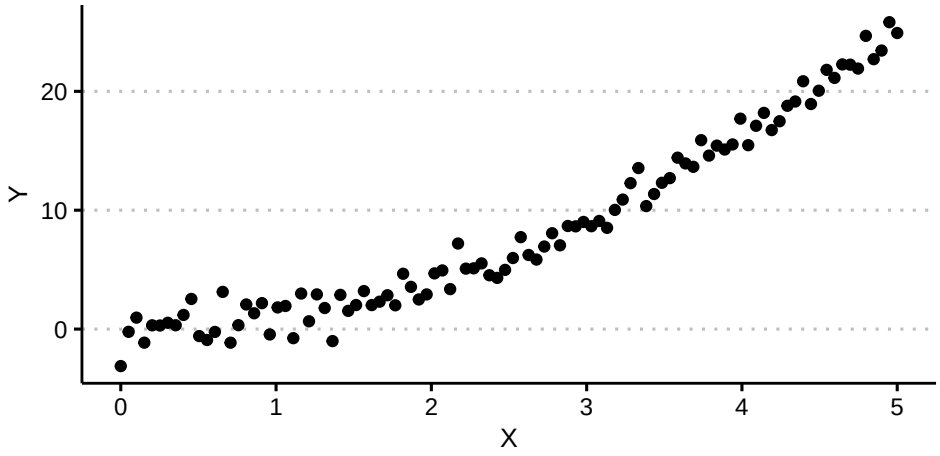
```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7450 -1.8655 -0.2869  1.4229  4.9288
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -4.2745     0.4487  -9.526 1.29e-15 ***
## X             5.0843     0.1550  32.793  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.26 on 98 degrees of freedom
## Multiple R-squared:  0.9165, Adjusted R-squared:  0.9156
## F-statistic:  1075 on 1 and 98 DF,  p-value: < 2.2e-16
```

Si analizamos los residuos de manera visual, se observa que estos no tienen un comportamiento completamente aleatorio, pues muestran una clara tendencia que no es capturada por el modelo.[4]

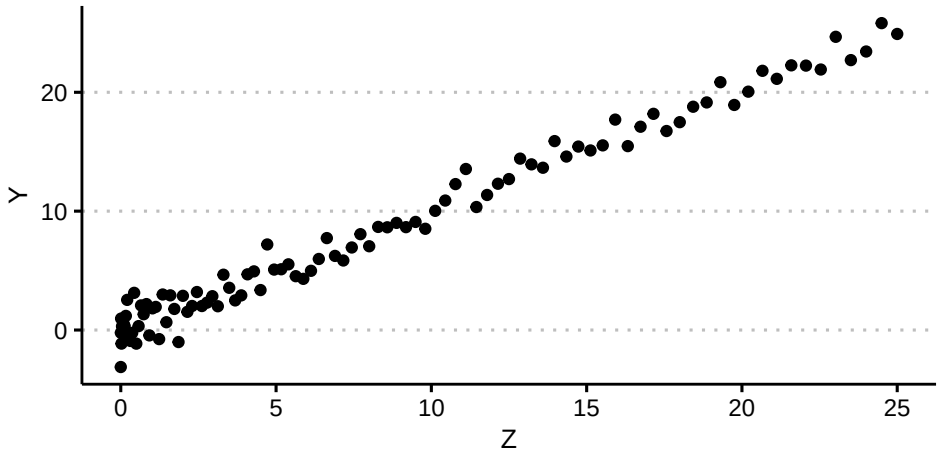
[4]Se utilizará la paquetería de *ggplot* para realizar las gráficas, pero su dominio no es fundamental, aunque recomendado.



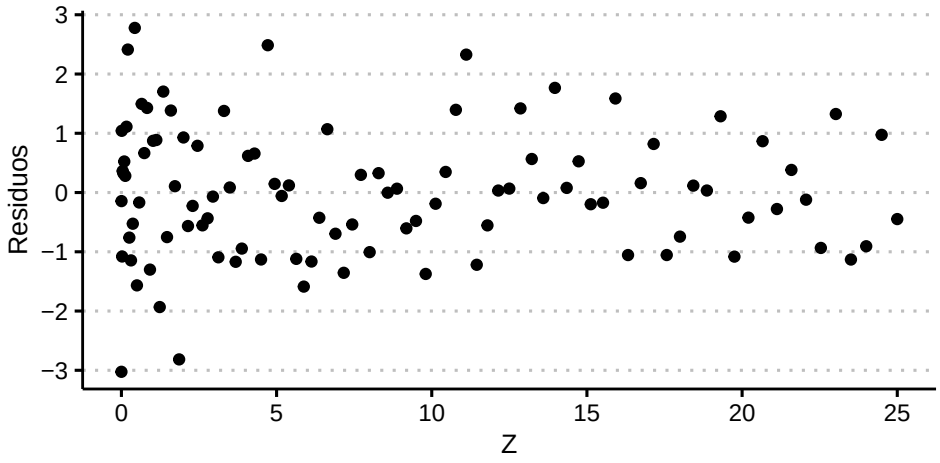
Para solucionar esto, se puede transformar la variable X tal que Y sea lineal en X, pues notamos gráficamente que Y no es una función lineal en X.



Así, se puede sugerir la transformación $Z = X^2$ (lo anterior no ocurre de manera tan natural, pues la elección de la transformación adecuada conlleva un mayor análisis). Se observa lo siguiente en el diagrama de dispersión:



Ahora, ajustando el nuevo modelos $\hat{Y} = \alpha + \beta Z$ el diagrama de dispersión de los residuos no muestra la parábola que claramente se observaba en la gráfica de residuos anterior.



Nótese que en los ejemplos anteriores se ignoró la variable *data* dentro de la función *lm()*, pues se tenían explícitamente las variables X y Y. Para ilustrar el uso de datos externos al ambiente, asumase que se tiene Edad, Sexo e Ingreso dentro del archivo datos.csv:

```
datos <- read.csv("datos.csv")
kable(head(datos), format = "markdown")
```

X	Persona	Edad	Sexo	Ingreso
1	1	21	1	141080
2	2	27	1	205230
3	3	32	1	224710
4	4	37	1	260170
5	5	42	1	273140
6	6	47	1	280930

Para ajustar el modelo $\widehat{\text{Ingreso}} = \hat{\alpha} + \hat{\beta}\text{Edad}$, utilizamos el siguiente código:

```
modelo <- lm(formula = Ingreso ~ Edad, data = datos)
summary(modelo)
```

```
##
## Call:
## lm(formula = Ingreso ~ Edad, data = datos)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -77782 -45015 -20829  44588  75487
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 129343.0    43017.8   3.007  0.00836 **
```

```
## Edad          1626.4      980.2   1.659  0.11654
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 54420 on 16 degrees of freedom
## Multiple R-squared:  0.1468, Adjusted R-squared:  0.09348
## F-statistic: 2.753 on 1 and 16 DF,  p-value: 0.1165
```

Podemos utilizar el siguiente código para quitar el intercepto de la ecuación de la recta:

```
modelo <- lm(Ingreso ~ Edad-1, data = datos)
summary(modelo)
```

```
##
## Call:
## lm(formula = Ingreso ~ Edad - 1, data = datos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -122862  -25461   19193   66159   95901
##
## Coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## Edad   4439.7      354.7   12.52 5.27e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 66040 on 17 degrees of freedom
## Multiple R-squared:  0.9021, Adjusted R-squared:  0.8963
## F-statistic: 156.7 on 1 and 17 DF,  p-value: 5.268e-10
```

Nótese que en el código anterior se utilizó $(Y \sim X - 1)$ y la variable explicativa no cambió a $X' = X - 1$. Al ingresar la fórmula dentro de la función $lm()$ se debe evitar realizar operaciones dentro de ella o transformaciones de las variables. Se prefiere hacerlo fuera de la fórmula en variables auxiliares.

2.2. Regresión lineal múltiple

La estimación para el método de regresión lineal múltiple es similar e igual de sencilla de implementar en R. Ahora tenemos p variables y la ecuación

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \beta_2 X_{2,i} + ... + \beta_p X_{p,i} + \varepsilon_i \tag{5}$$

Para el caso general, se utiliza el siguiente comando en R:[5]

$$\text{modelo <- lm(Y} \sim \text{X1 + X2 + ... + Xp)}$$

Todo lo ya mencionado para el caso de una variable sigue aplicando en este caso.

Para ilustrar la estimación de los parámetros β_i, tomemos $X_1 \in [0, 5]$, $X_2 = X_1^2$ y $Y = 3 + 5X_1 - X_2 + \varepsilon$ donde $\varepsilon \sim N(0, 1)$.

[5]Como ya se mencionó, debe evitarse incluir formulas dentro de la función *lm()*. Es decir, en lugar de utilizar la función $lm(Y \sim X_1^2)$ se debe definir $X_2 = X_1^2$ y correr $lm(Y \sim X_2)$, mas no $lm(Y \sim X^2)$

```r
X1 <- seq(0,5,length.out = 100)
X2 <- X1^2
Y <- 3 + 5*X1 - X2 +  rnorm(100,0,1)
```

Analicemos múltiples intentos de estimar los coeficientes. Primero, tomemos el modelo

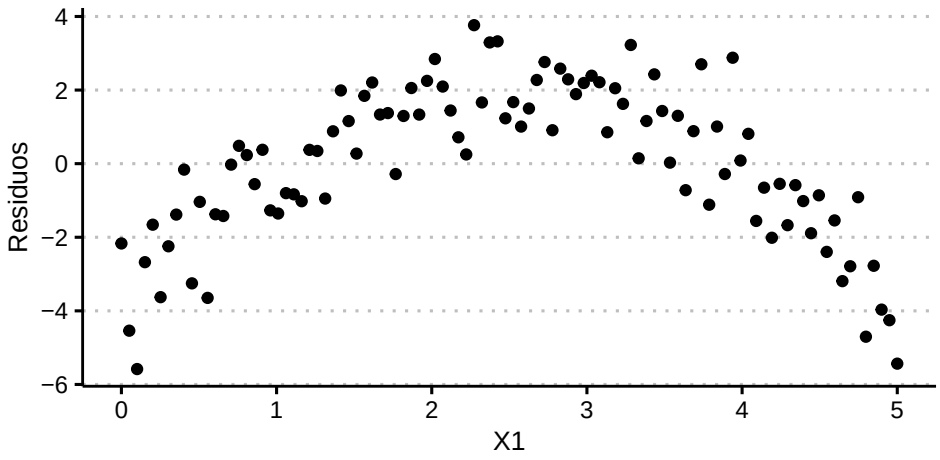
$$Y = \alpha + \beta X_1 + \varepsilon$$

con $\varepsilon \sim N(0, \sigma^2)$ con σ^2 desconocida pero constante.

```r
modelo_1 <- lm(Y ~ X1)
summary(modelo_1)
```

```
##
## Call:
## lm(formula = Y ~ X1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.5814 -1.3599  0.2622  1.6348  3.7638
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.01164    0.42290  16.580   <2e-16 ***
## X1          -0.02352    0.14613  -0.161    0.872
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.13 on 98 degrees of freedom
## Multiple R-squared:  0.0002643,  Adjusted R-squared:  -0.009937
## F-statistic: 0.02591 on 1 and 98 DF,  p-value: 0.8725
```

En este caso, los residuos tienen una clara tendencia.

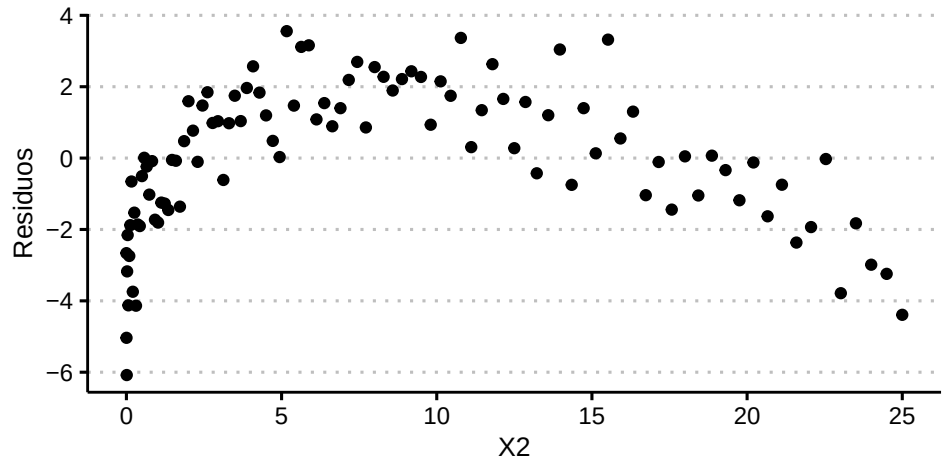


Así, se podría utilizar el modelo univariado $Y = \alpha + \beta X_1^2 + \varepsilon$. Ajustando el modelo, se obtiene lo siguiente:

```r
modelo_2 <- lm(Y ~ X2)
summary(modelo_2)
```

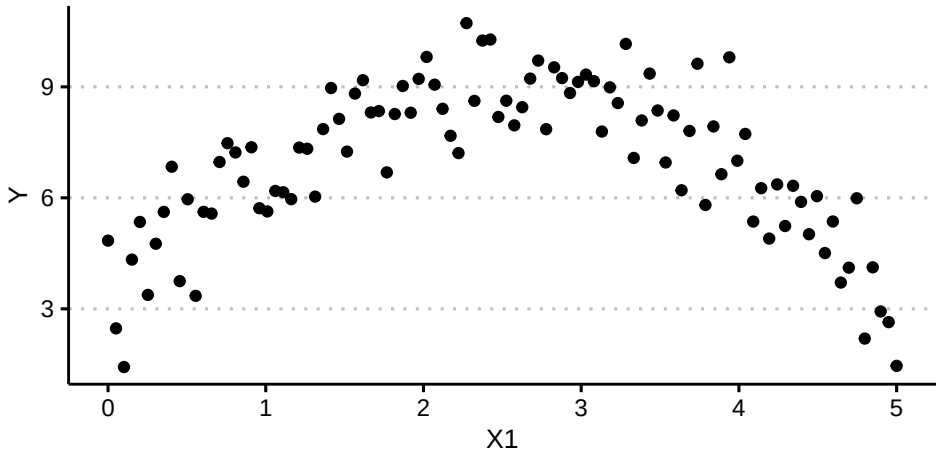
```
##
## Call:
## lm(formula = Y ~ X2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.0793 -1.3815  0.0584  1.5483  3.5564
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.50779    0.30956  24.253   <2e-16 ***
## X2          -0.06626    0.02748  -2.411   0.0178 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.07 on 98 degrees of freedom
## Multiple R-squared:  0.056,  Adjusted R-squared:  0.04637
## F-statistic: 5.814 on 1 and 98 DF,  p-value: 0.01776
```

En este caso, de nuevo se observa una tendencia parabálica en los residuos, pese a haber utilizado la variable $X_2 = X_1^2$.



Ahora bien, si analizamos el comportamiento de Y en función de X_1 (que es lo primero que debe hacerse), notamos que se trata de una parábola que abre hacia abajo, con un máximo entre 2 y 3, y raíces cercanas a 0 y 5. Así, dado que la parábola se encuentra desplazada del origen, se requiere añadir la variable X_1 al modelo anterior. Es decir,

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_1^2 + \varepsilon$$

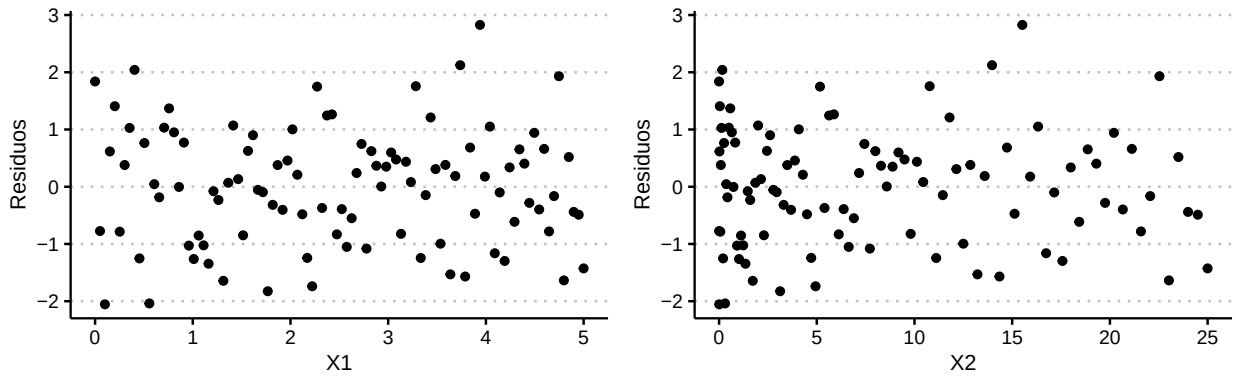


Ajustando el modelo anterior, se obtiene lo siguiente:

```
modelo_3 <- lm(Y ~ X1 + X2)
summary(modelo_3)
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.05484 -0.79498  0.02442  0.65504  2.82728
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.0042     0.3041   9.879 2.45e-16 ***
## X1            4.8344     0.2811  17.198  < 2e-16 ***
## X2           -0.9716     0.0544 -17.859  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.034 on 97 degrees of freedom
## Multiple R-squared:  0.7669, Adjusted R-squared:  0.7621
## F-statistic: 159.5 on 2 and 97 DF,  p-value: < 2.2e-16
```

Nótese que el coeficiente de determinación incrementa de manera importante relativo al llamado **modelo_2**. Además, los residuos en este caso no muestran una tendencia clara, tanto graficando los residuos contra X_1 o contra X_2. Finalmente, los estimadores de α, β_1 y β_2 son muy cercanos a los valores reales 3,5 y -1 respectivamente, y el error estándar de los residuos es muy cercano a 1, que es el valor real de la desviación estándar del error aleatorio.



Cabe recalcar que los ejemplos ilustrados aquí son construcciones. En la vida real es difícil encontrar comportamientos claros, relaciones entre variables, e incluso variables que claramente son explicativas y otras que son explicadas. Como se puede observar, es relativamente sencillo ajustar modelos si se tienen suficientes datos, pero nunca se tiene que perder de vista el modelo teórico que se está intentando probar (si es que el objetivo es primordialmente explicar).

3. Pruebas para el análisis de los supuestos del modelo de regresión lineal

3.1. Heterocedasticidad

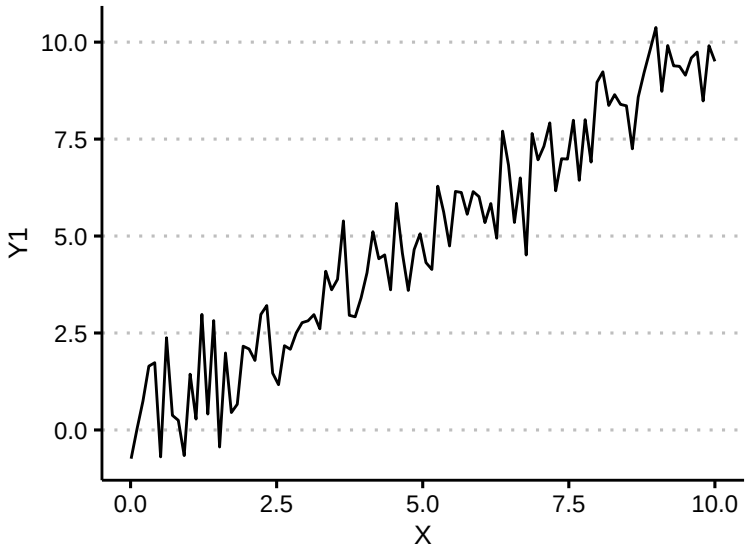
3.1.1. Prueba de Goldfeld-Quandt

El paquete *lmtest* ofrece la función *gqtest*() que realiza la prueba de Goldfeld-Quandt. Para cargar un paquete, primero se debe instalar siguiendo la línea *Packages>Install* y en la ventan que aparece, en el espacio *Packages* escribir el nombre del paquete. Una vez instalado, para cargarlo al ambiente se utiliza la función *library*(paquete).

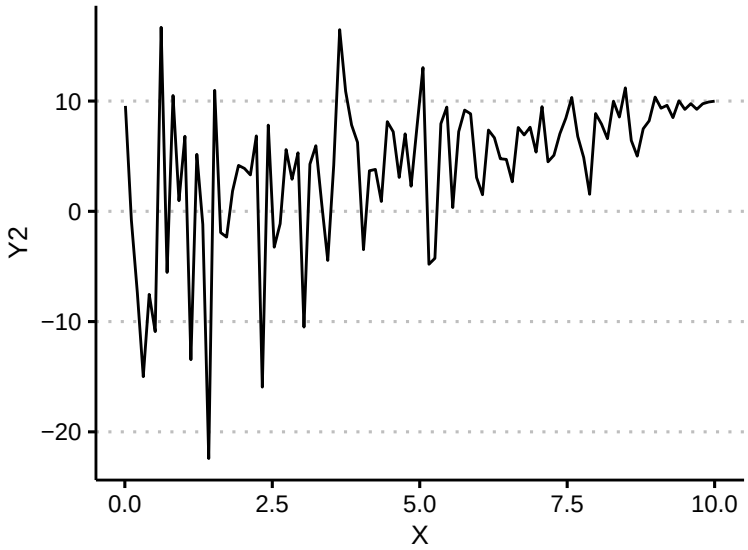
```
library(lmtest)
```

Realicemos la prueba con dos modelos distintos, uno con varianza constante y otro con varianza decreciente.

```
X <- seq(0.01,10,length.out = 100)
Y1 <- X + rnorm(length(X),0,1)
Y2 <- X + rnorm(length(X),0,rev(X))
df <- cbind.data.frame(X,Y1,Y2)

ggplot(df, aes(x = X)) +
  geom_line(aes(y = Y1)) +
  theme_clean() +
  theme(plot.background = element_blank(),
        axis.line = element_line(color = "black"))
```




```
ggplot(df, aes(x = X)) +
  geom_line(aes(y = Y2)) +
  theme_clean() +
  theme(plot.background = element_blank(),
        axis.line = element_line(color = "black"))
```



Realizando la prueba, se obtienen los siguientes resultados:

```
model1 <- lm(Y1 ~ X)
model2 <- lm(Y2 ~ X)

gqtest(model1)
```

```
##
```

```
##  Goldfeld-Quandt test
##
## data:  model1
## GQ = 0.72705, df1 = 48, df2 = 48, p-value = 0.8635
## alternative hypothesis: variance increases from segment 1 to 2
```

```
gqtest(model2)
```

```
##
##  Goldfeld-Quandt test
##
## data:  model2
## GQ = 0.16568, df1 = 48, df2 = 48, p-value = 1
## alternative hypothesis: variance increases from segment 1 to 2
```

Nótese que ambas pruebas concluyen lo mismo, es decir, que no hay incremento en la varianza de un segmento a otro, lo cual sabemos que es falso. Esto se debe a la prueba que se realiza de manera predeterminada es de varianza creciente. Para cambiar esto, tenemos que realizar la siguiente prueba:

```
gqtest(model2, alternative = "less")
```

```
##
##  Goldfeld-Quandt test
##
## data:  model2
## GQ = 0.16568, df1 = 48, df2 = 48, p-value = 2.579e-09
## alternative hypothesis: variance decreases from segment 1 to 2
```

En este caso, es claro que se rechaza la hipótesis nula, así que la varianza decrece. Este ejemplo sirve para ilustrar la necesidad de analizar los datos primero y realizar las pruebas adecuadas, pues se pueden llegar a conclusiones completamente erróneas.

3.1.2. Prueba de Breusch-Pagan

El paquete anterior, *lmtest*, ofrece también la prueba de Breausch-Pagan a través de la función *bptest*. Utilizando el ejemplo anterior, tenemos que

```
bptest(model1)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model1
## BP = 4.1088, df = 1, p-value = 0.04266
```

```
bptest(model2)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model2
## BP = 18.162, df = 1, p-value = 2.029e-05
```

Nótese que el primer modelo no rechaza la hipótesis nula (homoscedasticidad) para niveles de significancia del 5 %, mientras que para el segundo modelo se rechaza la homoscedasticidad.

3.1.3. Prueba de White

No existe una función en R para realizar la prueba de White sobre modelos lineales (existe una función para modelos de vectores autorregresivos), así que se tiene que hacer la prueba a través de la prueba de Breausch-Pagan:

```
bptest(model1, ~ X + I(X^2))
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model1
## BP = 4.9432, df = 2, p-value = 0.08445
```

```
bptest(model2, ~ X + I(X^2))
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model2
## BP = 20.34, df = 2, p-value = 3.831e-05
```

La conclusión resulta la misma: para el primer modelo no se rechaza la hipótesis nula H_0 : existe homoscedasticidad, mientra que para el segundo caso sí se rechaza.

3.2. Normalidad

3.2.1. Prueba de Jarque-Bera

La prueba de Jarque-Bera está disponible en el paquete *tseries* con la función *jarque.bera.test*(). En este caso, el argumento no es el modelo de clase «lm», sino los residuos:

```
library(tseries)
jarque.bera.test(model1$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  model1$residuals
## X-squared = 0.41162, df = 2, p-value = 0.814
```

```
jarque.bera.test(model2$residuals)
```

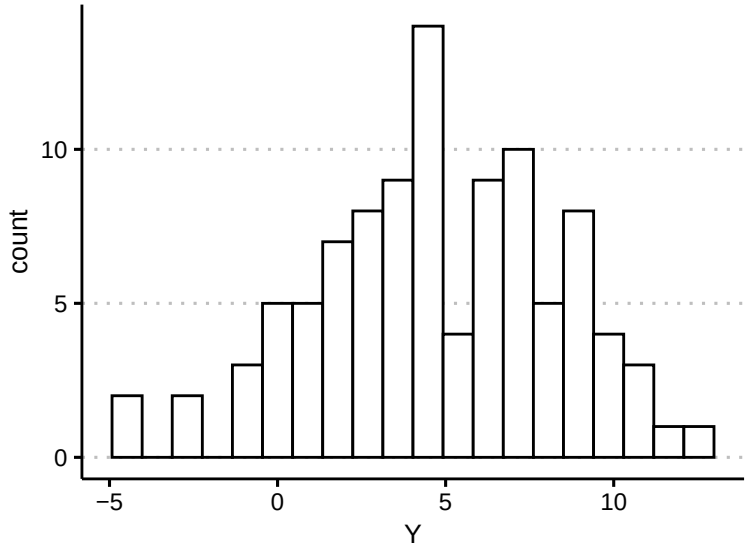
```
##
##  Jarque Bera Test
##
## data:  model2$residuals
## X-squared = 36.455, df = 2, p-value = 1.213e-08
```

Nótese que en el primer no se rechaza la hipótesis nula, mientras que en el segundo sí se rechaza para ciertos niveles de α debido a la presencia de varianza no constante. Para ejemplificar aún más cuando los errores no son normales, analicemos un ejemplo hipótetico. Asumamos que

$$Y = X + \varepsilon$$

donde $\varepsilon \sim t(2)$, es decir, una t de Student.

```
Y <- X + rt(length(X), df = 2)
data <- data.frame(Y)

ggplot(data,aes(x = Y)) +
  geom_histogram(fill = "white", color = "black", bins = 20) +
  theme_clean() +
  theme(plot.background = element_blank(),
        axis.line = element_line(color = "black"))
```



Nótese que el histograma de Y, aunque simÃ©trico, muestra valores extremos. Realizando la prueba de Jarque-Bera, utilizando un modelo de regresión lineal simple, resulta que los errores no son normales, lo cual es de esperarse debido a la presencia de varios valores considerados «extremos» bajo normalidad:

```
model3 <- lm(Y ~ X)
jarque.bera.test(model3$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  model3$residuals
## X-squared = 471.45, df = 2, p-value < 2.2e-16
```

3.2.2. Prueba de Anderson-Darling

Para realizar la prueba de Anderson-Darling se requiere el paquete *nortest*, el cual contiene la función *ad.test*():

```r
library(nortest)
```

Analizando el modelo 1, resulta lo siguiente:

```r
ad.test(model1$residuals)
```

```
##
##  Anderson-Darling normality test
##
## data:  model1$residuals
## A = 0.24663, p-value = 0.7494
```

Por otro lado, para el modelo 3 (errores t de Student) tenemos no-normalidad:

```r
ad.test(model3$residuals)
```

```
##
##  Anderson-Darling normality test
##
## data:  model3$residuals
## A = 4.2207, p-value = 1.453e-10
```

Nótese que, al igual que en el caso de la prueba de Jarque-Bera, el argumento de la función es el vector de residuos, **no el modelo lineal**.

3.3. Autocorrelación

Antes de realizar pruebas de autocorrelación, se tiene que estar seguro que los datos pueden presentar tal problema, es decir, que existe una dimensión en los datos tales que puedan ordenarse (usualmente una dimensión temporal). Si este no es el caso, realizar pruebas de autocorrelación es innecesario.

3.3.1. Prueba de Durbin-Watson

Para ejemplificar, asumamos que

$$Y_t = X_t + e_t$$

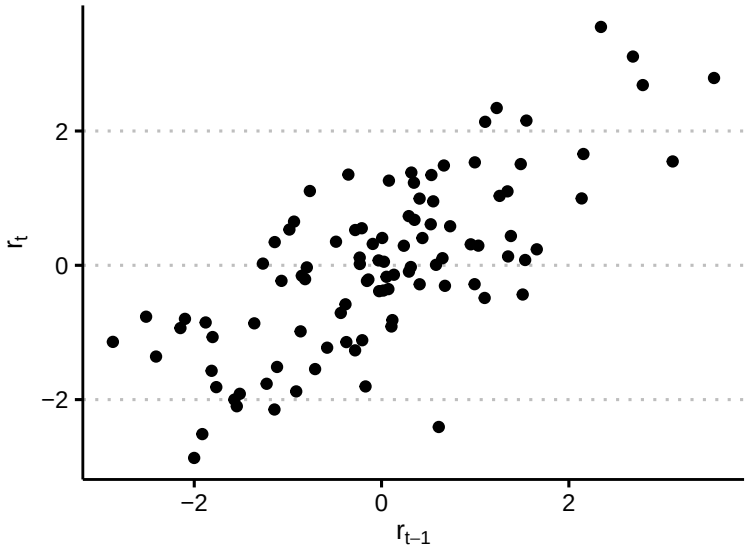
y

$$e_t = 0{,}9e_{t-1} + \nu_t$$

donde ν_t son choques aleatorios.

```r
X_t <- seq(0,1,length.out = 100)
v_t <- rnorm(100)
e_t <- arima.sim(model = list(order = c(1,0,0), ar = 0.9), n = 100)

Y_t <- X_t + e_t
modelAR <- lm(Y_t ~ X_t)
data <- cbind.data.frame(modelAR$residuals[-length(modelAR$residuals)], modelAR$residuals[-1])
colnames(data) <- c("X","Y")
ggplot(data, aes(X,Y)) +
  geom_point() +
```

```
  theme_clean() +
  theme(plot.background = element_blank(),
        axis.line = element_line(color = "black")) +
  xlab(expression("r"[t-1])) +
  ylab(expression("r"[t]))
```



Nótese que los errores estan correlacionados con un retrazo de 1, así que se tiene un problema de autocorrelación. Realizando la prueba de Durbin-Watson obtenemos lo siguiente:

```
dwtest(modelAR)
```

```
##
##  Durbin-Watson test
##
## data:  modelAR
## DW = 0.54317, p-value = 4.217e-14
## alternative hypothesis: true autocorrelation is greater than 0
```

Así, se rechaza la hipótesis nula que los errores no están autocorrelacionados.

3.3.2. Prueba de Breusch-Godfrey

Esta prueba permite analizar estructuras de autocorrelación con retrazos mayores o iguales a uno. Al igual que en el caso anterior, supongamos que

$$Y_t = X_t + e_t$$

pero que

$$e_t = 0{,}9\, e_{t-4} + \nu_t$$

con ν_t un error aleatorio. Para realizar la prueba, se puede utilizar la función *bgtest*() del paquete *lmtest*:

```r
e_t <- arima.sim(model = list(order = c(4,0,0), ar = c(0,0,0,0.9)), n = 100)
Y_t <- X_t + e_t
modelAR2 <- lm(Y_t ~ X_t)
```

Realizando la prueba con *order* = 1 no se rechaza la hipótesis nula de que de que $\rho_i = 0$, pero para *order* = 2,3,4 sí se rechaza la hipótesis nula:

```r
bgtest(modelAR2, order = 1)
```

```
##
##  Breusch-Godfrey test for serial correlation of order up to 1
##
## data:  modelAR2
## LM test = 0.039509, df = 1, p-value = 0.8424
```

```r
bgtest(modelAR2, order = 2)
```

```
##
##  Breusch-Godfrey test for serial correlation of order up to 2
##
## data:  modelAR2
## LM test = 16.822, df = 2, p-value = 0.0002225
```

```r
bgtest(modelAR2, order = 3)
```

```
##
##  Breusch-Godfrey test for serial correlation of order up to 3
##
## data:  modelAR2
## LM test = 16.863, df = 3, p-value = 0.0007543
```

```r
bgtest(modelAR2, order = 4)
```

```
##
##  Breusch-Godfrey test for serial correlation of order up to 4
##
## data:  modelAR2
## LM test = 80.754, df = 4, p-value < 2.2e-16
```

3.3.3. Colinealidad

3.4. Factor de inflación de la varianza (FIV)

Para calcular el factor de inflación de la varianza existe la función *ols_vif_tol*() del paquete *olsrr*. Para ejemplificar, se analizará una base de datos con información sobre densidad de huesos y distintas variables de 92 personas.

```r
datos <- read.csv("MulticollinearityExample.csv")
head(datos)
```

```
##   Femoral.Neck X.Fat Weight.kg Activity    X.Fat.S  Weight.S Activity.S
## 1        0.934  25.3  52.16313  3508.44 -3.2652174 -1.765066   946.4504
## 2        0.888  29.3  61.80196  2773.54  0.7347826  7.873772   211.5504
## 3        0.933  37.7  93.44003  1738.97  9.1347826 39.511842  -823.0196
## 4        0.757  32.8  59.87420  1665.29  4.2347826  5.946005  -896.6996
## 5        1.031  24.6  50.34876  3982.95 -3.9652174 -3.579436  1420.9604
## 6        0.883  26.5  57.60623  2985.74 -2.0652174  3.678043   423.7504
```

```r
modelo <- lm(data = datos, Femoral.Neck ~ Activity + X.Fat + Weight.kg + I(X.Fat*Weight.kg))
```

Ahora, calculamos los factores de inflación de la varianza del modelo anterior:

```r
library(olsrr)
ols_vif_tol(modelo)
```

```
##              Variables  Tolerance       VIF
## 1             Activity 0.94966333  1.053005
## 2                X.Fat 0.06697226 14.931555
## 3            Weight.kg 0.02945649 33.948375
## 4 I(X.Fat * Weight.kg) 0.01332281 75.059251
```

Existen otros paquetes que contienen funciones para analizar colinealidad, como *mctest* que se veran a continuación.

3.4.1. Número de condición e índice de condición

Para realizar este análisis se utilizará la función *omcdiag*() del paquete *mctest*. Utilizando el mismo ejemplo anterior, se obtiene lo siguiente:

```r
library(mctest)
omcdiag(x = cbind.data.frame(datos[,c(2,3,4)], datos$X.Fat*datos$Weight.kg), y = datos[,1])
```

```
##
## Call:
## omcdiag(x = cbind.data.frame(datos[, c(2, 3, 4)], datos$X.Fat *
##     datos$Weight.kg), y = datos[, 1])
##
##
## Overall Multicollinearity Diagnostics
##
##                        MC Results detection
## Determinant |X'X|:         0.0041         1
## Farrar Chi-Square:       488.0914         1
## Red Indicator:             0.6452         1
## Sum of Lambda Inverse:   124.9922         1
## Theil's Method:            1.2536         1
## Condition Number:         78.0125         1
##
## 1 --> COLLINEARITY is detected by the test
## 0 --> COLLINEARITY is not detected by the test
```

El comando anterior arroja distintas pruebas de colinealidad, y si la prueba detecta o no colinealidad. Asimismo, existe la función *eigprop*() para calcular el índice de condición:

```
imcdiag(x = cbind.data.frame(datos[,c(2,3,4)], datos$X.Fat*datos$Weight.kg), y = datos[,1])
```

```
##
## Call:
## imcdiag(x = cbind.data.frame(datos[, c(2, 3, 4)], datos$X.Fat *
##     datos$Weight.kg), y = datos[, 1])
##
##
## All Individual Multicollinearity Diagnostics Result
##
##                                     VIF    TOL        Wi        Fi Leamer
## X.Fat                           14.9316 0.0670  408.6589  619.9542 0.2588
## Weight.kg                       33.9484 0.0295  966.4857 1466.2027 0.1716
## Activity                         1.0530 0.9497    1.5548    2.3587 0.9745
## datos$X.Fat * datos$Weight.kg 75.0593 0.0133 2172.4047 3295.6367 0.1154
##                                     CVIF Klein   IND1   IND2
## X.Fat                           1072.5266     1 0.0023 1.2692
## Weight.kg                       2438.4960     1 0.0010 1.3202
## Activity                          75.6368     0 0.0324 0.0685
## datos$X.Fat * datos$Weight.kg 5391.4711     1 0.0005 1.3422
##
## 1 --> COLLINEARITY is detected by the test
## 0 --> COLLINEARITY is not detected by the test
##
## X.Fat , coefficient(s) are non-significant may be due to multicollinearity
##
## R-square of y on all x: 0.5623
##
## * use method argument to check which regressors may be the reason of collinearity
## ===================================
```

```
eigprop(x = cbind.data.frame(datos[,c(2,3,4)], datos$X.Fat*datos$Weight.kg))
```

```
##
## Call:
## eigprop(x = cbind.data.frame(datos[, c(2, 3, 4)], datos$X.Fat *
##     datos$Weight.kg))
##
##   Eigenvalues      CI Intercept  X.Fat Weight.kg Activity
## 1      4.7035  1.0000    0.0001 0.0002    0.0001   0.0052
## 2      0.2267  4.5545    0.0008 0.0003    0.0005   0.3119
## 3      0.0584  8.9762    0.0261 0.0013    0.0003   0.6548
## 4      0.0106 21.0749    0.0031 0.2090    0.0795   0.0146
## 5      0.0008 78.0125    0.9699 0.7893    0.9196   0.0135
##   datos$X.Fat * datos$Weight.kg
## 1                        0.0001
## 2                        0.0045
## 3                        0.0098
## 4                        0.0000
## 5                        0.9856
```

```
##
## ==============================
## Row 5==> X.Fat, proportion 0.789311 >= 0.50
## Row 5==> Weight.kg, proportion 0.919599 >= 0.50
## Row 3==> Activity, proportion 0.654810 >= 0.50
## Row 5==> datos$X.Fat * datos$Weight.kg, proportion 0.985563 >= 0.50
```

Finalmente, la función *imcdiag*() nos permite analizar los FIV al igual que en el apartado anterior:

```
imcdiag(x = cbind.data.frame(datos[,c(2,3,4)], datos$X.Fat*datos$Weight.kg), y = datos[,1])
```

```
##
## Call:
## imcdiag(x = cbind.data.frame(datos[, c(2, 3, 4)], datos$X.Fat *
##     datos$Weight.kg), y = datos[, 1])
##
##
## All Individual Multicollinearity Diagnostics Result
##
##                                     VIF    TOL        Wi        Fi Leamer
## X.Fat                           14.9316 0.0670  408.6589  619.9542 0.2588
## Weight.kg                       33.9484 0.0295  966.4857 1466.2027 0.1716
## Activity                         1.0530 0.9497    1.5548    2.3587 0.9745
## datos$X.Fat * datos$Weight.kg 75.0593 0.0133 2172.4047 3295.6367 0.1154
##                                     CVIF Klein   IND1   IND2
## X.Fat                           1072.5266     1 0.0023 1.2692
## Weight.kg                       2438.4960     1 0.0010 1.3202
## Activity                          75.6368     0 0.0324 0.0685
## datos$X.Fat * datos$Weight.kg 5391.4711     1 0.0005 1.3422
##
## 1 --> COLLINEARITY is detected by the test
## 0 --> COLLINEARITY is not detected by the test
##
## X.Fat , coefficient(s) are non-significant may be due to multicollinearity
##
## R-square of y on all x: 0.5623
##
## * use method argument to check which regressors may be the reason of collinearity
## ==================================
```

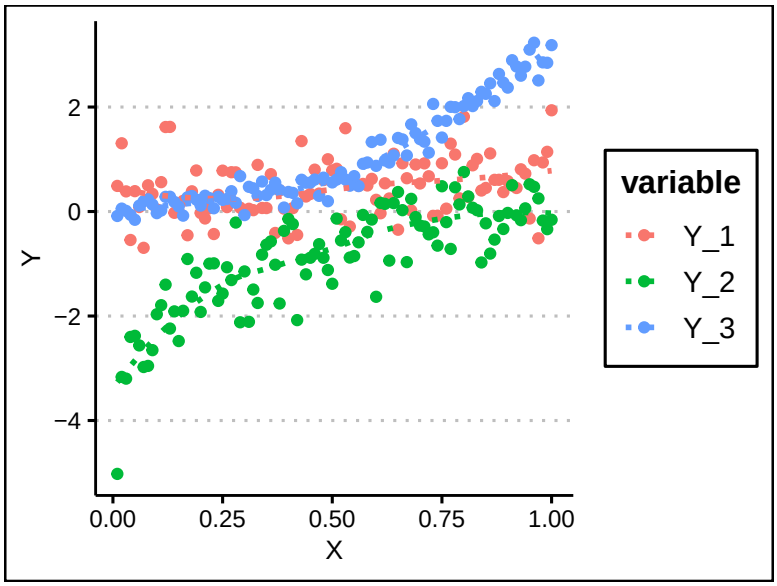
En las funciones se utilizan de argumentos x la matriz de diseño (variables explicativas) y y el vector de respuesta (variable explicada).

3.5. Error de especificación: Cambio estructural

3.5.1. Prueba de Chow

La prueba de Chow para analizar cambios estructurales se puede realizar con la función *chow.test*() del paquete *gap*. Para ejemplificar, se utilizarán las variables $Y_1 = X + \varepsilon_1$, $Y_2 = log(X) + \varepsilon_2$ y $Y_3 = X \times \mathbf{1}_{(0,1/2)}(X) + (5 \times X - 2) \times \mathbf{1}_{(1/2,1)}(X) + \varepsilon_3$, donde $\varepsilon_i \sim N(0, \sigma_i^2)$. En el primer caso, al ser una relación lineal, la prueba de Chow no debería indicar un cambio estructural, en el segundo caso tampoco debería haber diferencia en la regresión para $X \in (0, 1/2)$ y $X \in (1/2, 1)$, mientras que para el último caso sí la hay. En la gráfica se pueden observar estas tres variables.

```r
X <- seq(0.01,1,length.out = 100)
Y_1 <- X + rnorm(100, sd = 0.5)
Y_2 <- log(X) + rnorm(length(X), sd = 0.5)
Y_3 <- as.numeric(100)
Y_3[1:50] <- X[1:50]
Y_3[51:100] <- 5*X[51:100]+(1-5)*0.5
Y_3 <- Y_3 + rnorm(100, sd = 0.2)

data <- cbind.data.frame(X,Y_1,Y_2, Y_3)
data <- melt(data, id.vars = "X")

ggplot(data, aes(x = X, y = value, color = variable)) +
  geom_point() +
  theme_clean() +
  geom_smooth(se = FALSE, linetype = "dotted") +
  ylab("Y")
```



Para la variable Y_1 es claro el no-rechazo:

```r
library(gap)
chow.test(Y_1[1:50], X[1:50],Y_1[51:100], X[51:100])
```

```
##    F value      d.f.1      d.f.2    P value
##  0.3292465  2.0000000 96.0000000  0.7202748
```

Para la variable Y_2 se observa que la hipótesis nula de que las regresiones son equivalentes se rechaza, probablemente debido al decaimiento de la función logaritmo. Se concluye así que la prueba no es definitiva y es necesario realizar un análisis gráfico previo a la realización de pruebas.

```r
chow.test(Y_2[1:50], X[1:50],Y_2[51:100], X[51:100])
```

```
##      F value        d.f.1        d.f.2      P value
## 1.038355e+01 2.000000e+00 9.600000e+01 8.272489e-05
```

Finalmente, para la variable Y_3 también se rechaza la hipótesis nula:

```
chow.test(Y_3[1:50], X[1:50],Y_3[51:100], X[51:100])
```

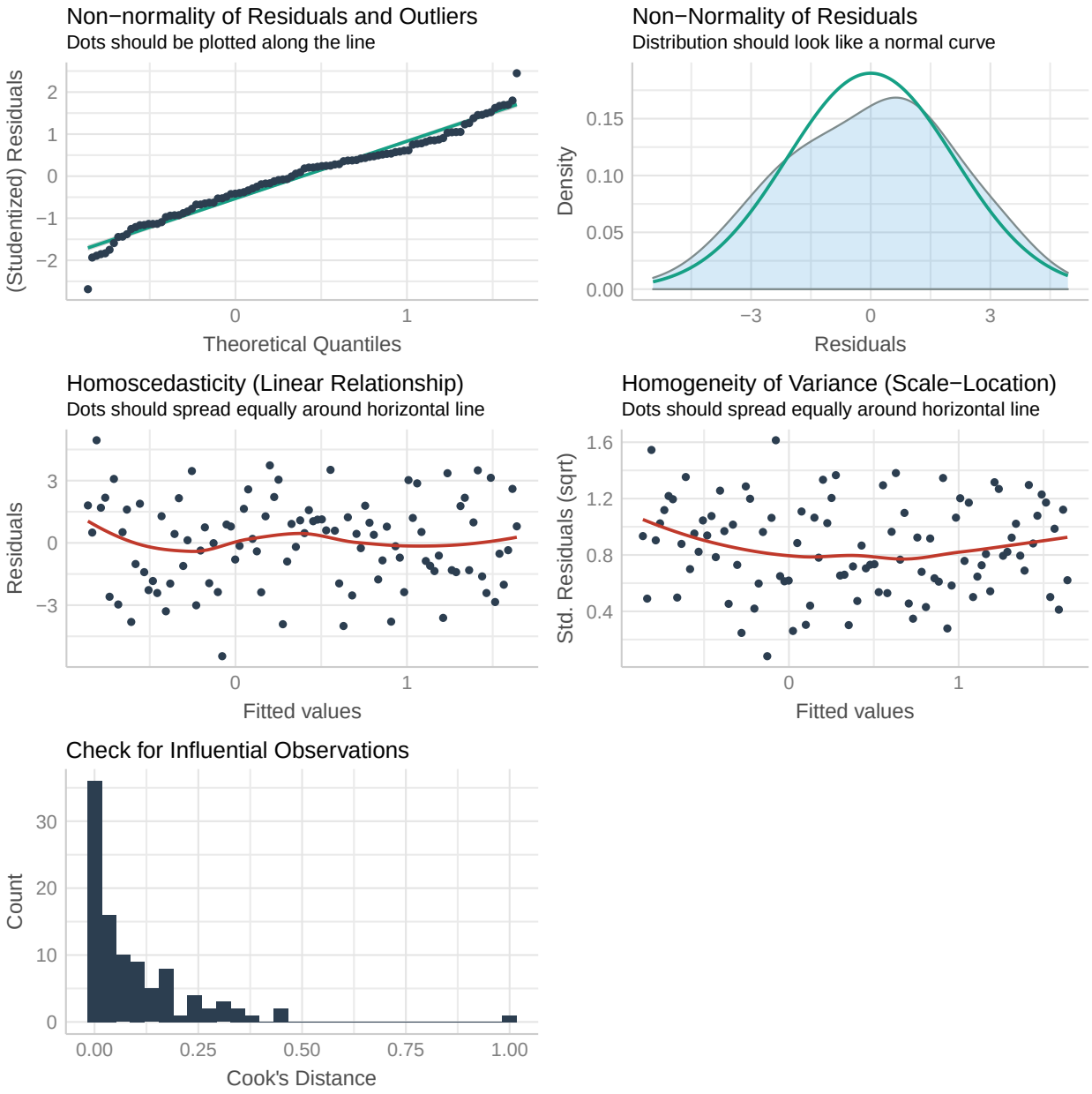
```
##      F value        d.f.1        d.f.2      P value
## 1.344136e+02 2.000000e+00 9.600000e+01 1.475152e-28
```

3.6. Forma rápida de analizar un modelo

La función *check_model*() de la librería *performance* permite un análisis rápido y gráfico de un objeto tipo *lm*. Para ejemplificar, sea $Y = X + \varepsilon$ con $\varepsilon \sim N(0, \sigma^2)$ y ajustamos un modelo lineal simple. En la gráfica podemos ver que los resiudos son aproximadamente normales, con la presencia de datos atípicos. Asimismo, la media de los errores parece 0 y tienen varianza constante.

```
X <- seq(0,1,length.out = 100)
Y <- X + rnorm(100,0,2)
modelo <- lm(Y ~ X)
performance::check_model(modelo)
```

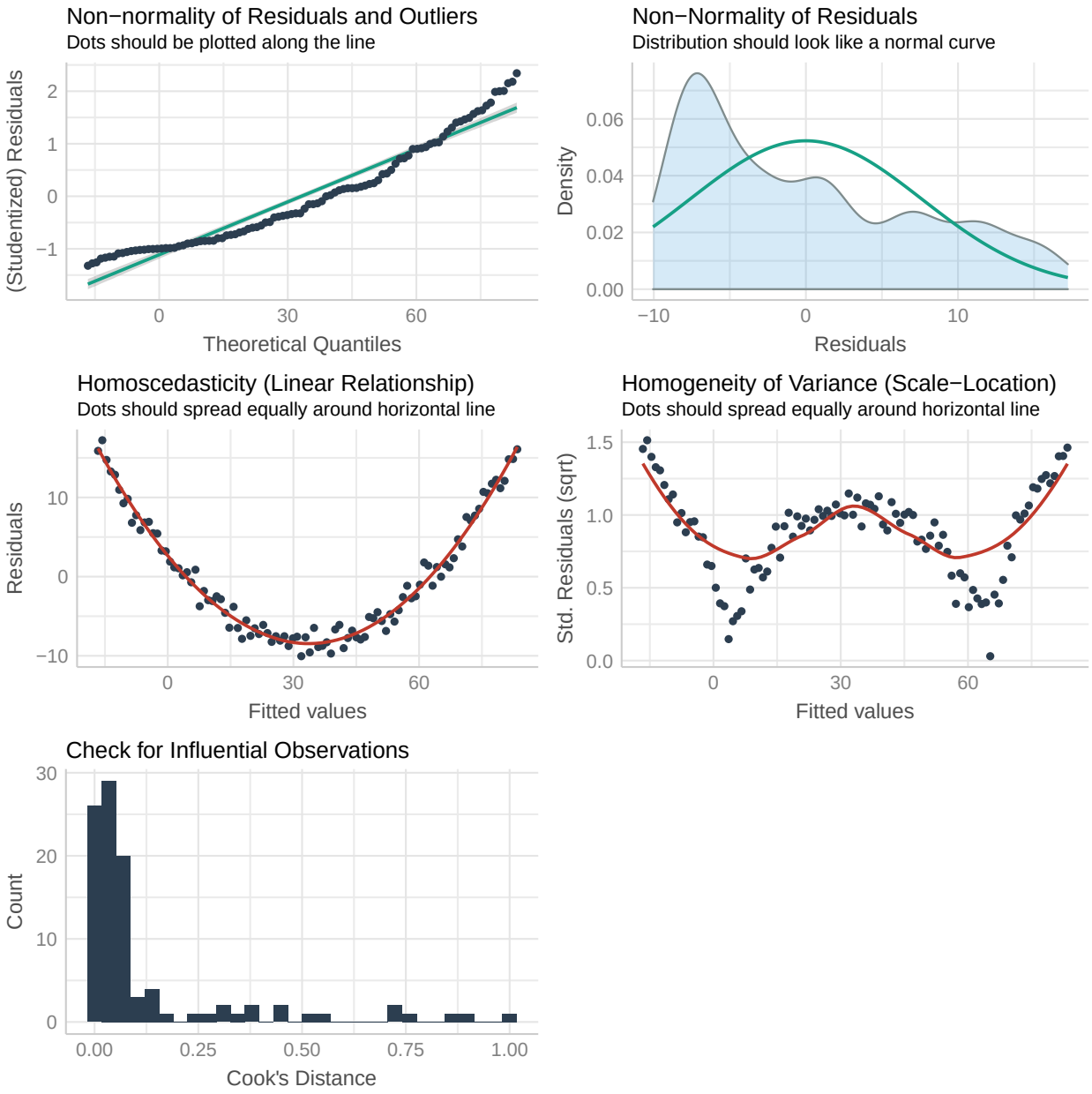
```
## Not enough model terms in the conditional part of the model to check for multicollinearity.
```



Ahora pensemos en el modelo $Y = X^2 + \varepsilon$. Si se inicia con un ajuste de la forma $Y = \beta_0 + \beta_1 X + \varepsilon$, la función nos muestra claramente que los residuos presentan múltiples problemas

```
X <- seq(0,10,length.out = 100)
Y <- X^2 + rnorm(100,0,1)
modelo <- lm(Y ~ X)
performance::check_model(modelo)
```

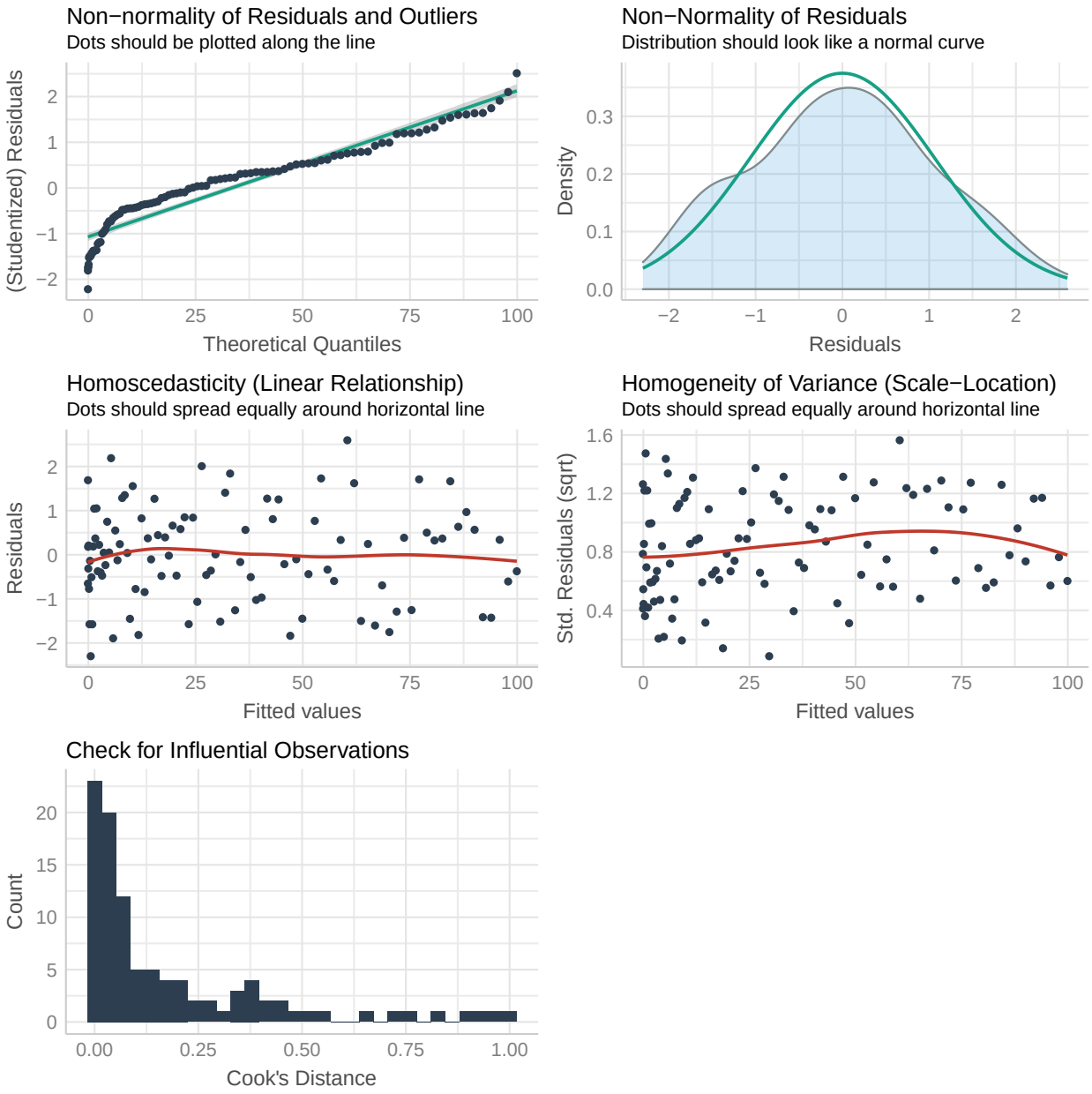
```
## Not enough model terms in the conditional part of the model to check for multicollinearity.
```



En cambio, si se ajusta $Y = \alpha_0 + \alpha_1 X^2 + \varepsilon$, se obtienen resultados más sensatos.

```
X_2 <- X^2
modelo <- lm(Y ~ X_2)
performance::check_model(modelo, bins = 40)
```

```
## Not enough model terms in the conditional part of the model to check for multicollinearity.
```



4. Graficar con ggplot

La librería *ggplot* permite realizar gráficos en $\mathbb{R}^{\not\varepsilon}$ con gran capacidad de edición. El motivo de esta sección no es dar un curso obre cómo utilizar *ggplot* en cualquier ámbito, pues es una librería muy extensa. En su lugar, se darán instrucciones y ejemplos de cómo realizar ciertas clases de gráficas útiles para el análisis econométrico.

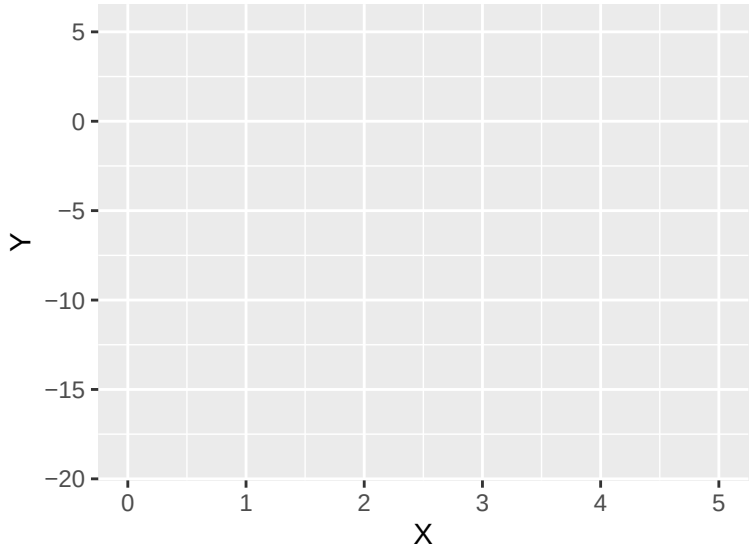
4.1. Gráficos básicos: dispersión y linea

Supóngase que se tienen 5 variables, Y la variable de respuesta y X^i para $i = 1, 2, 3, 4$ variables explicativas.

```
X <- seq(0,5,length.out = 500)
X_2 <- X^2
X_3 <- X^3
X_4 <- X^4
Y <- 2 + X - X^2 + rnorm(length(X))
datos <- data.frame(Y, X, X_2, X_3, X_4)
```

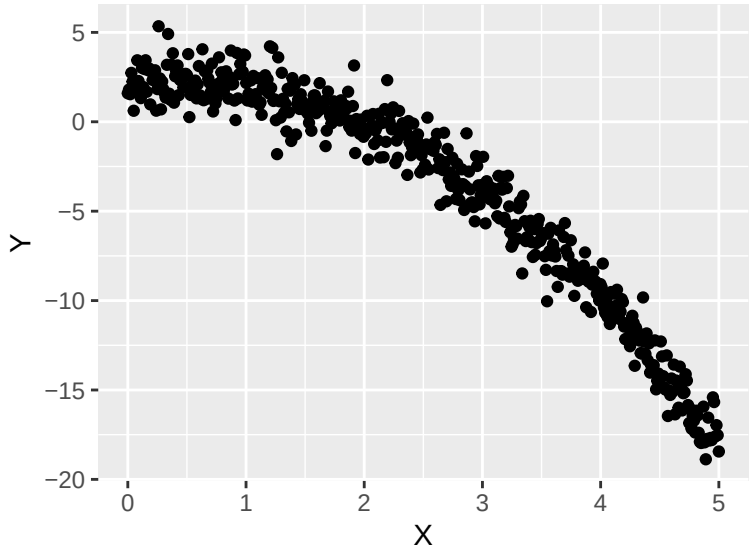
Para utilizar *ggplot*, es necesario que los datos se encuentren en un *data frame*. Para inicializar el plot, basta con utilizar el siguiente comando:

```
ggplot(data = datos, aes(x = X, y = Y))
```



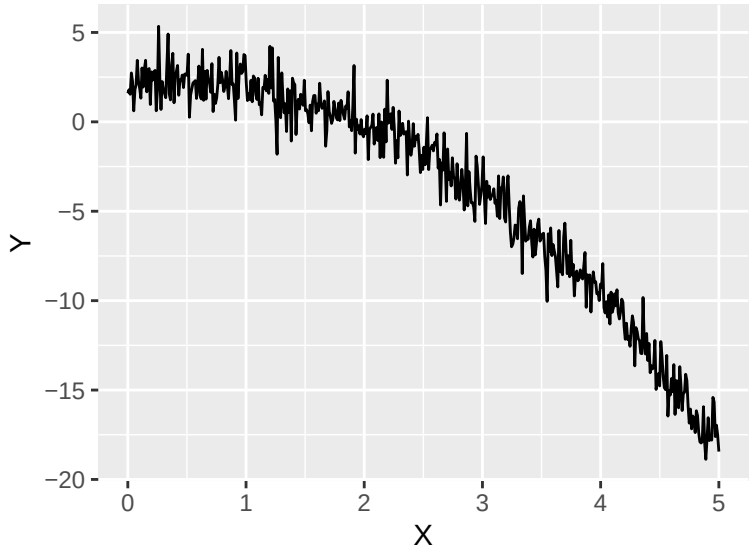
Este comando será similar para distintos tipos de gráficos de 2 variables. La variable *aes*() indica a *ggplot* cuáles son las variables estéticas del gráfico. Al correr lo anterior se crea un plot vacío, en el cual vamos a "dibujar´´ los plots que queramos. El primero, y uno de los más utilizados, es el diagrama de dispersión con *geom_point()*. Para añadir capas basta con utilizar el símbolo +.

```
ggplot(data = datos, aes(x = X, y = Y)) + geom_point()
```



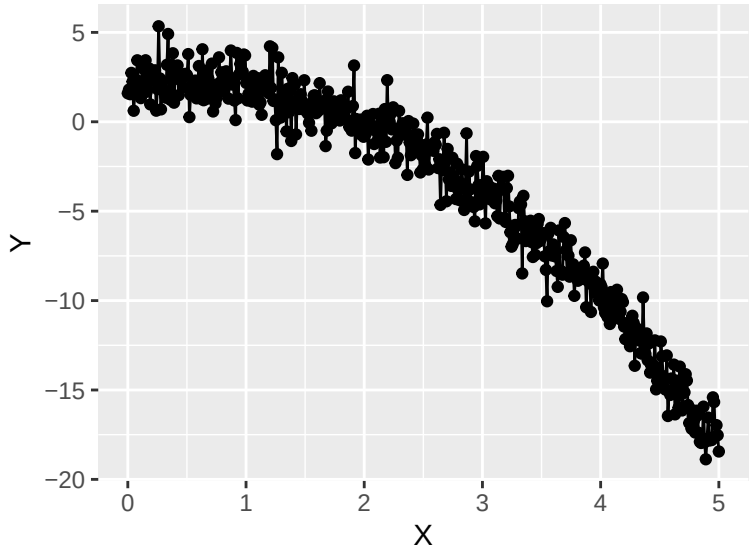
Ahora, si se desea realizar un gráfico de línea (que una cada uno de los puntos) se utiliza *geom_line*:

```
ggplot(data = datos, aes(x = X, y = Y)) + geom_line()
```

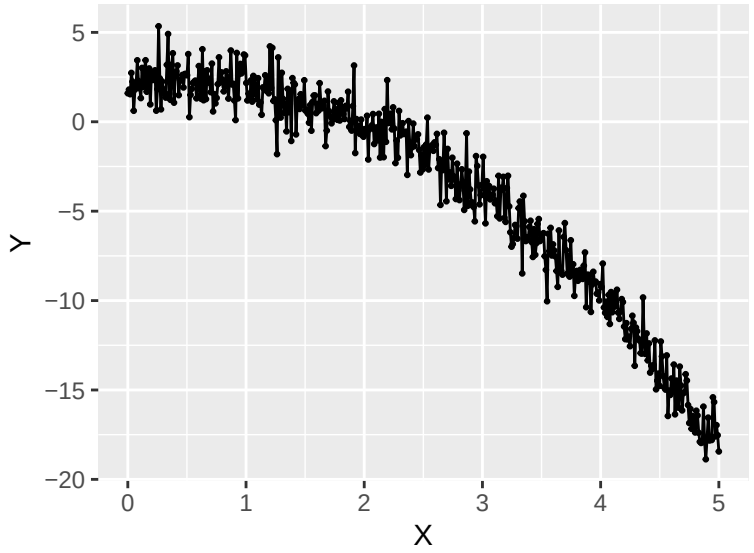


Finalmente, se puede realizar un gráfico donde se incluyan ambas capas, líneas y puntos. Notamos que en este último gráfico se satura. Esto se puede arreglar utilizando el argumento *size* dentro de alguna de las capas para cambiar el tamaño de la capa.

```
ggplot(data = datos, aes(x = X, y = Y)) + geom_point() + geom_line()
```

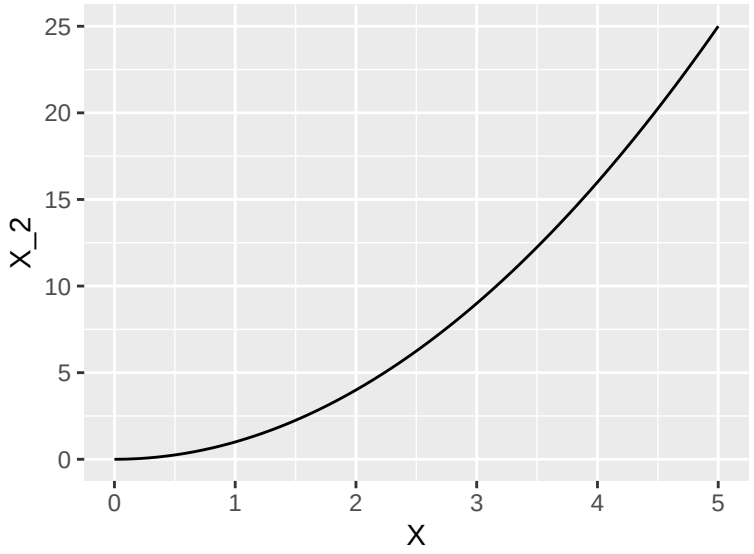



```
ggplot(data = datos, aes(x = X, y = Y)) + geom_point(size = 0.5) + geom_line()
```

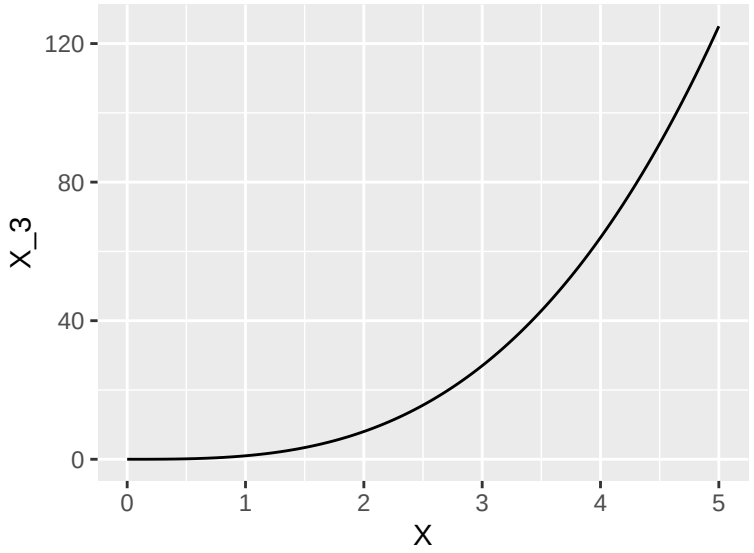


Finalmente, es posible utilizar dentro de *aes*() cualquier par de variables dentro de *data* como x y y. Por ejemplo, en este caso podemos graficar X^i, pero (por ahora) no en el mismo gráfico:

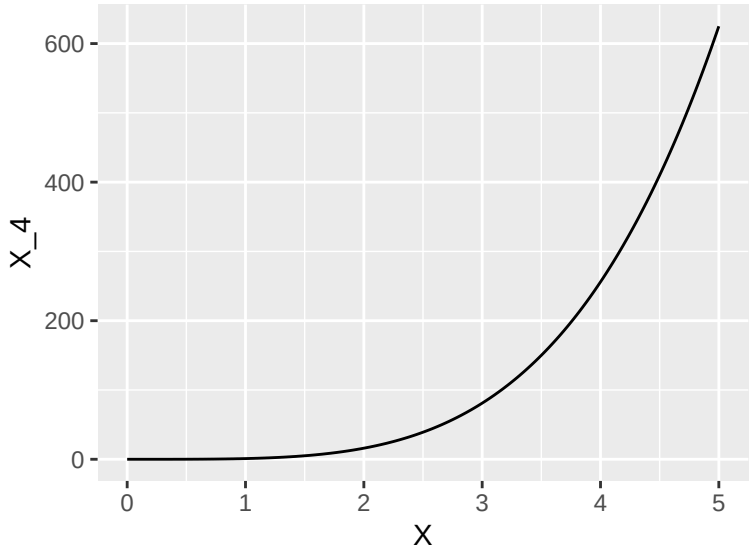
```
ggplot(data = datos, aes(x = X, y = X_2)) + geom_line()
```




```
ggplot(data = datos, aes(x = X, y = X_3)) + geom_line()
```




```
ggplot(data = datos, aes(x = X, y = X_4)) + geom_line()
```



4.2. Múltiples variables en una misma gráfica

Graficar en múltiples dimensiones con *ggplot* es imposible[6], pero sí se pueden graficar múltiples variables en un mismo gráfico. Piénsese, por ejemplo, en $f_i : \mathbb{R} \to \mathbb{R}$ para $i = 1, 2, ..., n$ con $f_i = f_i(x)$. En un mismo gráfico se puede graficar $(x, f_i(x))$ para cada i. A manera de ejemplo tomemos $x \in (0, 1)$, $f_1(x) = x, f_2(x) = log(x)$ y $f_3(x) = x^2$. Primero creamos un *data frame* X como secuencia de 0 a 1 de tamaño 100.

```
library(dplyr)

##Creacion de la base
X <- seq(0,1,length.out = 100)
X <- data.frame(X)
```

Luego, utilizando la librería *dplyr* y la función *mutate*() se crean las variables f_1, f_2 y f_3[7].

```
##Creacion de variables adicionales con mutate()
X <- X %>% mutate(f_1 = X, f_2 = log(X), f_3 = X^2)
head(X)
```

```
##            X        f_1       f_2          f_3
## 1 0.00000000 0.00000000      -Inf 0.0000000000
## 2 0.01010101 0.01010101 -4.595120 0.0001020304
## 3 0.02020202 0.02020202 -3.901973 0.0004081216
## 4 0.03030303 0.03030303 -3.496508 0.0009182736
## 5 0.04040404 0.04040404 -3.208825 0.0016324865
## 6 0.05050505 0.05050505 -2.985682 0.0025507601
```

Finalmente, se utiliza la función *melt*() de la librería *reshape2* para mezclar la base de datos utilizando una variable como identificador:

[6] Existen otras funciones como *plot_ly* y *presp* para graficar en 3 dimensiones.

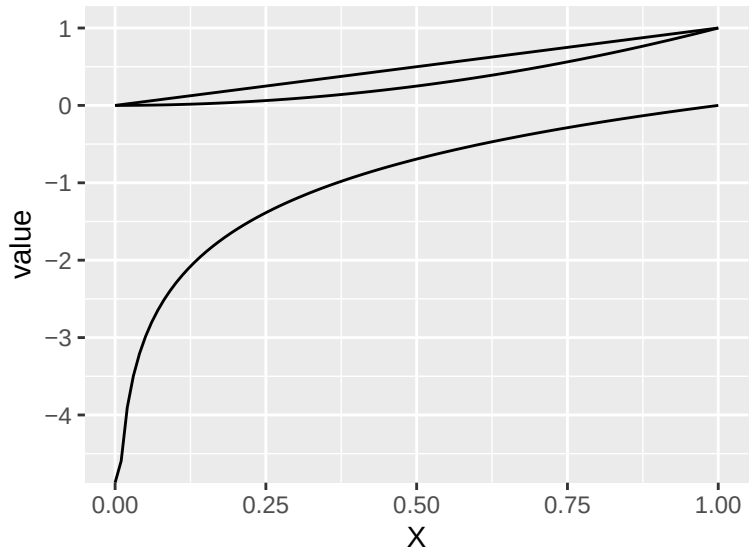
[7] Esta librería tiene una gran capacidad de manipulación de datos, pero su uso va más allá de estas notas. Basta con utilizarla como en el ejemplo.

```
##Mezcla de la base de datos utilizando X como identificador
X_mezcl <- melt(X, id.vars = "X")
head(X_mezcl)
```

```
##            X variable      value
## 1 0.00000000      f_1 0.00000000
## 2 0.01010101      f_1 0.01010101
## 3 0.02020202      f_1 0.02020202
## 4 0.03030303      f_1 0.03030303
## 5 0.04040404      f_1 0.04040404
## 6 0.05050505      f_1 0.05050505
```

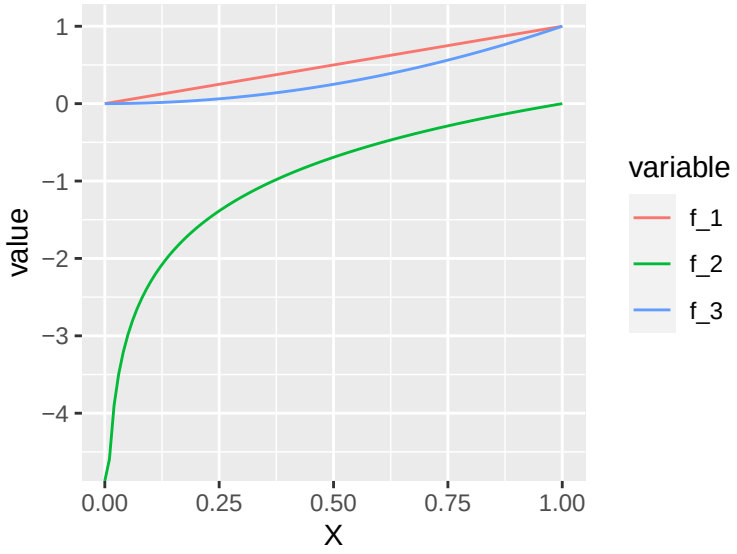
Notamos que se creó una nueva base de datos con tres columnas: la variable identificadora, el nombre de las variables, y los valores para cada identificador, para cada variable. Dada esta nueva base de datos, es más sencillo realizar gráficos con múltiples valores utilizando *group* para agrupar los valores de y con base en *group*:

```
ggplot(X_mezcl, aes(x = X, y = value, group = variable)) +
  geom_line()
```



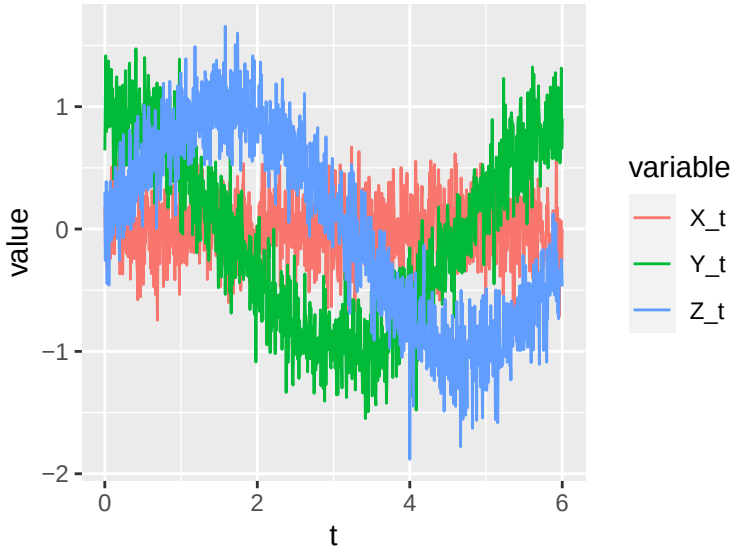
Para distinguir entre las distintas funciones, basta con utilizar el *aesthetic color* y las variables:

```
ggplot(X_mezcl, aes(x = X, y = value, group = variable, color = variable)) +
  geom_line()
```



Esta metodología es especialmente Ãºtil para analizar variables indexadas por el tiempo. Pensemos, por ejemplo, que $t \in (0, 2\pi)$, $X_t \quad i.i.d. \quad N(0, \sigma^2)$, $Y_t = cos(t) + \varepsilon_t$ donde $\varepsilon_t \quad i.i.d. \quad N(0, \sigma^2)$ y $Z_t = sin(t) + \eta_t$ donde $\eta_t \quad i.i.d. \quad N(0, \sigma^2)$, con X_t, ε_t y η_t independientes.

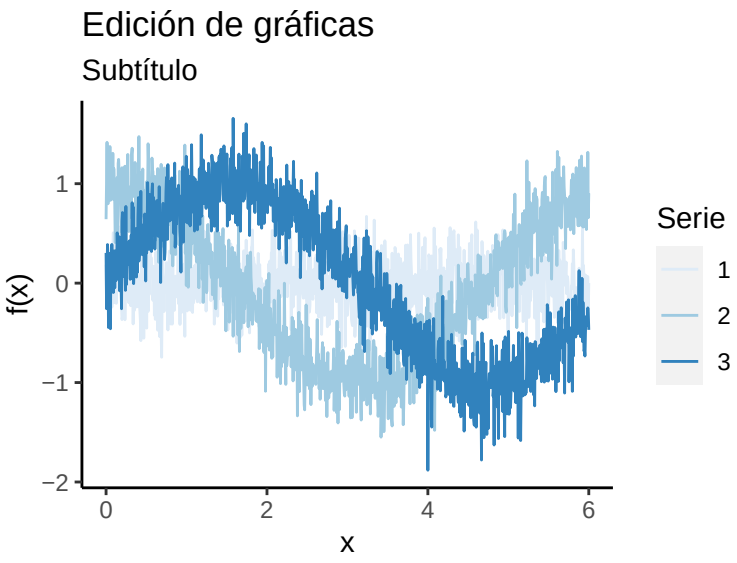
```
sigma <- 0.25
t <- seq(0,2*pi,length.out = 1000)
n <- length(t)
X_t <- rnorm(n, 0, sigma)
Y_t <- cos(t) + rnorm(n, 0, sigma)
Z_t <- sin(t) + rnorm(n, 0, sigma)

datos <- data.frame(t, X_t, Y_t, Z_t)

datos_melt <- melt(datos, id.vars = "t")

ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line()
```



4.3. Edición de gráficas

Así como se pueden añadir elementos al plot, también se pueden editar sus distintos elementos como el fondo, los ejes, leyenda, etc. Algunas de las funciones más importantes son las siguientes:

```
ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  xlab("x") +
  ylab("f(x)") +
  labs(color = "Serie", title = "Edición de gráficas", subtitle = "Subtítulo") +
  theme(plot.background = element_blank(),
        axis.line = element_line(color = "black"),
        panel.background = element_rect(fill = "white")) +
  scale_color_brewer(palette = "Blues", labels = c(1,2,3))
```

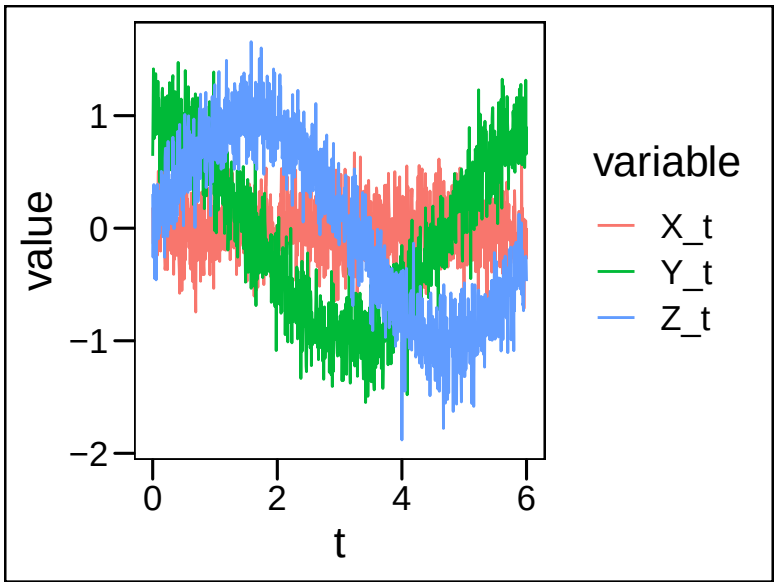


Las distintas adiciones realizan lo siguiente:

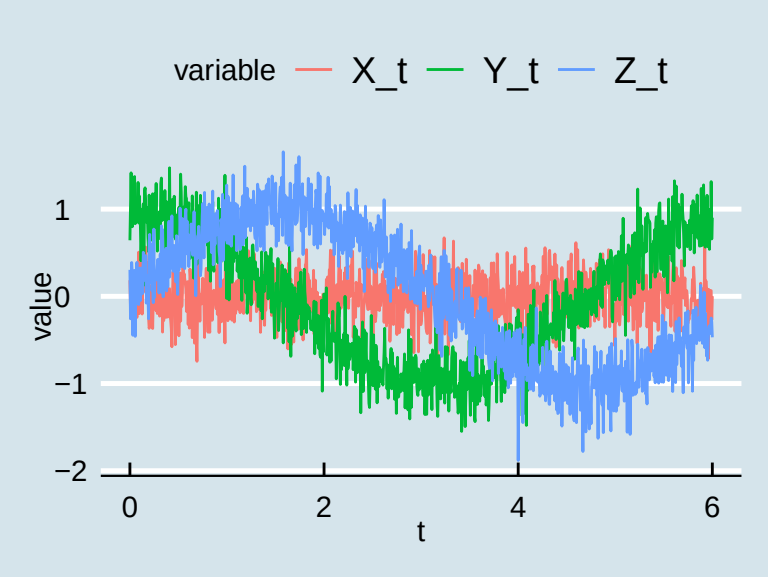
- *xlab* y *ylab* cabian los nombres en los ejes
- *labs* cambia el título, subtítulo y el título de la leyenda creada a través de los colores
- *theme* cambia distintos aspectos de la gráfica: plot cambia los elementos del gráfico entero, axis los de los ejes y panel los del interior de los ejes.
- *scale_x_y* cambia la escala y la leyenda de x con el método y. En este caso, por ejemplo, se cambió el color a través de una paleta (*Blues*), pero se puede utilizar el método *manual* para ingresar los valores de forma manual, o utilizar el elemento *fill* en lugar de color si se tiene.

Estos son solamente algunos ejemplos de cómo editar gráficas producidas por *ggplot*. Sin embargo, existen temas específicos creados por la comunidad los cuales se pueden utilizar para dar cierto formato a las gráficas. La librería *ggthemes* contiene estos temas, que incluyen temas como *The Economist*, *Excel* o *clean*:

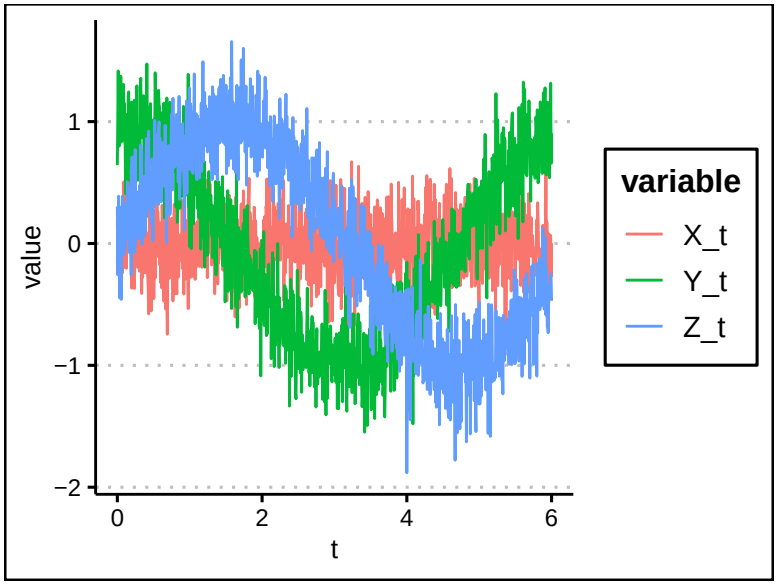
```
library(ggthemes)

ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  theme_base()
```



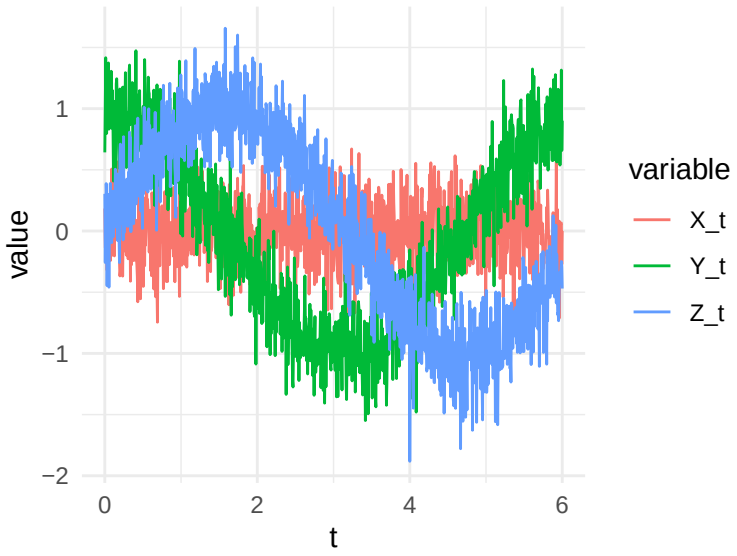

```
ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  theme_economist()
```



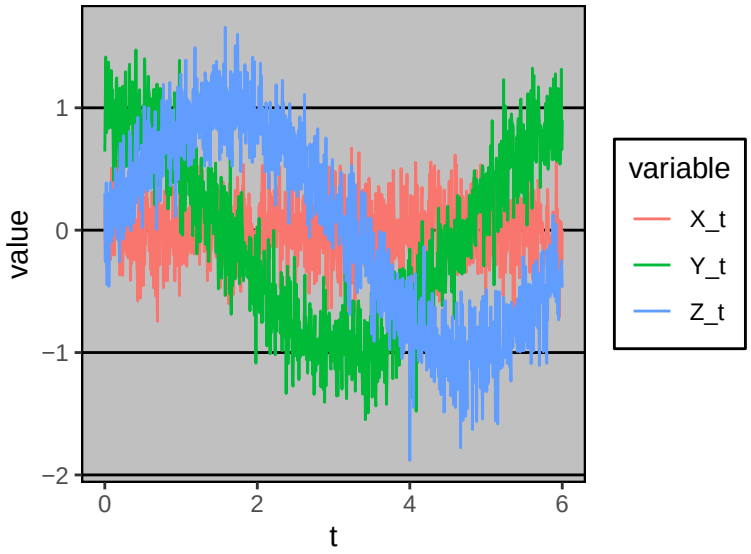

```
ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  theme_clean()
```




```
ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  theme_minimal()
```



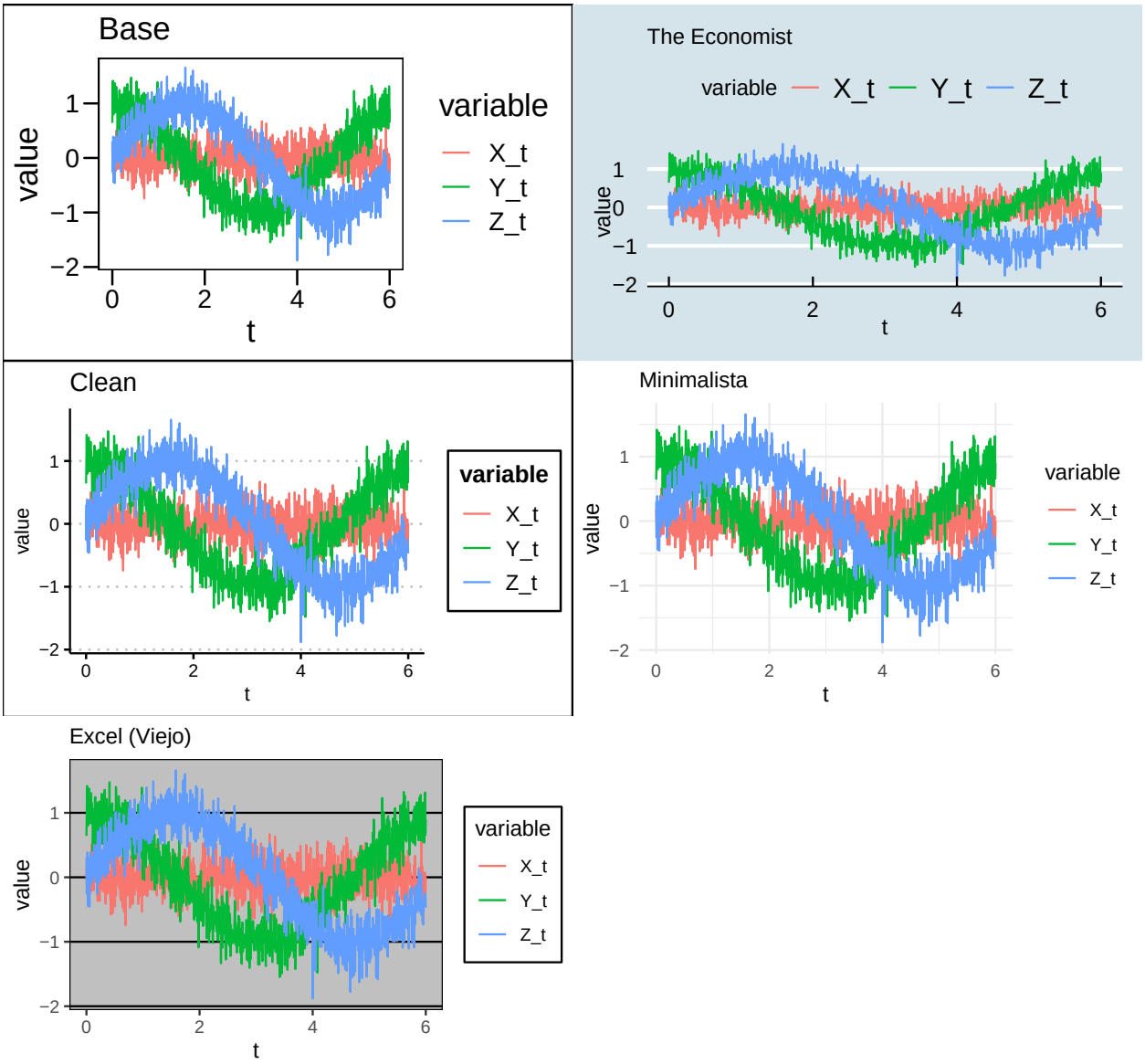

```
ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  theme_excel()
```



Finalmente, se puede utilizar la librería *gridExtra* para crear *plots* con múltiples gráficas:

```
library(gridExtra)

p1 <- ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  theme_base() +
  labs(subtitle = "Base")

p2 <- ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
```

```
  theme_economist()+
  labs(subtitle = "The Economist")

p3 <- ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  theme_clean() +
  labs(subtitle = "Clean")

p4 <- ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  theme_minimal() +
  labs(subtitle = "Minimalista")

p5 <- ggplot(datos_melt, aes(x = t, y = value, color = variable)) +
  geom_line() +
  theme_excel() +
  labs(subtitle = "Excel (Viejo)")

grid.arrange(p1,p2,p3,p4,p5, nrow = 3, ncol = 2)
```


Base
value
t
variable
X_t
Y_t
Z_t
The Economist
variable X_t Y_t Z_t
Clean
Minimalista
Excel (Viejo)