Gráficos en R (II)

Francisco Rodríguez Redondo

10 de noviembre de 2018

Índice general

1 Introducción.

En otro trabajo de este autor ya se han presentado los gráficos que se pueden crear con el paquete base de R. Son gráficos de alto nivel que permiten obtener representaciones gráficas bastante estéticas y potentes, pero tiene sus limitación.

Por ese motivo, se han venido desarrollando paquetes que mejoran las posibilidades que ofrecen esos gráficos y **en esa línea está el paquete gráfico de R** denominado ggplot2 que fue creado por Hadley Wickham en el año 2005 el cual hizo una implementación del trabajo denominado "The grammar of graphics" cuyo autor fue Leland Wilkinson publishes (año 1999), donde se describe formalmente los elementos y características de los gráficos estadísticos. Posteriormente a este último autor y en concreto en 2009Hadley Wickam publica otra obra «A layered grammar of graphics», que es una revisión del anterior trabajo y donde ya se organiza la gramática en capas, las cuales pueden sobreponerse y así mejorar de una forma fácil el gráficos. Estas capas, son hoy en día un elemento esencial de ggplot2.

ggplot2 contiene multitud de funciones y parámetros gráficos que permiten al usuario obtener mucho refinamiento en la elaboración de los gráficos, que como se ha dicho anteriormente, la idea consiste en crear capas e ir solapando dichos elementos hasta conseguir elementos gráficos con un alto nivel de calidad. El tutorial original de este paquete se puede encontrar en este enlace.

Antes de empezar a mostrar los elementos y comandos que ofrece este paquete, conviene inicialmente comprender dos formatos de presentación de los datos primarios: el formato wide frente al formato tidy. Para intercambiar ambos formatos se utiliza la librería «reshape2» (Ver un tutorial aquí) [1]. Con el formato wide se tendría en cada columna una variable, mientras que con el formato tidy, alguna de esas columnas se pasan a filas y se repite el nombre de la variable tantas veces como observaciones tengan los datos iniciales. Para comprender esto mejor, vamos a ver un ejemplo con el data set iris.


```
if(!require("reshape2")){install.packages("reshape2")}  ;library("reshape2")

## Loading required package: reshape2

print("formato wide del data set iris, ( en cada columna una variable)")

## [1] "formato wide del data set iris, ( en cada columna una variable)"

head(iris)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
```

[1] Aunque la librería lubridate también permite hacer este tipo de transformaciones

```
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Para pasar del formato wide al tyde se utiliza la función melt, de la siguiente forma:

```
#Añado la columna id que va a servir después para cambiar formatos
iris$id <- row.names(iris)
print("observar ahora el formato wide de iris con la nueva columna")

## [1] "observar ahora el formato wide de iris con la nueva columna"

print(head(iris))

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species id
## 1          5.1         3.5          1.4         0.2  setosa  1
## 2          4.9         3.0          1.4         0.2  setosa  2
## 3          4.7         3.2          1.3         0.2  setosa  3
## 4          4.6         3.1          1.5         0.2  setosa  4
## 5          5.0         3.6          1.4         0.2  setosa  5
## 6          5.4         3.9          1.7         0.4  setosa  6

iris.tidy <- melt(iris, id.vars=c("id", "Species"),
        variable.name="Part.Measure", value.name="Value")
print("formato tidy")

## [1] "formato tidy"

head(iris.tidy)

##   id Species Part.Measure Value
## 1  1  setosa Sepal.Length   5.1
## 2  2  setosa Sepal.Length   4.9
## 3  3  setosa Sepal.Length   4.7
## 4  4  setosa Sepal.Length   4.6
## 5  5  setosa Sepal.Length   5.0
## 6  6  setosa Sepal.Length   5.4
```

Como puede verse, se mantienen las columnas «id» y «Species», pero las otras columnas pasan a formar parte de las filas y se colocan en la nueva columna «Part.Measure», igualmente se añade una nueva columna «Value» que es la que refleja el valor concreto que figuraba en el dataset.

Si ahora queremos pasar de nuevo al formato «wide», se haría de la siguiente manera (utilizando la función dcast):

```
iris.wide <- dcast(iris.tidy, id + Species ~ Part.Measure, value.var=c("Value"))
head(iris.wide,3)

##    id    Species Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1   1     setosa          5.1         3.5          1.4         0.2
## 2  10     setosa          4.9         3.1          1.5         0.1
## 3 100 versicolor          5.7         2.8          4.1         1.3
```

Para entender mejor este concepto, pues es útil en la generación de gráficos con ggplot2, vamos a continuación a ver otro ejemplo más sencillo y corto y a buen seguro fijará esta

idea en el lector. Primero creamos un dataframe de R, que será el formato wide del mismo.

```
dat=data.frame(uno=c(1,2,3),dos=c(4,5,6),tres=c(7,8,9),cuatro=c(10,11,12))
print("Formato wide del dataframe")

## [1] "Formato wide del dataframe"

dat

##   uno dos tres cuatro
## 1   1   4    7     10
## 2   2   5    8     11
## 3   3   6    9     12
```

A continuación lo pasamos a un formato «tidy»

```
dat.tidy=melt(dat, id.vars=c("uno","dos"),variable.name="Transformada",value.name="Dato")
print("Formato tidy:")

## [1] "Formato tidy:"

print(dat.tidy)

##   uno dos Transformada Dato
## 1   1   4         tres    7
## 2   2   5         tres    8
## 3   3   6         tres    9
## 4   1   4       cuatro   10
## 5   2   5       cuatro   11
## 6   3   6       cuatro   12
```

Ahora hacemos el camino recíproco, y lo transformamos en un formato «wide».

```
dat.wide<-dcast(dat.tidy, uno+dos ~ Transformada, value.var=c("Dato"))
print("Formato wide:")

## [1] "Formato wide:"

print(dat.wide)

##   uno dos tres cuatro
## 1   1   4    7     10
## 2   2   5    8     11
## 3   3   6    9     12
```

Estos tipos de formatos son muy importantes en aquellos casos en los que se quiere hacer una representación gráfica, por ejemplo en diagramas de líneas, ya que en esto casos, el formato que se requiere es el tidy para tener en la columna que se repite el nombre de la variable, la columna donde se indica el tipo de gráfica al que pertenece cada uno de los puntos del dataframe a representar. Esto lo podemos ver en la figura 1.1, que aunque no se entienda en estos momentos bien su estructura, sí da idea del formato que deben seguir los datos para conseguir este objetivo.

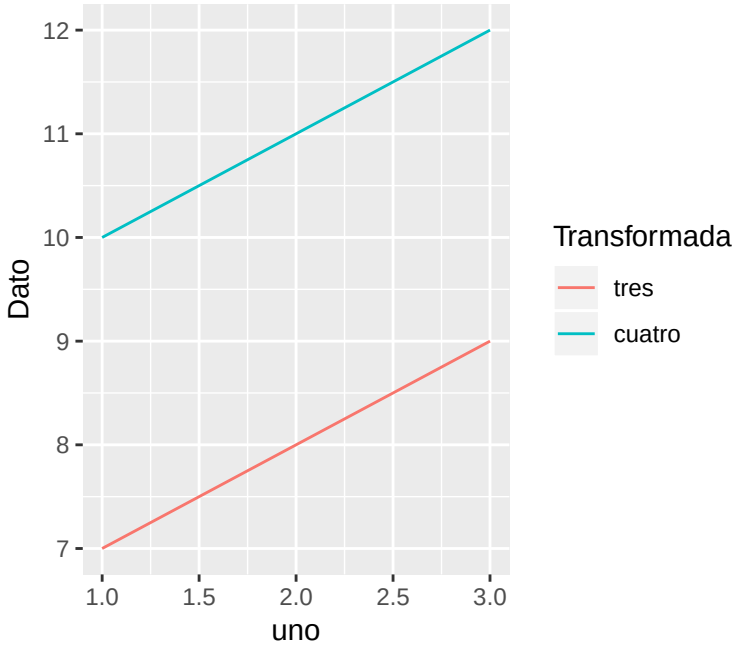

Figura 1.1: Gráfico de dos series de datos

```
## fig1
if(!require("ggplot2")){install.packages("ggplot2")}  ;library("ggplot2")

## Loading required package: ggplot2

p<-ggplot(dat.tidy,aes(x=uno,y=Dato,colour=Transformada))+geom_line()
p
```

El paquete ggplot2 tiene una función de alto nivel que de forma rápida nos puede facilitar la creación de gráficos, siguiendo el patrón de diseño de ggplot2. En el siguiente apartado se procede a su presentación.

1.1. Función qplot().

El formato para esta función que figura en el tutorial es el siguiente:

```
qplot(x, y, ..., data, facets = NULL, margins = FALSE, geom = "auto",
  xlim = c(NA, NA), ylim = c(NA, NA), log = "", main = NULL,
  xlab = NULL, ylab = NULL, asp = NA, stat = NULL,
  position = NULL)
```

Donde el significado de los parámetros es muy similar a los que se utilizan para la función plot() y que por lo tanto aquí no se explican, los nuevos significan los siguiente:

- **facets**. Sirve para descomponer el gráfico en varios subgráficos, en base a los valores de una variable categórica. Los valores pueden ser «facet_grid()» ó «facet_wrap()»,

y su significado se deja para más adelante cuando se estudie la función ggplot(). No obstante y para que se vaya conociendo su significado se puede adelantar que con este parámetro se genera un gráfico de tipo «trellis» mediante la especificación de una variable condicional. Su valor está representado de la siguiente forma: *variable_fila ~variable_ columna*. Si se quiere obtener un gráfico de tipo «trellis» con una sola variable condicional, se utilizará la expresión *variable_fila ~*. ó bien *. ~variable_ columna*.

- **geom**. Sirve para definir una geometría, es decir las capas que se van a representar. El valor por defecto es «point» si se especifica los valores de x e y. Si sólo se da el valor de x, el valor por defecto es «histogram».
- **log**. Sirve para hacer transformación de la variable. Puede valer «X», «Y» o bien «XY».
- **asp**. Es el valor de la relación que hay entre los valores del eje X y el eje Y. Si el valor es 1, no se cambia el aspecto del gráfico.

Veamos en la figura 1.2 un primer ejemplo. Como puede verse, se han creado dos gráficos con la intención de ver dos geometrías diferentes. Estos dos gráficos se han dispuesto en una matriz gráfica de una fila por dos columnas. Para conseguir esto en ggplot2, hay que instalar el paquete «gridExtra», y después ejecutar la función «grid.arrange» como se indica en el código.

```
## fig2
if(!require("gridExtra")){install.packages("gridExtra")}  ;library("gridExtra")

## Loading required package: gridExtra

q1<-qplot( data=iris, x=Sepal.Length,y=Sepal.Width, main="Datos iris. Puntos",
        xlab="Eje X",ylab="Eje Y",col=Species)
q2<- qplot( data=iris, x=Sepal.Length,y=Sepal.Width, main="Datos iris. lineas",
        geom="line",xlab="Eje X",ylab="Eje Y")
grid.arrange(q1, q2, ncol=2)
```

Para ver más ejemplos sobre esta función, se va a utilizar el conjunto de datos contenidos en «mtcars», cuyas estructura es la siguiente:

```
head(mtcars)

##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

A continuación se hacen algunos ajustes sobre determinadas variables, para convertirlas en variables de tipo factor, con los «labels» y «levels» que se indican a continuación.

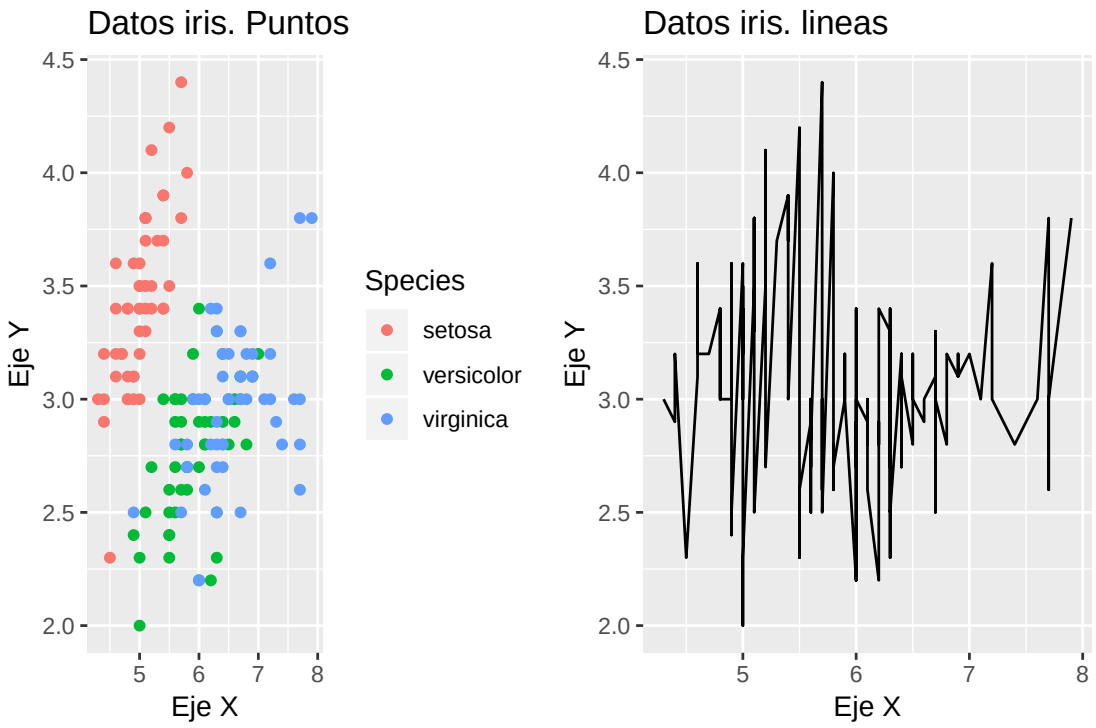

Figura 1.2: Primer gráfico con qplot

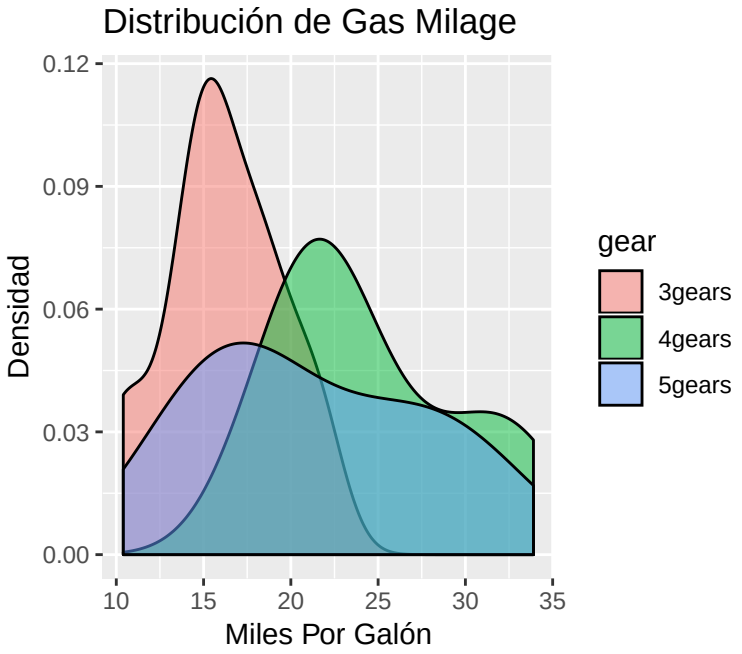

Figura 1.3: Densidades con qgplot

```
# Se crean factores con los labels y levels siguientes
mtcars$gear <- factor(mtcars$gear,levels=c(3,4,5),
        labels=c("3gears","4gears","5gears"))
mtcars$am <- factor(mtcars$am,levels=c(0,1),
        labels=c("Automatic","Manual"))
mtcars$cyl <- factor(mtcars$cyl,levels=c(4,6,8),
   labels=c("4cyl","6cyl","8cyl"))
```

En la figura 1.3 se muestra cómo poder estimar las densidades de la variable «mpg» pero distinguiendo con colores por la variable «gear». Esta distinción por colores se consigue con la opción «fill=gear».

```
### fig 3
# estimación de densidades para  mpg
# Agrupadas por número de  gears (diferenciados por el color)
qplot(mpg, data=mtcars, geom="density", fill=gear, alpha=I(.5),
   main="Distribución de Gas Milage", xlab="Miles Por Galón",
   ylab="Densidad")
```

Para un ejemplo de scaterplot, se muestra en la figura 1.4 un scaterplot de la variable «hp» y «mpg» distinguiendo además por el cruce de las variable «gear» y «cyl». Observar que también se hace distinción por el símbolo utilizado, cuando se emplea la expresión «shape=am».

```
###fig 4
# Scatterplot de mpg vs. hp para combinación de gears y cylinders
```

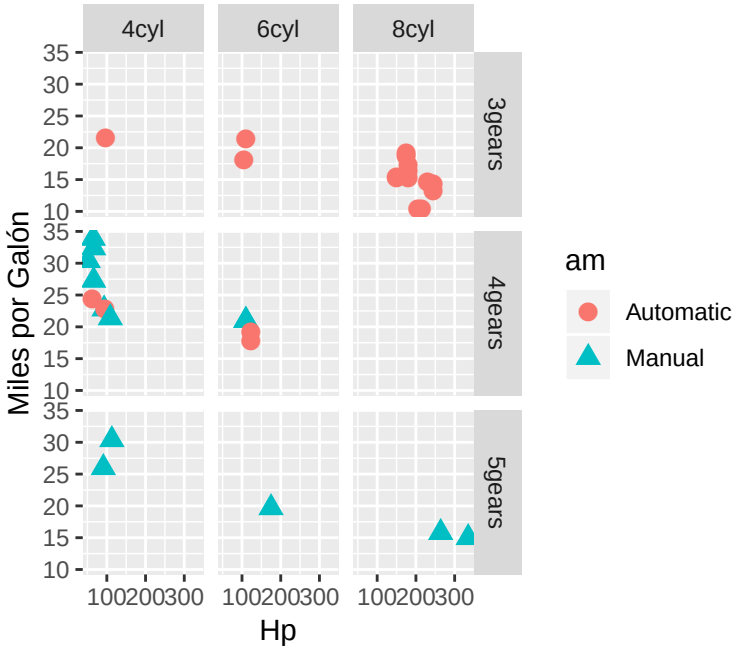

Figura 1.4: Ejemplo de scaterplot, con gráfico trellis

```
# el gráfico trellis se construye con facets=gear~cyl
qplot(hp, mpg, data=mtcars, shape=am, color=am,
   facets=gear~cyl, size=I(3),
   xlab="Hp", ylab="Miles por Galón")
```

Un modelo de regresión se obtiene en la figura 1.5, donde se obtiene una recta de regresión para cada valor de la variable «cyl». Observar que sale un mensaje de alerta que dice: «Warning: Ignoring unknown parameters: method, formula». Sin embargo el resultado que se obtiene es el de una recta de regresión.

```
### fig 5
# Regresión de mpg sobre wt para cada cada valor dado por cyl
qplot(wt, mpg, data=mtcars, geom=c("point", "smooth"),
   method="lm", formula=y~x, color=cyl,
   main="Regresión of MPG sobre peso",
   xlab="Peso", ylab="Miles por Galón")

## Warning: Ignoring unknown parameters: method, formula
```

Si quitamos el parámetro «method» y «formula», se puede comprobar que el resultado es diferente, y es el que se muestra en la figura 1.6 que como puede verse el resultado es bien diferente al anterior, y además la misma salida del código indica indica que el método utilizado es el método «loess» que por lo tanto es el valor por defecto para este tipo de gráficos.

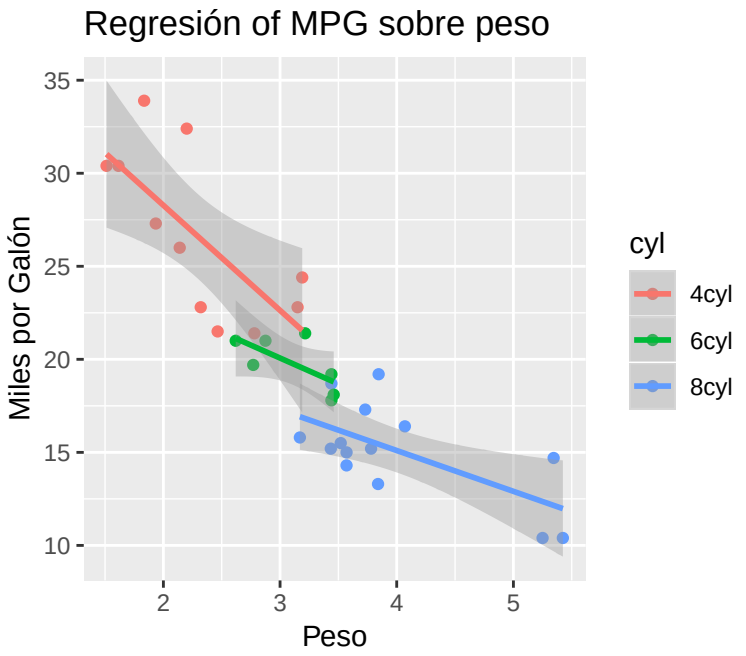

Figura 1.5: Regresión de tipo líneal para cada valor cyl

```
## fig 6
qplot(wt, mpg, data=mtcars, geom=c("point", "smooth"),
   color=cyl,
   main="Regresión of MPG sobre peso",
   xlab="Peso", ylab="Miles por Galón")

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

Por último, vamos a representar un «boxplot», cuya construcción es bien fácil y en resultado obtenido es el mostrado en la figura 1.7. Se puede ver que se obtiene como resultado tres boxplot's uno por cada valor de la variable «gear», lo cual se indica en el parámetro «fill=gear».

```
###fig 7
# Boxplot's of mpg por cada valor de gear
# Los puntos son movidos levemente mediante "jittered"
qplot(gear, mpg, data=mtcars, geom=c("boxplot", "jitter"),
   fill=gear, main="mpg por gear",
   xlab="", ylab="Miles por Galón")
```

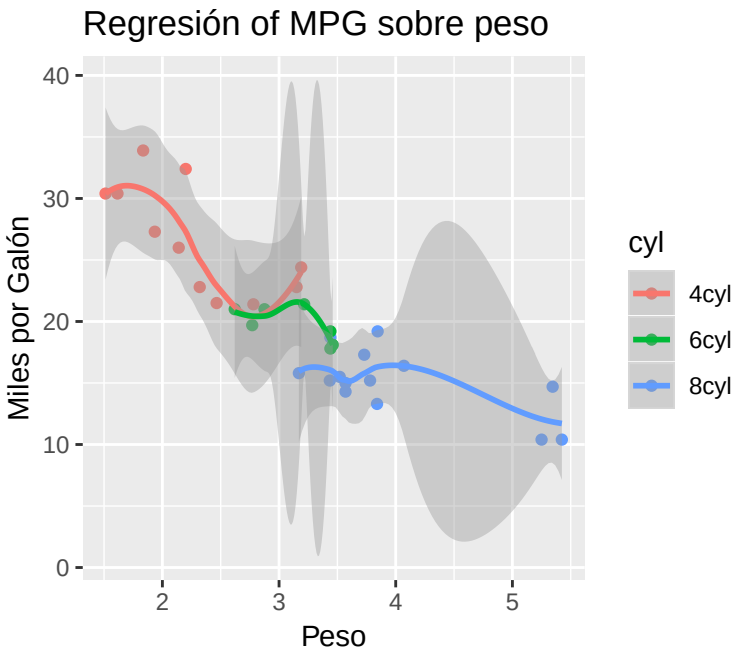

Figura 1.6: Regresión con método loess

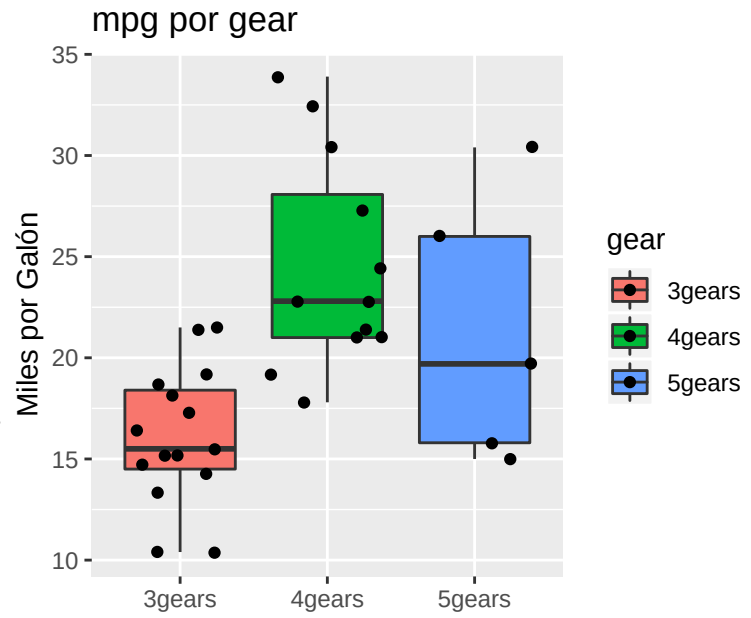

Figura 1.7: Representación de Boxplot's

2 ggplot().

En el capítulo anterior, se han visto de una manera sucinta las posibilidades que ofrece qgplot(), que como ya se ha comentado es un sucedáneo de ggplot(), de tal manera que aunque qgplot presenta como positivo su facilidad de uso, también ofrece el inconveniente de su limitación a la hora de ponerlo en práctica. Para mejorar y potenciar esto está la posibilidad de utiliza ggplot(), que es algo más difícil de utilizar pero por contra presenta muchas posibilidades que qgplot(). En lo que sigue me voy a centrar en presentar ggplot(), junto con las posibilidades que ofrece a la hora de generar gráficos estéticos y potentes de datos estadísticos. La estructura y las componentes del panel gráfico que utiliza ggplo2 y a las que se hará referencia en apartados posteriores, se muestran en la figura 2.1

2.1. Capa de Geometrías y aestética.

Como ya se ha dicho anteriormente ggplot2 es una herramienta gráfica que se distingue entre otras cosas, porque se puede presentar la información en capas, que sobrepuestas unas con otras nos permiten obtener los gráficos que deseamos. Igualmente otro elemento básico de ggplot2 es la aestética. En ggplot2, se entiende por aestética a un conjunto de elementos tales como por ejemplo la distancia horizontal y vertical a los ejes, el clor, la forma (shape) de los puntos, el tamaño de los puntos o el grosor de las líneas. Normalmente los elementos de la aestética se introducen dentro de la función aes(), a continuación se muestra un pequeño ejemplo de lo que es una aestética.

```
p<-gglop(data=iris,aes(x=Petal.Length, y=Petal.Width, color=Species)
```

En el ejemplo anterior, la información de aestética es la siguiente:

- **x=Petal.Length** ; es la información de la distancia horizontal, y viene dada por la longitud del pétalo.
- **y=Petal.width** ; es la información de la distancia vertical, y viene dada por su anchura.
- **color=Species** ; es el color que se utilizará para colorear los puntos, cada clase tiene un color diferente.

El objeto p que se ha construido anteriormente, aun no es un gráfico, pues la falta la información de las capas, no obstante este elemento se puede inspeccionar con la siguiente instrucción.

```
summary(p)
```

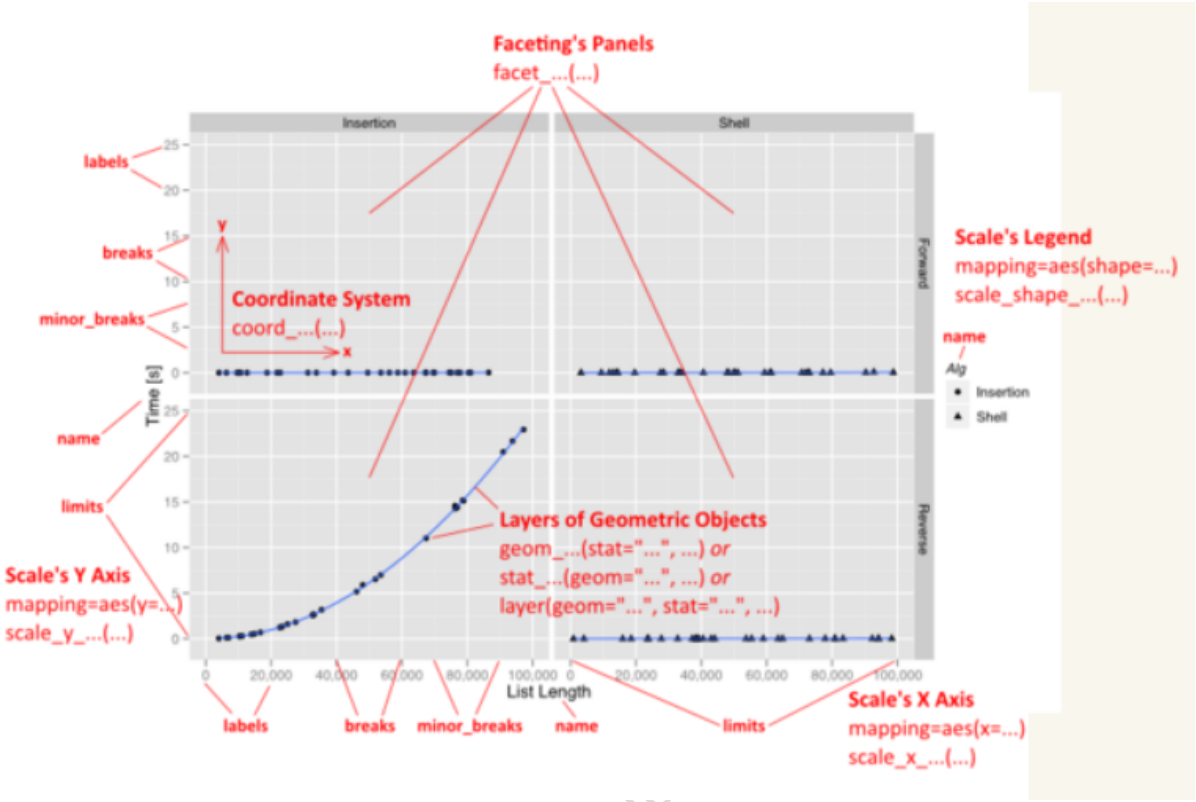

Figura 2.1: Panel Gráfico ggplot2

Existen alrededor de una docena de aestéticas, pero las más frecuentes son las siguientes:

- **x e y** que son las distancias horizontal y vertical.
- **colour**. Para definir el color de cada clase de puntos o líneas.
- **shape**. Indica la forma de los puntos (cuadrados, triángulos, etc.), o la forma del trazo de las líneas (continuo, punteado, etc.)
- **alpha**. Para indicar la transparencia. Valores altos son para formas opacas, y cuanto más bajos, más transparentes.
- **fill**. Para indicar el color de relleno de las formas sólidas, como pueden ser las barras, el piechart, etc.

Las capas, también conocidas como geometrías indican qué hacer con los datos y las aestéticas elegidas, es decir cómo se representan en el lienzo del dibujo. De esta manera a continuación en el gráfico vemos un ejemplo sencillo de cómo actúan tanto la geometría punto, como unos elementos determinados de aestéticas.

```
fig 2bis
p<-ggplot(iris,aes(x=Petal.Length, y=Petal.Width, color=Species))
summary(p)
#Añadimos una geometría o capa de puntos.
p<-p+geom_point()
#ahora lo imprimimos, para que se vea el gráfico.
p

## Error: <text>:1:5: unexpected numeric constant
## 1: fig 2
##        ^
```

Vemos que se dibujan todos los puntos del conjunto estudiado y cada punto tiene un color diferente para cada una de las categorías de la variable «Species» (esto se indica con «color=Species»).

En los ejemplos que siguen vamos a utilizar otro conjunto de datos que vienen cargados por defecto en ggplot2. Este conjunto de datos se domina «diamons» y tiene la siguiente estructura.

```
head(diamonds,2)

## # A tibble: 2 x 10
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal   E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium E     SI1      59.8    61   326  3.89  3.84  2.31
```

En los dos ejemplos que se van a mostrar a continuación, se muestran para ver la diferencia que hay entre poner el parámetro color dentro o fuera de una aestética. Si se pone dentro de el valor que se le asigna es el de una determinada variable para que los elementos dibujados tengan un color diferente y de acuerdo al valor de esa variable. Si se coloca fuera de una aestética, entonces el valor que toma es uno determinado y todos los elementos tendrán ese color.

```
###fig8
ggplot(diamonds,aes(x=carat,y=price))+geom_point(color="blue")
```

En la figura 2.2 se muestra un gráfico donde el color se ha indicado fuera es la aestética y por lo tanto todos los puntos tienen el mismo color azul. Sin embargo en la figura 2.3 el parámetro «colour» queda dentro de una aestética y se le asigna el valor de una variable, por lo que los puntos tendrán un color diferente y dependiente de la clase «color» a la que pertenezcan.

```
###fig9
ggplot(diamonds,aes(x=carat,y=price))+geom_point(aes(colour=color))
```

La transparencia de los puntos es otro factor que se puede controlar utilizando para ello el parámetro alpha. En la figura 2.3 se muestran dos ejemplos, uno sin el parámetros transparencia, y el otro controlando la transparencia mediante la variable «clarity».

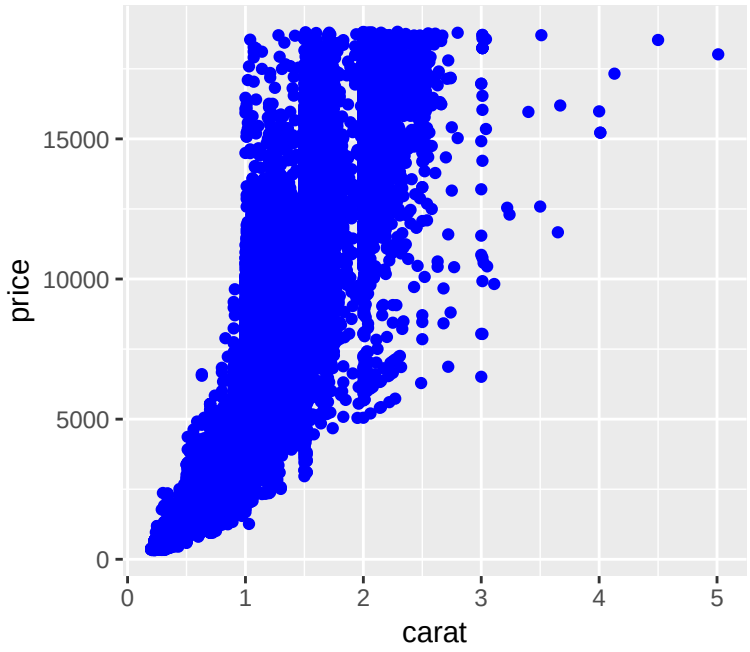

Figura 2.2: Scaterplot de caret y price (un solo color)

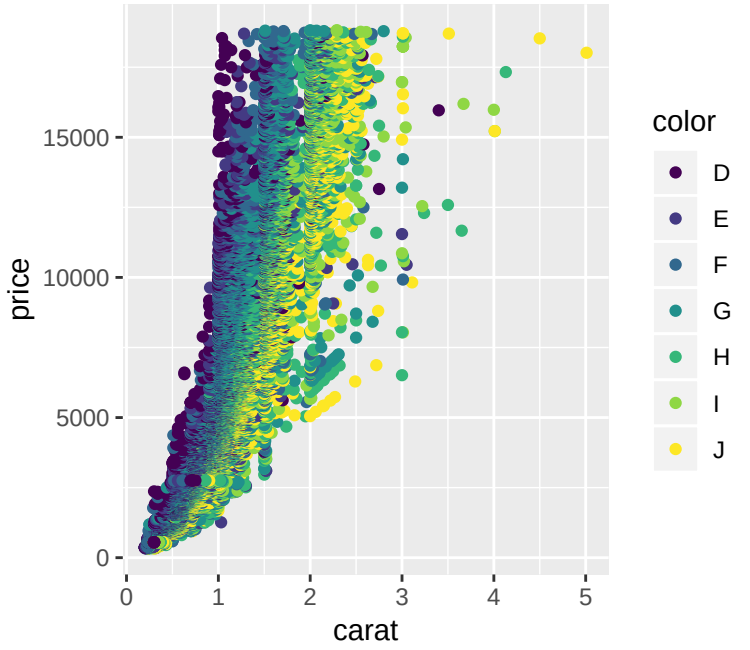

Figura 2.3: Scaterplot de caret y price con color diferente

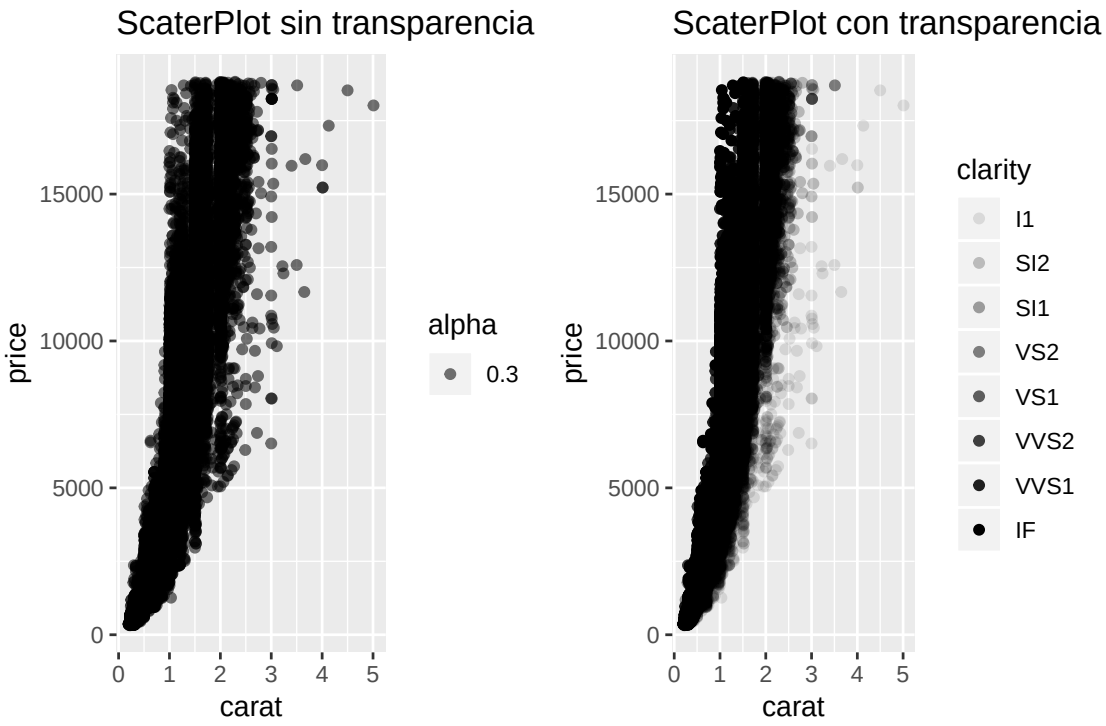

Figura 2.4: Sacterplot con diferente transparencia de los puntos

```
###fig9
p1<- ggplot(diamonds,aes(x=carat,y=price,alpha=0.3))+geom_point()+
        ggtitle("ScaterPlot sin transparencia")
p2<- ggplot(diamonds,aes(x=carat,y=price,alpha=clarity))+geom_point()+
        ggtitle("ScaterPlot con transparencia ( clarity)")

grid.arrange(p1, p2, ncol=2)
```

Otra posibilidad es definir diferente tamaño de los puntos o diferente forma (shape) para representar los puntos. Estas características las podemos ver en la figura 2.5.

```
###fig11
#Cambiando la forma (shape) de los puntos
p1<-ggplot(diamonds,aes(x=carat,y=price,shape=cut))+geom_point()+
        ggtitle("ScaterPlot con formas diferentes")
#Cambiando el tamaño de los puntos
p2<-ggplot(diamonds,aes(x=carat,y=price,size=depth))+geom_point()+
        ggtitle("ScaterPlot con diferentes tamaños")
grid.arrange(p1,p2,ncol=2)

## Warning: Using shapes for an ordinal variable is not advised
```

Existen muchos tipos de geometrías en ggplot2, el total de ellas se pueden encontrar en esta dirección web. De entre las más utilizadas se pueden destacar las siguiente:

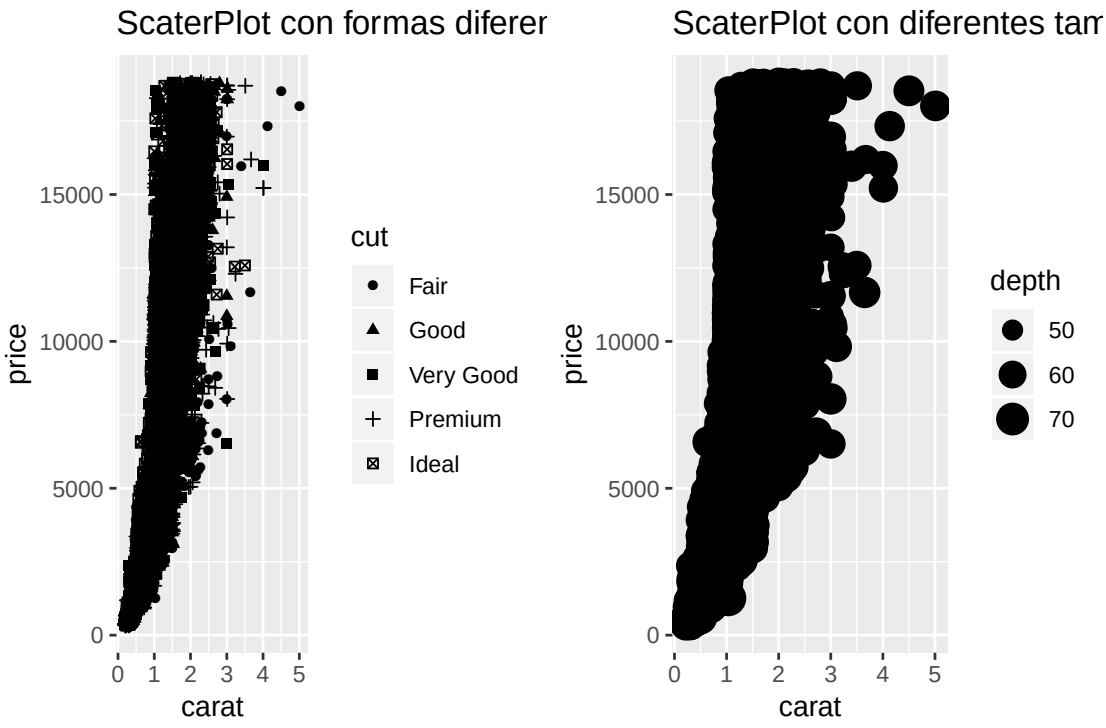

Figura 2.5: Formas para dibujar los puntos, y tamaños de los mismos

- **geom_abline()**, **geom_hline()**, **geom_vline()**, Utilizadas para dibujar una línea con una pendiente y una ordenada en el origen (primer caso), una línea horizontal (segundo caso) y una línea vertical (tercer caso).
- **geom_bar()**, Para generar gráficos de barras.
- **geom_blank()** , Para generar un plot sin contenido.
- **geom_boxplot()**, Para generar un gráfico de tipo boxplot.
- **geom_contour()**. Para generar contornos.
- **geom_density()** , Para generar gráficos de densidad estimada.
- **geom_jitter()** . Mete algo de ruido a los puntos.
- **geom_freqpoly() geom_histogram()**, Para generar histogramas y polígonos de frecuencias.
- **geom_map()**, para generar mapas.
- **geom_point()** , para generar diagramas de puntos.
- **geom_smooth()** , sirve para representar curvas y rectas de regresión, junto a sus intervalos de confianza.
- **geom_violin()** , para dibujar violin plot.

NOTA: Una lista completa de geometrías se pueden encontrar en este enlace.

Veamos a continuación algunos ejemplos donde se utilizan este tipo de geometrías. En la figura 2.6 se pueden ver dos gráficos. En el de la izquierda se representan los puntos con un color rojo, y en el de la derecha se puede ver el efecto de un pequeño ruido (generado con geom_jitter). En este último caso se han recortado los valores tanto del eje X como del ee Y, para apreciar mejor el resultado.


```
###fig12
p1<- ggplot(diamonds,aes(x=carat,y=price))+geom_point(col="red")+
        ggtitle("Diagrama de puntos")+ theme(legend.position="none")
p2<- ggplot(diamonds,aes(x=carat,y=price,size=depth))+geom_point(size=0.6)+
        geom_jitter(col='red',alpha=0.2,size=0.6)+xlim(2,3)+ylim(10000,15000)+
        ggtitle("Efecto del ruido generado con geom_jitter")+ theme(legend.position="none")
grid.arrange(p1,p2,ncol=2)

## Warning: Removed 53066 rows containing missing values (geom_point).
## Warning: Removed 53122 rows containing missing values (geom_point).
```

Se pueden obtener líneas de ajustes de regresión, y en la figura 2.7 se muestran dos ejemplos. En la parte de la izquierda, se utiliza el método de regresión «loess» mientras que en el ejemplo de la derecha el método usado es de regresión lineal.

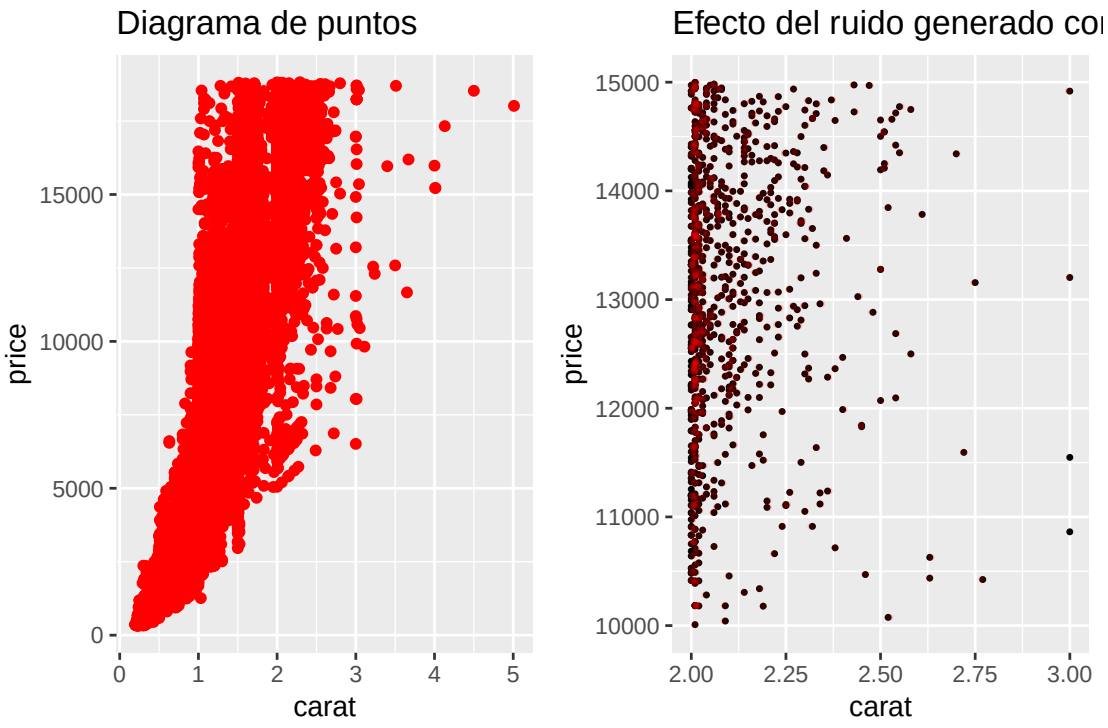

Figura 2.6: Diagrama de puntos y ruido sobre ellos

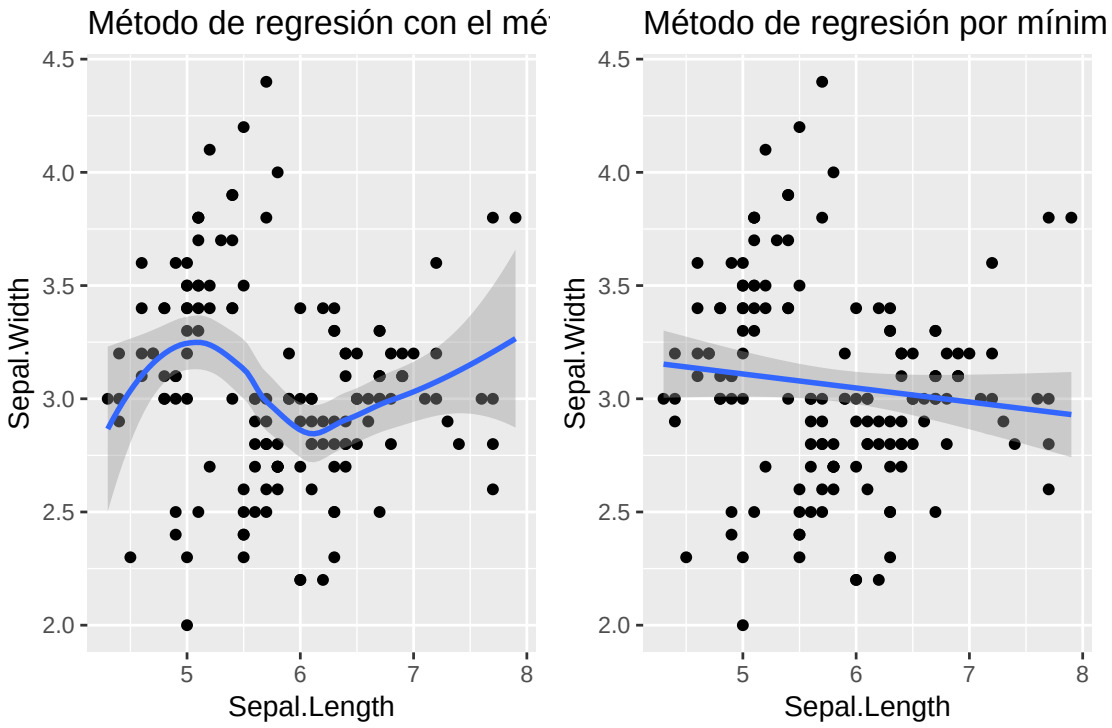

Figura 2.7: Métodos de regresión con ggplot2

```
###fig13
p1<-ggplot(iris, aes(x=Sepal.Length,y=Sepal.Width))+geom_point()+geom_smooth()+
        ggtitle("Método de regresión con el método loess")
p2<-ggplot(iris, aes(x=Sepal.Length,y=Sepal.Width))+geom_point()+geom_smooth(method = lm)+
        ggtitle("Método de regresión por mínimos cuadrados")
grid.arrange(p1,p2,ncol=2)

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

2.2. Las escalas en ggplot2.

Otra aspecto muy importante de ggplot2 son las escalas, que sirven o se utilizan para controlar los detalles de cómo los datos y los elementos gráficos se muestran en la figura que se cree. Existen también muchas escalas en ggplot2, cada una especializada en un tema concreto. Las escalas más utilizadas a la hora de generar un gráfico, son las que se detallan a continuación.

- **scale_alpha()** , Una escala para controlar la transparencia de los objetos.
- **scale_alpha_continuous()** , similar a la anterior para variables continuas.
- **scale_alpha_discrete()** , idem para variables discretas.

- **scale_x_continuous() scale_y_continuous()**. Para definir el aspecto del eje X e Y para variables de tipo continuo.
- **scale_x_date() , scale_y_date()**. Para definir aspecto ejes en caso de variables de tipo date().
- **scale_x_datetime() ,scale_y_datetime()**, como antes para variables de tipo datetime()
- **scale_x_discrete() scale_y_discrete()**, para aspecto eje x y eje y en caso de variables discretas.
- **scale_x_log10()** , para obtener una escala logarítmica.
- **scale_y_reverse()** , para invertir el orden de presentación de los datos.

En la figura 2.8 se muestran dos gráficos. En el de la izquierda, se representa un scater plot normal, y en el de la derecha el mismo scater plot pero modificando el eje x (se limitan los valores, y se definen las marcas del eje x).

```
### fig14
#Sin escala
p1<- ggplot(iris,aes(x=Sepal.Length,y=Sepal.Width))+geom_point()+
        ggtitle("Gráfico sin escala")
#Con escala eje x para variable tipo continua
p2<-p1+scale_x_continuous(name='prueba eje x',
                      limits = c(5,7),
                      breaks = seq(5,7,0.5),
                      minor_breaks = seq(5,7,0.25))+
        ggtitle("Gráfico con una escala continua para eje x")
grid.arrange(p1,p2,ncol=2)

## Warning: Removed 34 rows containing missing values (geom_point).
```

En el caso de variables de tipo discreto también existe la opción de adaptar a nuestras necesidades el formato de los ejes que queremos representar. En la figura reffig:fig15 mostramos su uso. De esta manera en la gráfica de la izquierda, se genera un gráfico normal y en el de la derecha se modifica la presentación de los datos del eje x que es de tipo discreto y la del eje y que es de tipo continuo.

```
###  fig15
p1<- ggplot(iris, aes(x=Species, y=Sepal.Length) )+geom_jitter()+
        ggtitle("Gráfico con geom_jitter")

p2<-p1+ scale_x_discrete( name="Eje X",
                  limits=c("setosa","versicolor"),
                  label=c("Tipo1","Tipo2"))+
  scale_y_continuous( limits = c(5,7))+
        ggtitle("Cambios de escala en los dos ejes")
grid.arrange(p1,p2,ncol=2)

## Warning: Removed 77 rows containing missing values (geom_point).
```

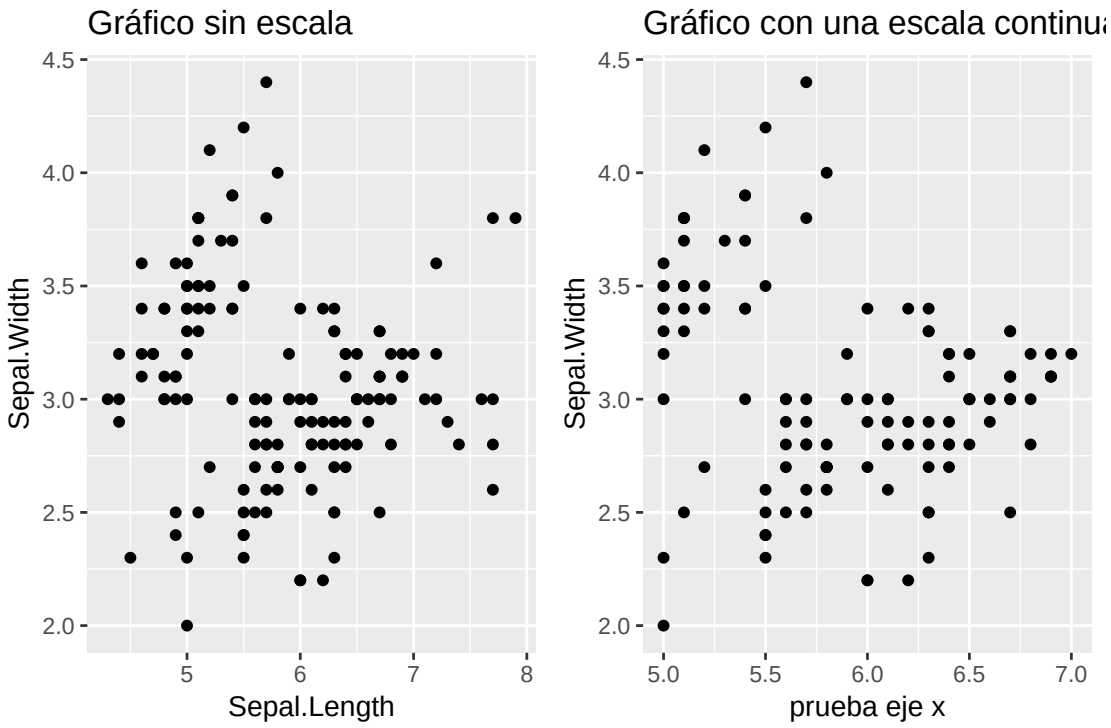

Figura 2.8: Utilización escala tipo continua

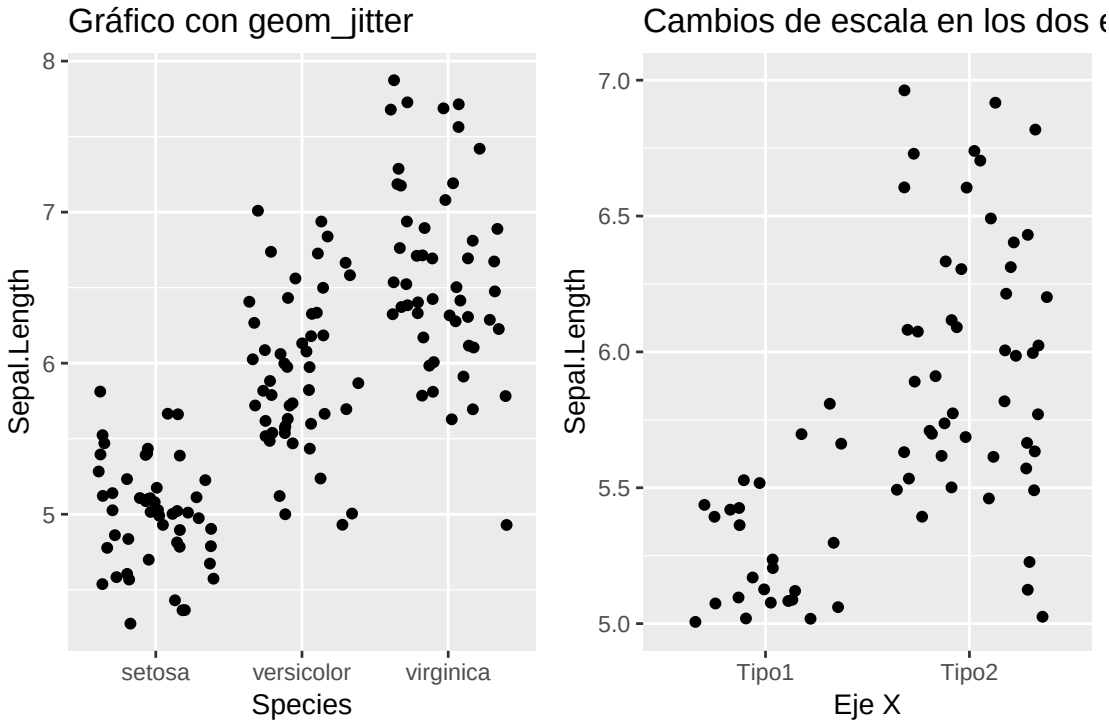

Figura 2.9: Cambio de escalas de los dos ejes

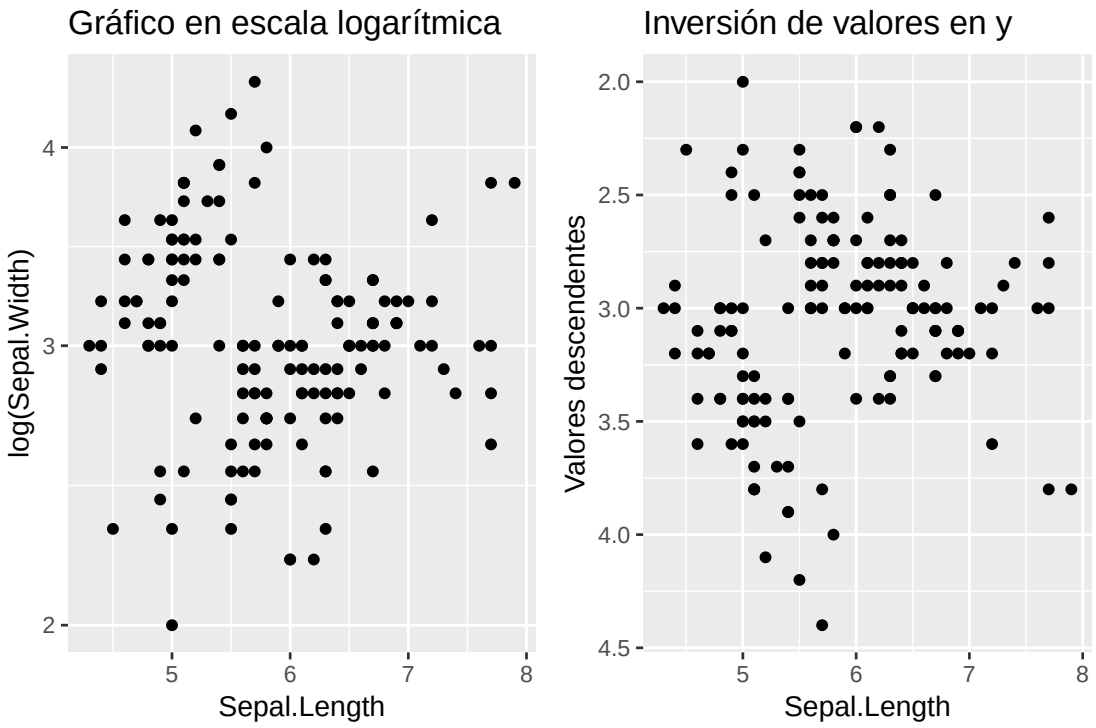

Figura 2.10: Ejes logarítmicos e inversión de un eje

Respecto al uso de escalas logarítmicas, en la figura reffig:fig16 mostramos otros dos gráficos. El de la izquierda se le ha añadido una escala logarítmica al eje y y en el de la derecha se le ha invertido el orden de colocación de los valores de ese mismo eje y.

```
###fig16
p1<- ggplot(iris, aes(x=Sepal.Length,y=Sepal.Width))+ geom_point()+
  scale_y_log10(name="log(Sepal.Width)")+
        ggtitle("Gráfico en escala logarítmica")
p2<-p1+scale_y_reverse(name="Valores descendentes")+
        ggtitle("Inversión de valores en y")

## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.

grid.arrange(p1,p2,ncol=2)
```

En los colores que se utilizan para la generación del gráfico, también se puede utilizar una escala de color para conseguir un gradiente de colores, para hacer eso, existen varias escalas de colores, se puede consultar las página de ggplot2 que se ha indicado anteriormente, y en la figura 2.11 se muestra alguna posibilidad sobre este aspecto. En el gráfico de la izquierda, se utilizan dos colores, mientras que en el de la derecha son tres y se obliga a pasar por uno intermedio. Hay muchas más posibilidades, y las misma se pueden ver como ya se ha dicho antes en la página de este paquete.

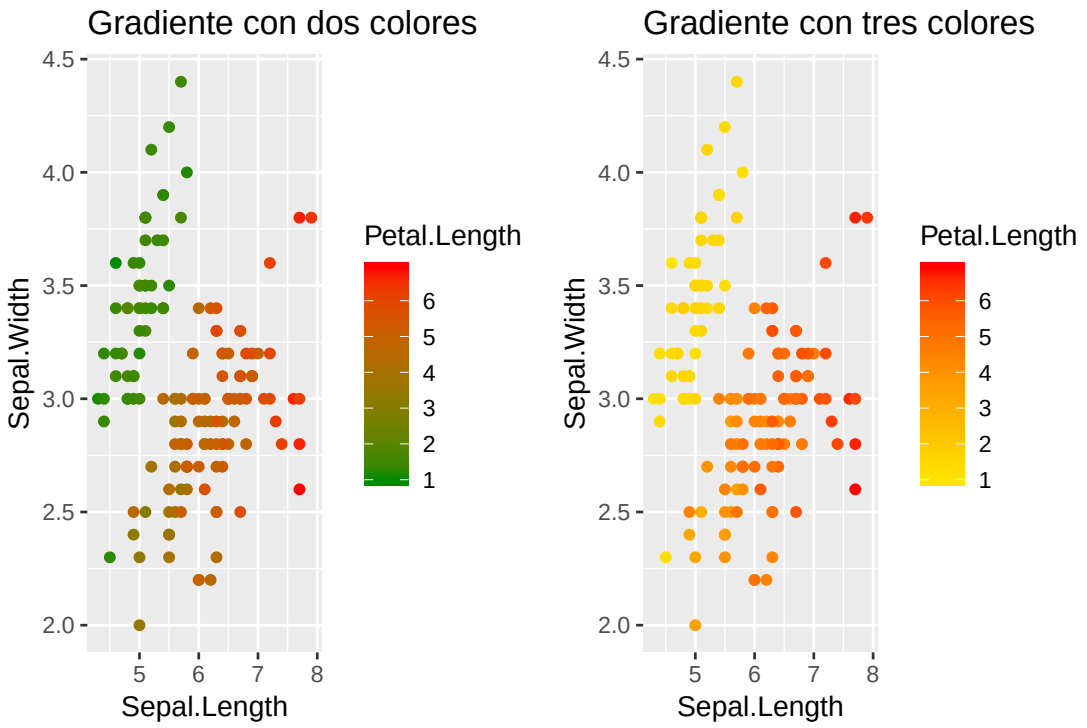

Figura 2.11: Gradientes de color

```
##fig17
p1<-ggplot(iris,aes(x=Sepal.Length,y=Sepal.Width,col=Petal.Length))+geom_point()+
  scale_color_gradient(low = "green4", high = "red1")+
        ggtitle("Gradiente con dos colores")
p2<- ggplot(iris,aes(x=Sepal.Length,y=Sepal.Width,col=Petal.Length))+geom_point()+
  scale_color_gradient2(low = "green4", high = "red1", mid = "yellow")+
        ggtitle("Gradiente con tres colores")
grid.arrange(p1,p2,ncol=2)
```

Para ver la gran variedad de opciones que podemos utilizar con una escala de colores, en la figura 2.12 se muestran otros dos formatos de elección de la escala de colores.

```
####fig18
#Para utilizar n colores
p1<-ggplot(iris,aes(x=Sepal.Length,y=Sepal.Width,col=Petal.Length))+geom_point()+
  scale_color_gradientn(colours = c("green4", "yellow", "red1", "blue4"))+
        ggtitle("Usando n colores")
#utilizando una paleta de colores
p2<- ggplot(iris,aes(x=Sepal.Length,y=Sepal.Width,col=Petal.Length))+geom_point()+
  scale_color_gradientn(colours = rainbow(5))+
        ggtitle("Con una paleta de colores")
grid.arrange(p1,p2,ncol=2)
```

También existen escalas de colores para variables de tipo discretas, en la figura 2.13 se muestran tres ejemplos de ello más otra figura en la que se utiliza otro tipo de escala .

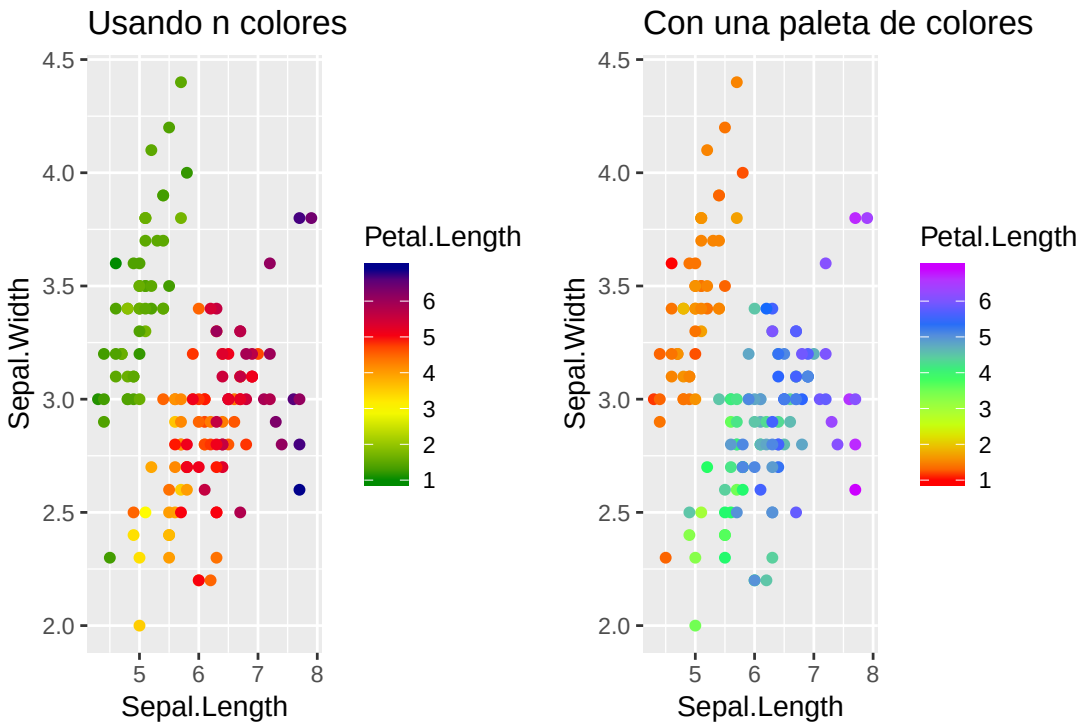

Figura 2.12: Otras escalas de colores

```
### fig19
###Utilizamos el conjunto de datos mtcars, veamos algunos registros
head(mtcars)
```

```
##                    mpg  cyl disp  hp drat    wt  qsec vs        am   gear
## Mazda RX4         21.0 6cyl  160 110 3.90 2.620 16.46  0    Manual 4gears
## Mazda RX4 Wag     21.0 6cyl  160 110 3.90 2.875 17.02  0    Manual 4gears
## Datsun 710        22.8 4cyl  108  93 3.85 2.320 18.61  1    Manual 4gears
## Hornet 4 Drive    21.4 6cyl  258 110 3.08 3.215 19.44  1 Automatic 3gears
## Hornet Sportabout 18.7 8cyl  360 175 3.15 3.440 17.02  0 Automatic 3gears
## Valiant           18.1 6cyl  225 105 2.76 3.460 20.22  1 Automatic 3gears
##                   carb
## Mazda RX4            4
## Mazda RX4 Wag        4
## Datsun 710           1
## Hornet 4 Drive       1
## Hornet Sportabout    2
## Valiant              1
```

```
p1<- ggplot(mtcars, aes(x = cyl, fill=factor(gear))) +
  geom_bar()+ggtitle("Gráfico barras")

#Escala de grises
p2<- ggplot(mtcars, aes(x = cyl, fill=factor(gear))) +
  geom_bar()+ scale_fill_grey()+ggtitle("Escala grises")

# Con paleta de colores
p3<-ggplot(mtcars, aes(x = cyl, fill=factor(gear))) +
  geom_bar()+scale_fill_brewer(palette = "Blues")+
        ggtitle("Paleta RColorBrewer")

# Otro tipo de escala
p4 <- ggplot(faithfuld, aes(waiting, eruptions, fill = density)) +
  geom_tile()+scale_fill_continuous(type = "gradient")+
        ggtitle("Otro tipo de escala")
grid.arrange(p1,p2,p3,p4,ncol=2)
```

2.3. Diagrama de sectores.

Los diagramas de sectores o también conocidos como piechars, se obtienen en ggplot2 de una forma un tanto peculiar. Para ello, se utilizan las coordenadas polares. En la figura 2.14 se puede ver cómo representar la variable «cyl» del conjunto de datos «mtcars».

```
p1 <- ggplot(mtcars, aes(x = factor(1), fill = factor(cyl))) +
 geom_bar(width = 1)+ coord_polar(theta = "y")

p2 <- ggplot(mtcars, aes(x = factor(cyl))) +
  geom_bar(width = 1, colour = "black")+ coord_polar()

p3<-p2 + coord_polar(theta = "y")
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing
one.
```

```
p4<- ggplot(mtcars, aes(x = factor(1), fill = factor(cyl))) +
 geom_bar(width = 1)+ coord_polar()

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

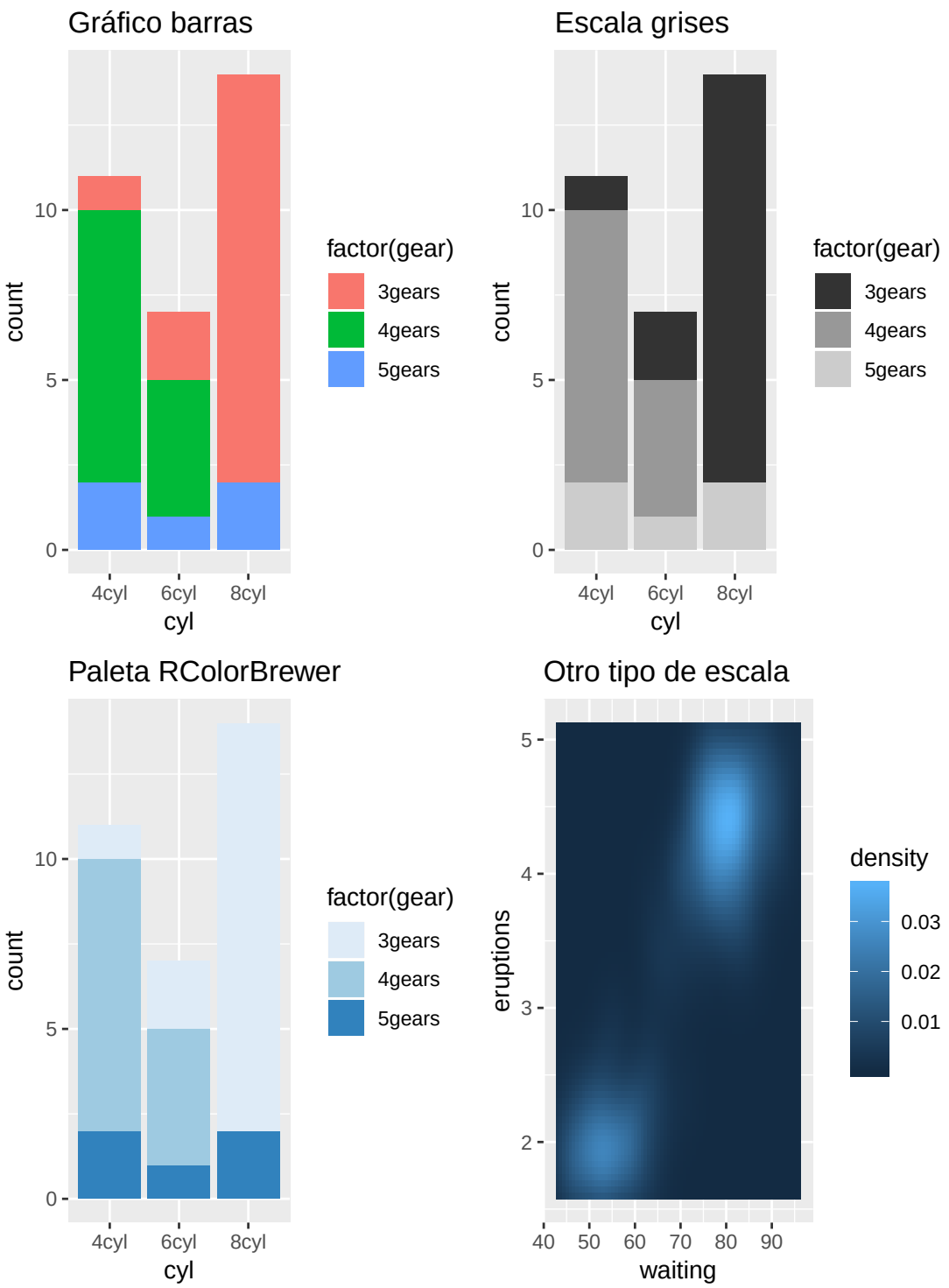

Figura 2.13: Otros ejemplos de escalas de color

Otros formatos que es posible obtener con este método se muestra en la figura 2.15

```
p1<-ggplot(mtcars, aes(x = cyl, fill = factor(cyl)))
p1 <- p1+geom_bar()

p2<-p1+coord_polar()

p3<-p1+ coord_polar(theta="y")
grid.arrange(p2,p3,ncol=2)
```

2.4. Utilización de temas.

Los temas de ggplo2 están pensados e implementados para modificar aspectos de los gráficos que no tienen una relación directa con los datos, como por ejemplo los ejes, las etiquetas, el color de fondo, etc. Un tema esta constituido por un conjunto de elementos (por ejemplo, los paneles, la leyenda, etc.) sobre los cuales se puede incidir para modificarlo. El tema que por defecto utiliza ggplot2 es el «theme_grey()» y es el que se ha visto en todos los ejemplos anteriores.

Los temas no solo son modificables, sino que además se puede elegir cualquier otro de los que por defecto tiene ggplot2. Por ejemplo en la figura 2.16 se muestra cómo utilizar el tema «theme_bw()», aunque de igual manera se puede utilizar otro tema de los que por defecto ya tiene instalado ggplot2.

```
##fig20
p<-ggplot(iris,aes(x=Petal.Length, y=Petal.Width, color=Species))+geom_point()
p<-p+theme_bw()
print(p)
```

También se puede modificar alguno de los elementos del tema, por ejemplo en la figura 2.17 modificamos el color del lienzo y el tipo de línea con la que se dibuja la malla del gráfico. Como puede verse en esta figura, existen una serie de elementos de muy frecuente uso a la hora de trabajar con los temas.

```
##fig21
p<-ggplot(iris,aes(x=Petal.Length, y=Petal.Width, color=Species))+geom_point()
p<-p+theme_bw()+
        theme(
                panel.background=element_rect(fill="lightblue"),
                panel.grid.minor=element_line(linetype="dotted")
        )
print(p)
```

Los elementos de más frecuente uso son:

- **margin()** . Para definir los márgenes.
- **element_blank()** . Para definir un elemento en blanco. Por ejemplo, si en un tema se pone «axis.text=element_blank()», no se verán los ejes.
- **element_rect()** . Se utiliza para modificaciones en los bordes y el fondo gráfico.

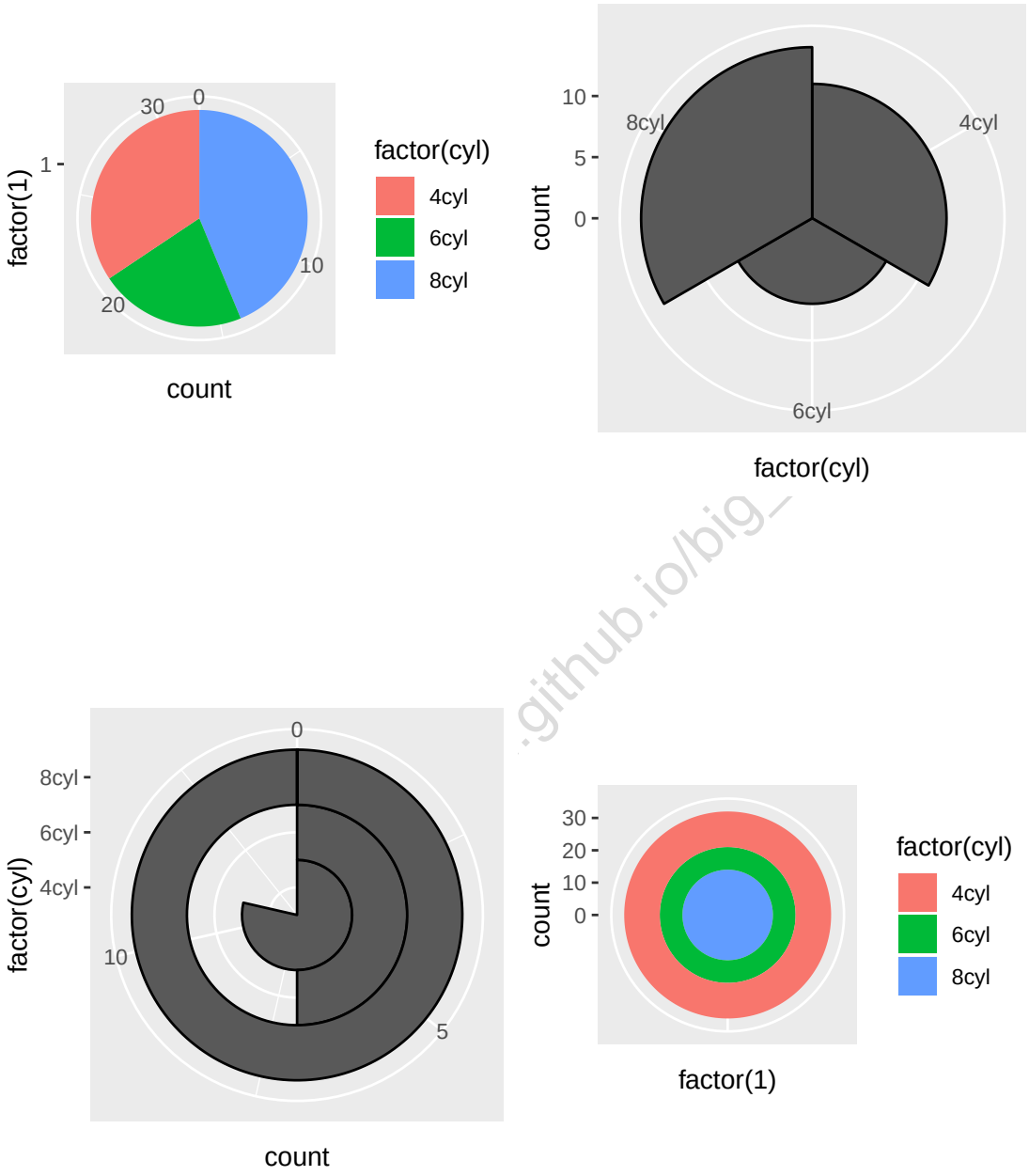

Figura 2.14: Pie chart de cyl

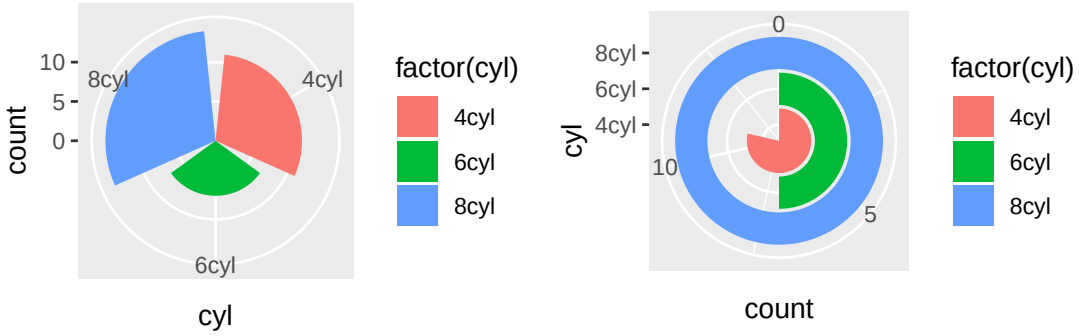

Figura 2.15: Otros gráficos con polares

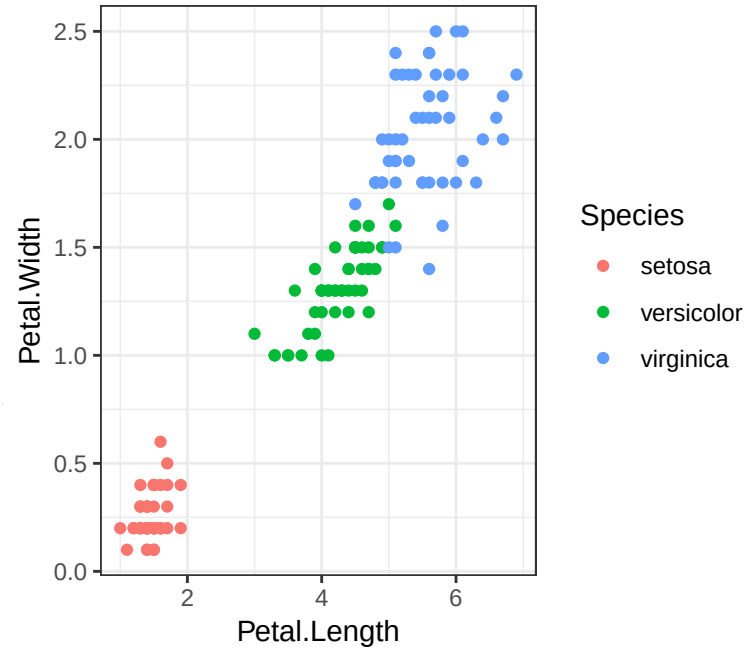

Figura 2.16: Scater plot con tema blanco-negro

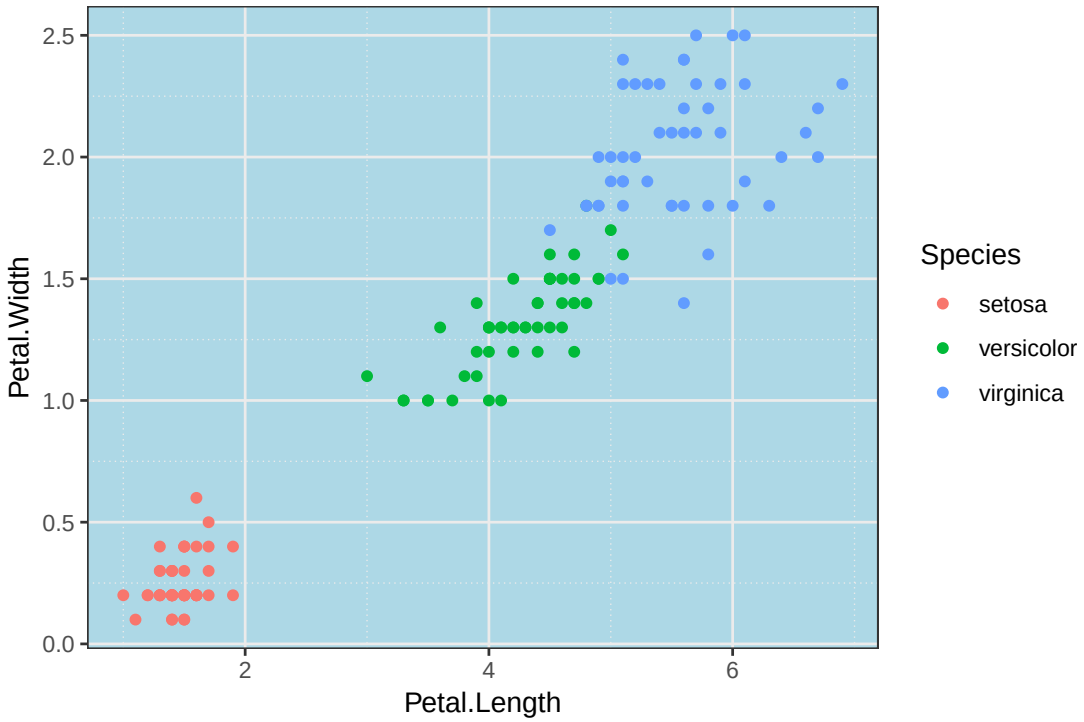

Figura 2.17: Tema blanco-negro modificado

- **element_line()** . Se utiliza para las modificaciones de las líneas.
- **element_text()** . Se utiliza ara las modificaciones de los textos.

Se puede incorporar en una nueva variable el tema que se ha modificado. La instrucción a utilizar sería la siguiente:

```
newtheme<-theme_bw()+theme(plot.title=element_text(color="darkred"))
```

Definido de esta manera el nuevo tema, se 'pueden inspeccionar sus elementos. Por ejemplo, si se quieren conocer los valores de la propiedad «panel.border», se puede usar la siguiente instrucción.

```
newtheme$panel.border

## List of 5
##  $ fill         : logi NA
##  $ colour       : chr "grey20"
##  $ size         : NULL
##  $ linetype     : NULL
##  $ inherit.blank: logi TRUE
##  - attr(*, "class")= chr [1:2] "element_rect" "element"
```

Si ahora quisiéramos modificar alguno de esos elementos lo haríamos de nuevo con la siguiente instrucción:

```
newtheme<-newtheme+
        theme(panel.border = element_rect(color="steelblue",size = 12))
```

Para mantener un tema que se quiera utilizar por defecto cuando se cargue de nuevo R, se deberá ejecutar lo siguiente (en este caso para mantener el theme_minimal()):

```
setHook(packageEvent("ggplot2", "onLoad"),
        function(...) ggplot2::theme_set(ggplot2::theme_minimal()))
```

Si se quieren ampliar las posibilidades que nos ofrecen los temas de ggplot2, existe un paquete denominado ggthemes de CRAN que muestra y pone a disposición de los usuarios muchos más temas de los cargados por defecto en ggplot2. Igualmente en este enlace, se muestran muchos temas de usos libre con los que puedes enriquecer considerablemente tus gráficos.

Para terminar este apartado, se muestra la figura 2.18 donde se puede ver distintos cambios con la función theme.

```
gg <- ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_jitter()
#Cambiamos propiedades de las líneas
p1<- gg +
  theme(axis.line = element_line(color = "black", size=1),
        axis.ticks = element_line(color = "black", size=1),
        panel.grid.major = element_line(color = "lightgrey", linetype=2),
        panel.grid.minor = element_line(color = "lightgrey", linetype=2))+
                ggtitle("Cambios propiedades de línea")
#Cambiamos propiedades del fondo, y leyenda
```

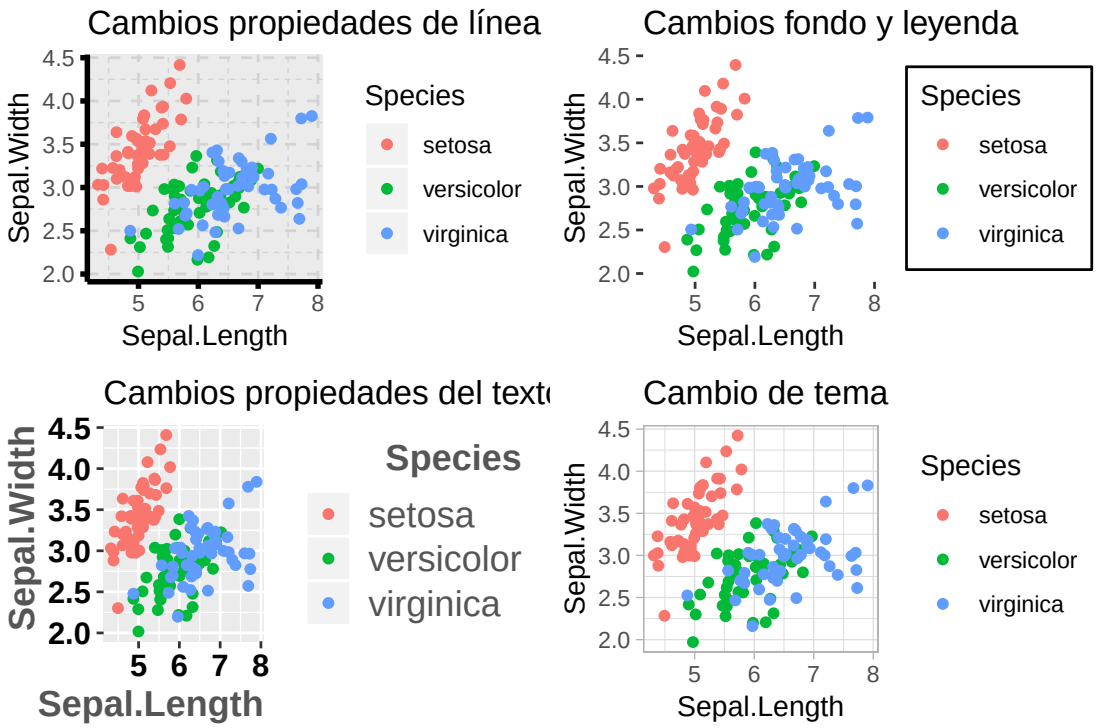

Figura 2.18: Diferentes cambios del tema

```
p2 <- gg +
  theme(panel.background = element_blank(),
        legend.background = element_rect(color="black"),
        legend.key = element_blank())+
                ggtitle("Cambios fondo y leyenda")

#Cambiamos propiedades del texto.
p3 <- gg +
  theme(axis.title = element_text(size = 14, color = "#555555", face="bold", hjust = 1),
        axis.text = element_text(color = "black", size=12, face = "bold"),
        legend.title = element_text(size = 14, color = "#555555", face="bold", hjust = 1),
        legend.text = element_text(size = 14, color = "#555555"))+
                ggtitle("Cambios propiedades del texto")
# Otro tema
p4 <- gg + theme_light()+
        ggtitle("Cambio de tema")
grid.arrange(p1,p2,p3,p4,ncol=2)
```

2.5. Matrices scatterPlots con GGally.

Con el paquete básico de R, se puede utilizar la función pair(), para generar cruces entre variables y poder dibujar la nube de puntos correspondientes. Sin embargo esa función, aunque interesante, queda un tanto limitada. Sin embargo con GGally se pueden

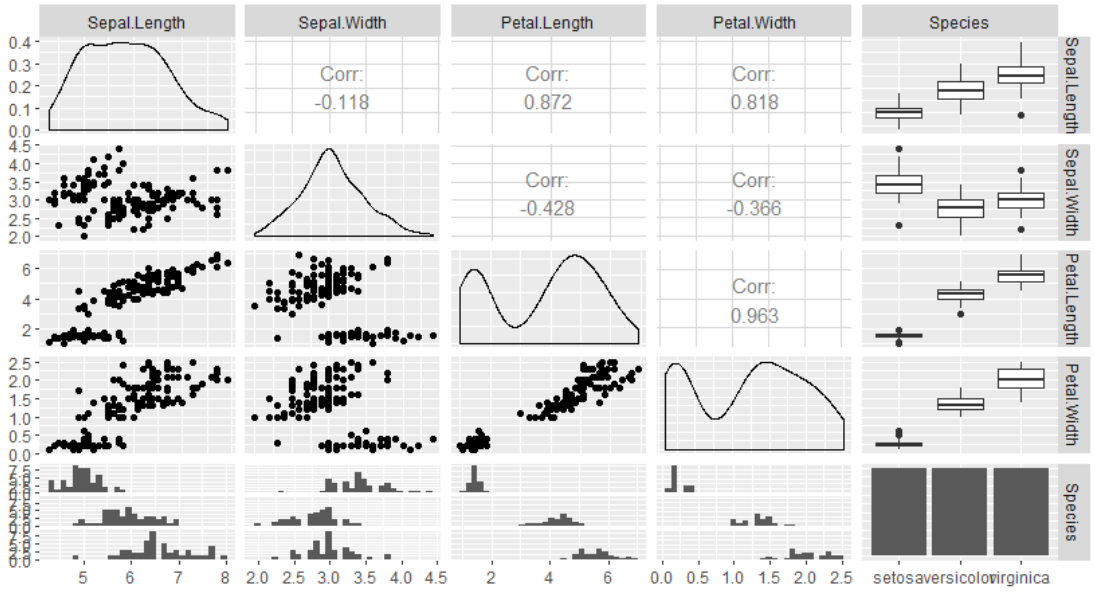

Figura 2.19: Salida con ggpairs (GGally)

obtener unos resultados mucho más estéticos y con mucha más información. La figura 2.19, muestra una salida muy estética y además con mucho más contenido que la función pairs().

```
if (!require("GGally")) install.packages("GGally"); require("GGally")
ggpairs(iris, color='Species', alpha=0.4)
```

2.6. ggmat.

Como se ha visto en los apartados anteriores, con ggplot se pueden construir muchos gráficos con contenido estadístico y con muchas opciones para adaptarlo a las necesidades de cada uno. Pero además está potencialidad gráfica también puede ser utilizada para los datos georreferenciados, es decir para la implementación de información estadística basada en mapas. Básicamente de lo que se trata es añadir gráficos ya conocidos, a una capa cartográfica adicional. Para conseguir esto se apoya en la utilización de recursos disponibles en la web a través de APs (googlemaps, openstreetmap, etc.).

Existen varios proveedores que proporcionan APIs de geolocalización - Google Maps ("google"), OpenStreetMap ("osm"), Stamen Maps ("stamen"), o CloudMade maps ("cloudmade") -. Uno de ellos es Google: dado el nombre más o menos normalizado de un lugar, la API de Google devuelve sus coordenadas. Este servicio tiene una versión gratuita que permite realizar un determinado número de consultas diarias (2500 actualmente); para usos más intensivos, es necesario adquirir una licencia. Esta situación era la anterior al 11 de junio de 2018, ya que se puede ver en esta dirección : «A partir del 11

de junio del 2018, necesitarás una clave de API válida y una cuenta de facturación para acceder a nuestras API. Cuando habilites la facturación, recibirás un crédito mensual gratuito de 200 $ para usar en Maps, Routes o Places. Actualmente, la mayoría de los millones de usuarios que utilizan nuestras API pueden seguir usando Google Maps Platform de forma gratuita con este crédito. Las cuentas de facturación nos ayudan a conocer mejor las necesidades de nuestros desarrolladores y permiten que tu negocio crezca sin problemas.».

Sin embargo cuando se va a dar de alta en ese servicio lo que se pide es el número de una tarjeta de crédito, lo que al menos a mi me hizo desistir de ese empeño y buscar otras alternativas como OpenStreetMap que también permite representar datos georreferenciados con la ayuda de ggplot.

No obstante, lo que sí se puede utilizar de forma gratuita (al menos cuando se está escribiendo este trabajo) es la coordenadas de longitud y latitud de los enclaves geográficos a través de su literal. Para conseguir esto se utiliza la función geocode(), de la siguiente manera:

```
if (!require("ggmap")) install.packages("ggmap"); require("ggmap")

## Loading required package: ggmap

punto<- geocode("Calle Santiago, 13, valladolid,spain", source = "dsk")

## Information from URL : http://www.datasciencetoolkit.org/maps/api/geocode/json?address=Calle%20Santiago,%2013,%20val

punto

##        lon      lat
## 1 -4.72372 41.65518
```

NOTA: observar que se ha puesto «source="dsk"» (Data Science Toolkit), ya que la otra opción es poner «google» para obtenerlos de google maps, pero esto daría un error de que no se puede facilitar la información por no tener la api-key correspondiente.

Una vez identificadas las coordenadas, si se tuviera la api key de google maps, se podría utilizar la función «get_map» para obtener el mapa correspondiente, pero si ejecutamos esa función sin la api key, es decir de las siguiente manera.

```
map.punto <- get_map(location = as.numeric(punto),
                     source = "google",
                     scale= 2,zoom=16)
```

El mensaje que se nos va a devolver es el siguiente:

```
Error in download.file(url, destfile = tmp, quiet = !messaging, mode = "wb") :
  cannot open URL 'http://maps.googleapis.com/maps/api/staticmap?center=41.65518,-4.7
zoom=16&size=640x640&scale=2&maptype=terrain&sensor=false'
In addition: Warning message:
In download.file(url, destfile = tmp, quiet = !messaging, mode = "wb") :
  cannot open URL 'http://maps.googleapis.com/maps/api/staticmap?center=41.65518,-4.7
zoom=16&size=640x640&scale=2&maptype=terrain&sensor=false': HTTP status was '403 Forb
```

Es decir que no se puede acceder a la petición solicitada, y el motivo es a falta de esa api key que debe facilitarse en un parámetro de la función. No obstante, si se mira la documentación de esta función, en la misma se dice que el parámetro «source» sirve para definir el servicio Web del que se va a solicitar la información. En base a eso, se ha intentado cambiar el valor de ese parámetro por el correspondiente a «openstreetMap» y el error devuelto es muy similar.

Dadas las limitaciones que se tenía para trabajar con esa función, se ha buscado otra alternativa y la solución se ha encontrado en el paquete «OpentreetMap» que trabaja con mapas sacados de Open Street Map y cuya documentación de puede ver en este enlace.

Lo que se ha hecho es ir a un mapa de Valladolid, en concreto al que se encuentra en este enlace, después en la parte superior izquierda, hay un botón con el epígrafe «exportar». Se hace click en él y nos aparece una opción para poder elegir la zona a exportar. También nos aparecen las coordenadas que debemos utilizar para indicar en R la zona a descargar. Con esa información se ejecuta la siguiente instrucción y se obtiene el mapa que se puede ver en la figura 2.20.

```
if (!require("maps")) install.packages("maps"); require("maps")

## Loading required package: maps

if (!require("OpenStreetMap")) install.packages("OpenStreetMap"); require("OpenStreetMap")

## Loading required package: OpenStreetMap

mp <- openmap(c(41.6504,-4.7482), ##Hay que dar lat y lon en este orden
              c(41.6342,-4.7082),type="bing",zoom = NULL)

plot(mp)
```

De esta manera hemos conseguido obtener un plano de una zona de la ciudad de Valladolid en España. El tipo de mapa que se ha bajado se ha obtenido mediante el parámetro «type="bing"», pero hay muchos más tipos. Para consultarlos visitar este enlace. Esto es lo que se hace y se asigna a la variable «mp.pro».

Como ya hemos visto antes, existe la función «geocode()» del paquete ggmap que sirve para georreferenciar un determinado lugar. Para poder utilizar una función similar pero usando la herramienta de georreferenciación de OpenStreetMap que se denomina «nominatim» y que lo puedes visitar en este enlace, se ha cargado el paquete «tmaptools» para obtener las coordenadas de la dirección que queremos colocar en el mapa, para ello se ha utilizado la función «geocode_OSM()».

```
#Cambio la projection
mp.pro<-openproj(mp, projection = "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
#Paquete para georreferenciar con OSM (herramienta Nominatim )
if (!require("tmaptools")) install.packages("tmaptools"); require("tmaptools")

## Loading required package: tmaptools

punto2<-geocode_OSM("paseo de zorrilla 13 valladolid spain")
print(punto2)
```

Figura 2.20: Mapa zona Valladolid con Open Street Map

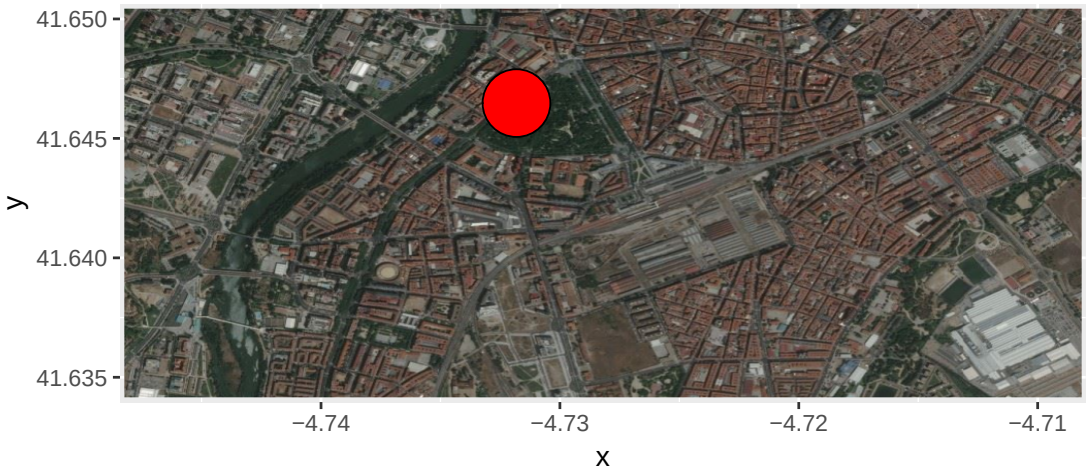

Figura 2.21: Colocación del punto geolocalizado

```
## $query
## [1] "paseo de zorrilla 13 valladolid spain"
##
## $coords
##         x         y
## -4.731817 41.646477
##
## $bbox
##      xmin      ymin      xmax      ymax
## -4.733525 41.645113 -4.730877 41.647215
```

Como puede verse la variable «punto2», contiene las coordenadas del punto en la propiedad «coords». Se utilizan esas coordenadas para colocarlas en el mapa mediante una capa de ggplot2. Observar que se utiliza la función «autplot()» del paquete «OpenStreetMap», con la finalidad de obtener un objeto compatible con ggplot2.

```
mp.punto<-autoplot(mp.pro)+geom_point(data=punto,aes(x=punto2$coords[1],
                                      y=punto2$coords[2]),size=12,shape=21,
                                      fill='red')
mp.punto
```

En la figura 2.21 se puede ver el mapa de la zona de Valladolid, junto con la representación del punto que se ha geolocalizado anteriormente.

```
LAT1 =  30 ; LAT2 = 50
LON1 = -10 ; LON2 = 10

map <- openmap(c(LAT2,LON1), c(LAT1,LON2), zoom = NULL,
               type = c("osm", "stamen-toner", "stamen-terrain","stamen-watercolor", "esri","esri-topo")[6],
               mergeTiles = TRUE)
## Se cambia la projection
## OSM CRS :: "+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +

map.latlon <- openproj(map, projection = "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")

bias.dtf <-  data.frame(LON  = runif(10, LON1, LON2),
                        LAT  = runif(10, LAT1, LAT2),
                        BIAS = runif(10, 5, 10),
                        ID   = paste("Pt.",sprintf("%02d",1:10)))
mytheme <- theme(plot.title = element_text(face = "bold",size = rel(1.2), hjust = 0.5),
                 panel.background = element_rect(colour = NA),
                 plot.background = element_rect(colour = NA),
                 axis.title = element_text(face = "bold",size = rel(1)),
                 axis.title.y = element_text(angle=90,vjust =2),
                 axis.title.x = element_text(vjust = -0.2))

OSMap <- autoplot(map.latlon)  +
  labs(title = "Plot over OpenStreetMap", subtitle = "Bias [%]",x = "Longitude", y="Latitude")+
  geom_label(data=bias.dtf, aes(x=LON ,y=LAT, label=ID, fontface=7), hjust=1, vjust=0, size=4) +
  geom_point(data=bias.dtf, aes(x=LON, y=LAT, fill=BIAS), size=4, shape=21) +
  scale_fill_gradientn(colours = rainbow(10)) + mytheme

OSMap
```

Para finalizar, en la figura 2.22 se muestra una representación más lograda que en el ejemplo anterior, sobre la colocación de determinados puntos en el mapa. Como el lector puede comprender ahora, se pueden conseguir representaciones gráficas sobre mapas muy potentes, como por ejemplo mapas de densidad, diagramas de barras, o circulaes, localizados en los puntos de mapa que se deseen,etc.

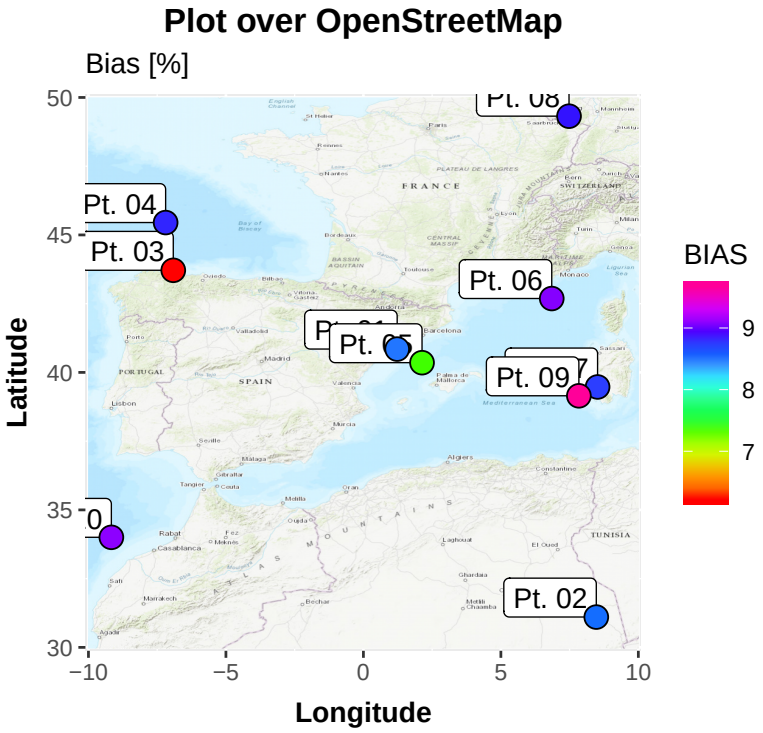

Figura 2.22: Puntos representados sobre un mapa

Bibliografía

[https://mrccsc.github.io/r-intermediate/ggplot2.html]

[https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf]

Índice alfabético

Índice de figuras