Gráficos en ggplot2

Table of contents

Este material es parte del curso **Introducción a R *tidyverse* del Instituto Nacional de Epidemiología "Dr. Juan H. Jara" - ANLIS**

¿Qué es `ggplot2`?

Es un paquete incluido en `tidyverse` que permite generar gráficos de alta calidad y distintos niveles de complejidad usando una gramática sencilla e intuitiva. El nombre `ggplot2` viene precisamente de *grammar of graphics* (gramática de los gráficos), donde se sigue una serie de pasos ordenados para generar el gráfico.

Creación de gráficos

La función principal del paquete se llama `ggplot()` y es la primera que deberemos llamar para iniciar un gráfico. Esto podemos hacerlo con la sintaxis tradicional de `R base` o mediante el formato `tidyverse`:

```
## Carga set de datos ejemplo
data(iris)

## Inicio gráfico: formato de R base
ggplot(data = iris)
```



```
# Inicio gráfico: formato tidyverse
iris %>%
  ggplot()
```

Como pueden observar, cualquiera de las dos opciones nos va a mostrar un recuadro vacío, esto es porque aún no hemos asignado nada a los ejes `x` e `y` ni definimos que tipo de gráfico queremos obtener.

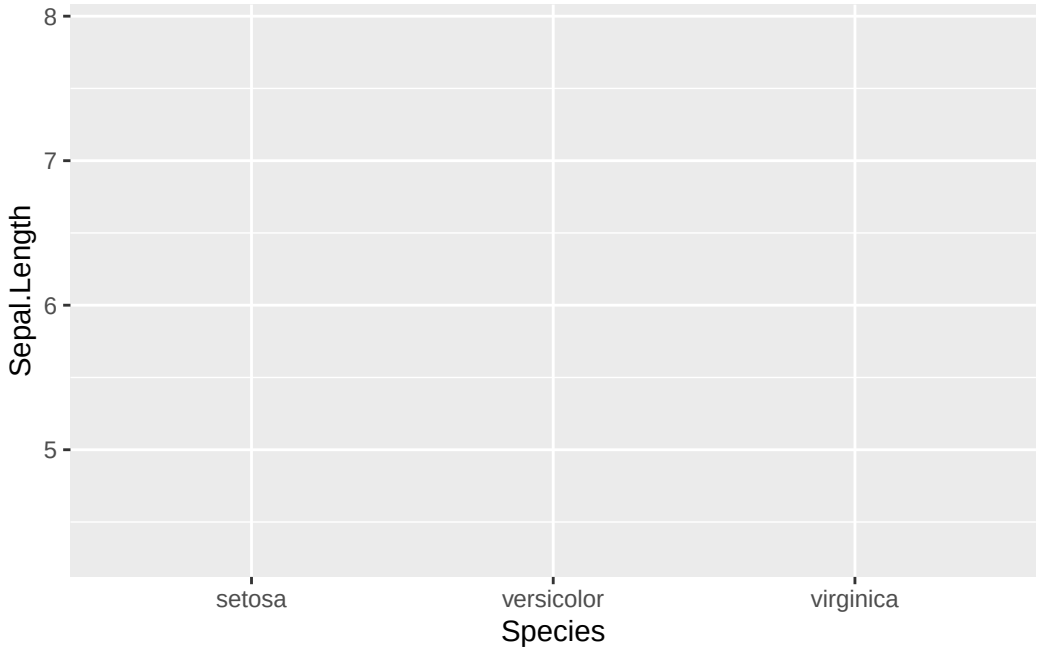
Mapeo estético

Un segundo paso entonces es hacer el "mapeo estético" de los ejes mediante el argumento `mapping =` y la función `aes()`

```
## Consulto nombres de variables del dataset
names(iris)
```

```
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
## Inicio el gráfico
iris %>%
  # Defino ejes x e y
  ggplot(mapping = aes(x = Species, y = Sepal.Length))
```



Al ejecutar el comando vemos que, si bien nos sigue generando un recuadro vacío, ahora en este aparecen los nombres de los ejes y sus escalas.

Geomas

El siguiente paso es indicarle a `ggplot` de que forma queremos que aparezcan los datos utilizando la familia de comandos `geom_xxx()`. Los geoms, geomas o geometrías son "apilables" entre sí, pudiendo poner uno o varios en el mismo gráfico y aparecerá por encima el último que llamemos.

Algunos geomas comunes son:

`geom_histogram()` - histograma de frecuencias

`geom_point()` - gráfico de puntos

`geom_line()` - gráfico de líneas

`geom_bar()` y `geom_col()`- gráficos de barras

`geom_boxplot()` - gráfico de cajas (*boxplot*)

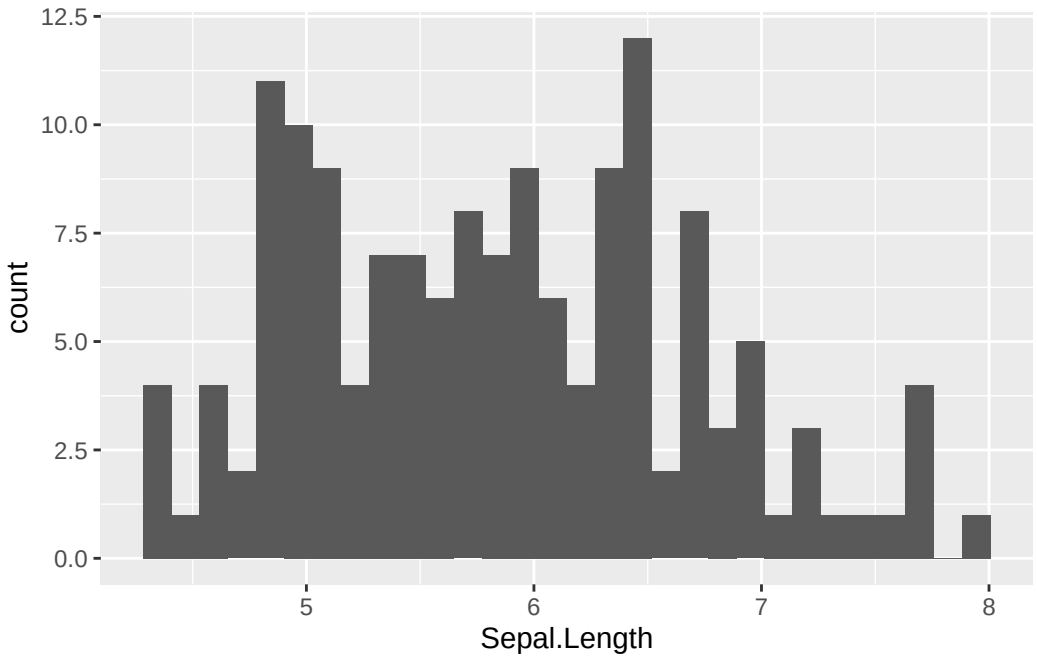
`geom_violin()` - gráfico de violines

`geom_smooth()` - curvas suavizadas y líneas de regresión

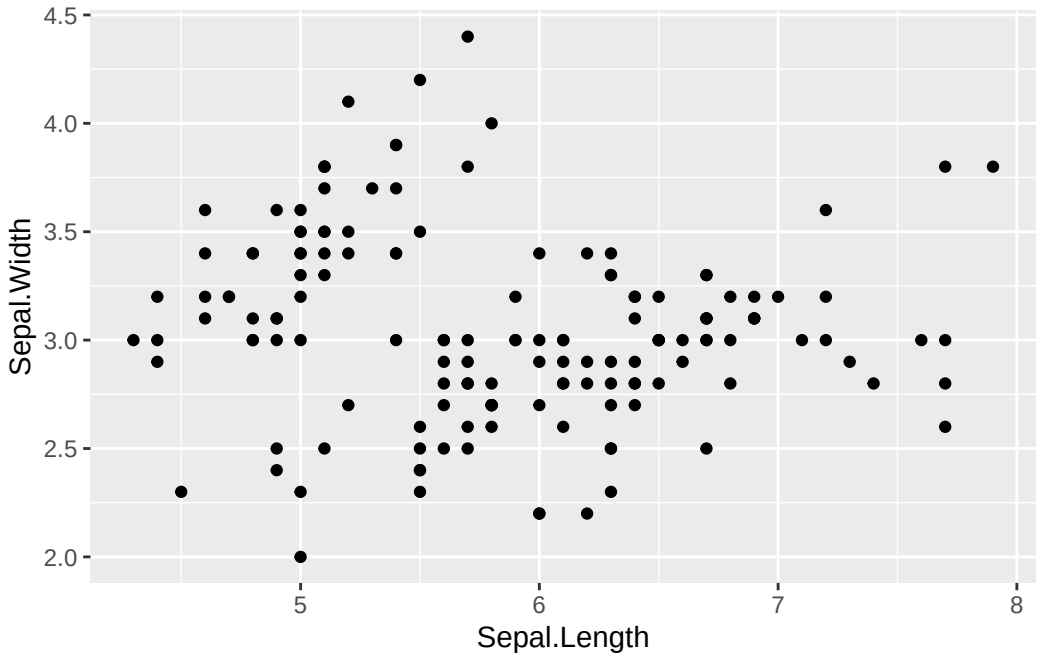
La lista completa de geoms se puede encontrar en el siguiente [LINK]

Antes de mostrar ejemplos de como generar cada tipo de gráfico, debemos tener en cuenta que, si bien es parte de `tidyverse`, el paquete `ggplot2` tiene su propio operador pipe para unir partes del gráfico y es el signo `+`

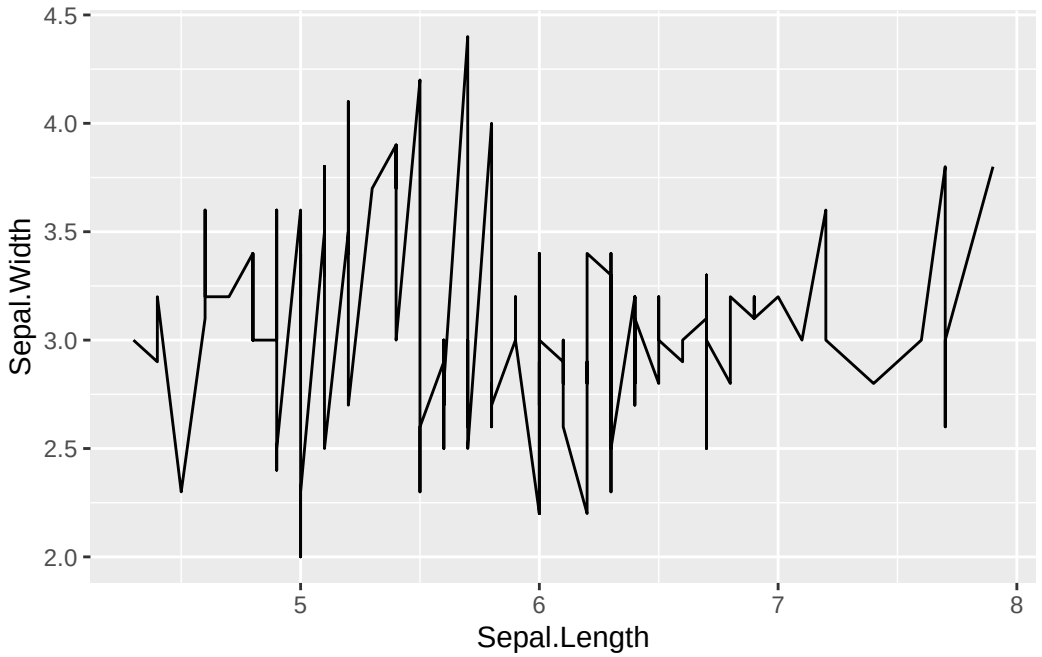
```
## Histograma de frecuencias
iris %>%
  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length)) +
  # especifico geoma
  geom_histogram()
```



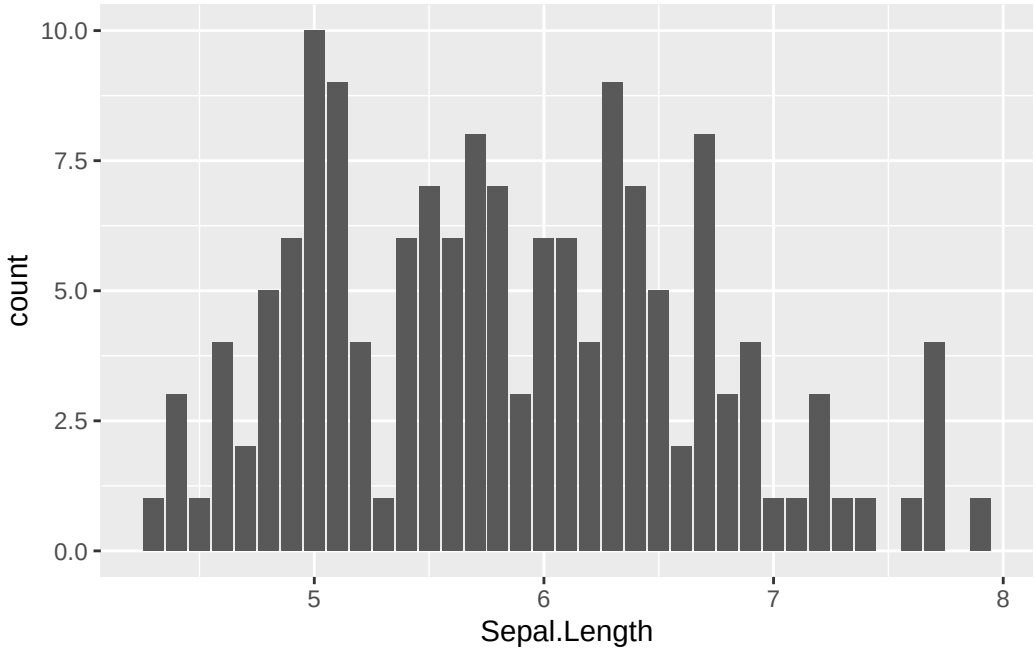

```
## Gráfico de puntos
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +

  # especifico geoma
  geom_point()
```



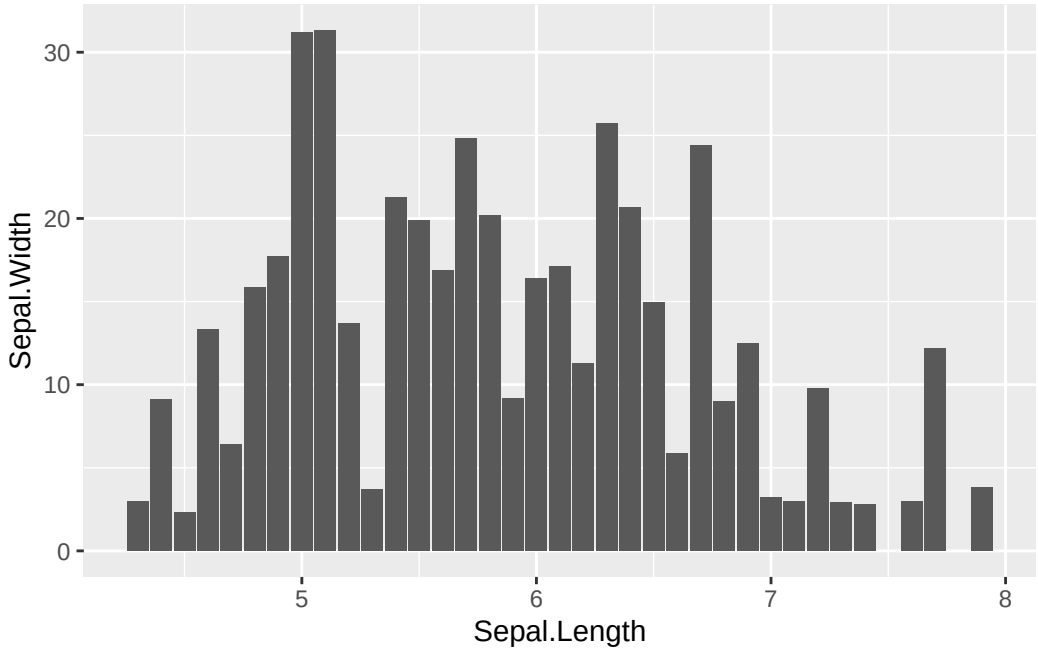

```
## Gráfico de líneas
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +

  # especifico geoma
  geom_line()
```



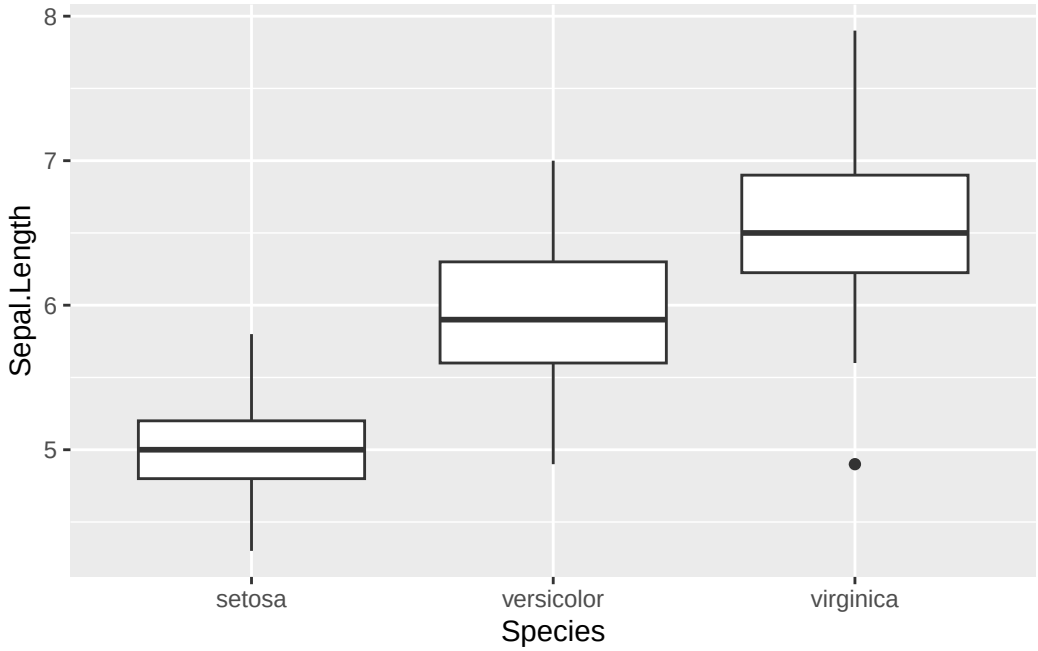

```
## Gráfico de barras (solo eje X)
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length)) +

  # especifico geoma
  geom_bar()
```



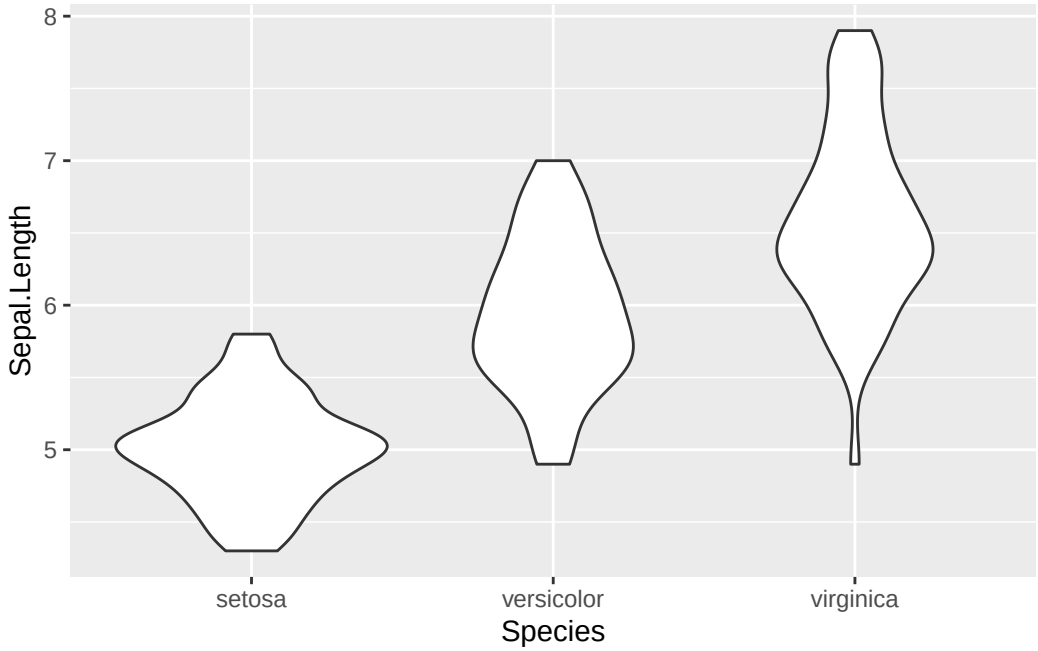

```
## Gráfico de barras (ejes X e Y)
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +

  # especifico geoma
  geom_col()
```



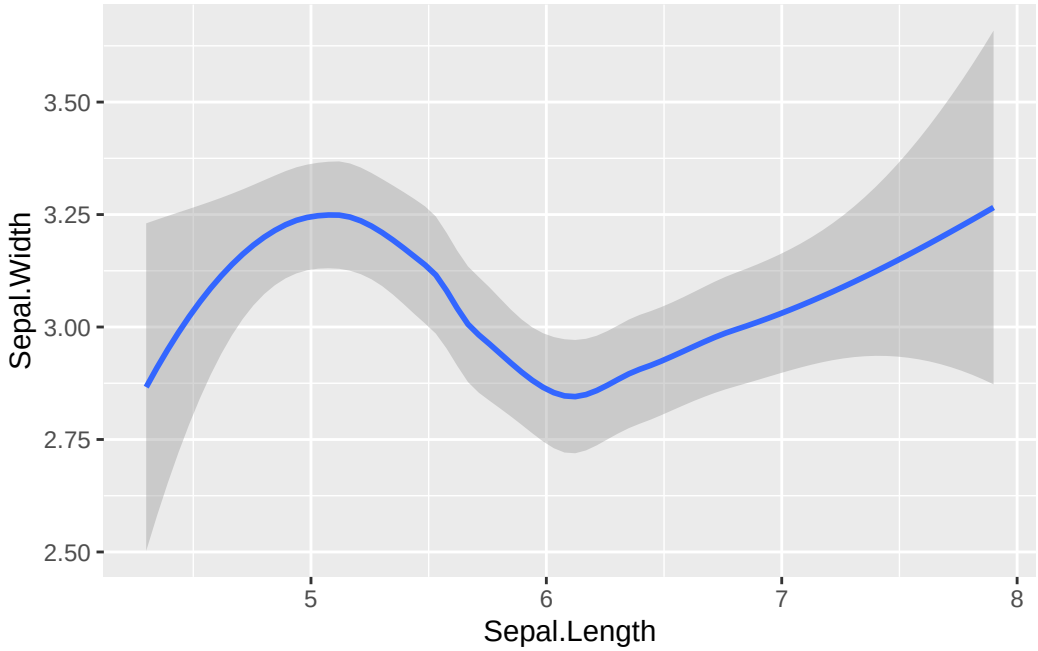

```
## Boxplot (eje X categórico y eje Y numérico)
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Species, y = Sepal.Length)) +

  # especifico geoma
  geom_boxplot()
```



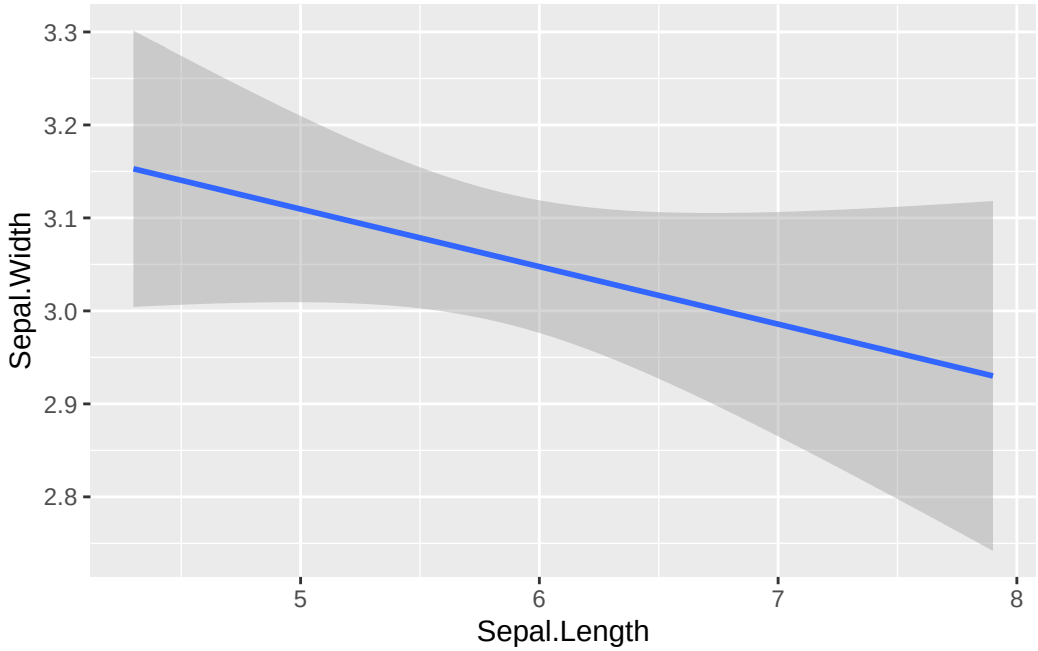

```
## Gráfico de violin
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Species, y = Sepal.Length)) +

  # especifico geoma
  geom_violin()
```




```
## Curvas suavizadas
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +

  # especifico geoma
  geom_smooth()
```

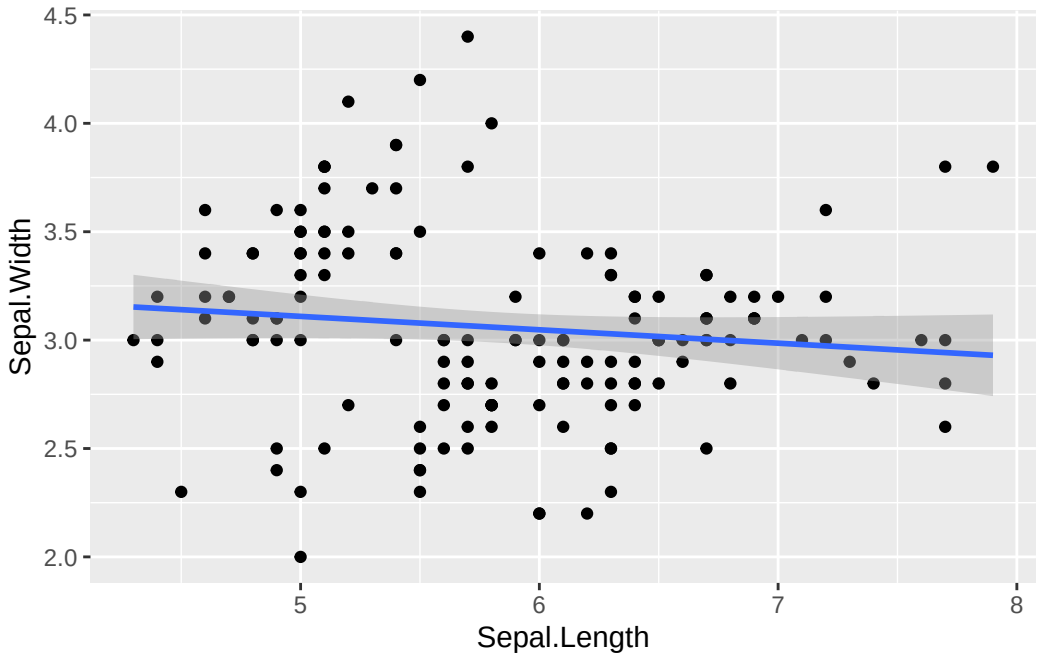



```
## Líneas de regresión (argumento method = "lm")
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +

  # especifico geoma
  geom_smooth(method = "lm")
```



Como dijimos anteriormente, se pueden colocar varios geomas en un mismo gráfico:

```
## Gráfico de puntos con línea de suavizado
iris %>%
  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +

  # añado geoma de gráfico de puntos
  geom_point() +

  # añado geoma de recta de regresión linear
  geom_smooth(method = "lm")
```



Elementos estéticos

Una vez definido nuestro gráfico podemos editar aspectos visuales con los argumentos:

`color =` - Define color de línea

`fill =` - Define color de relleno

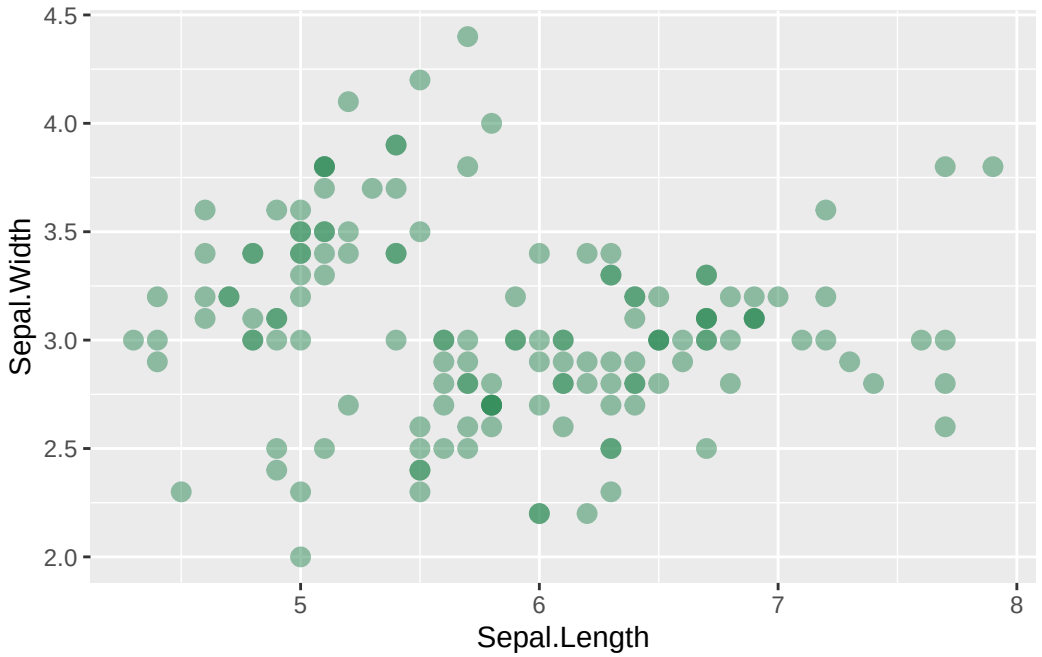
`alpha =` - Transparencia (valores entre 0 y 1)

`size =` - Tamaño de puntos y ancho de líneas

`linetype =` - Formato de líneas

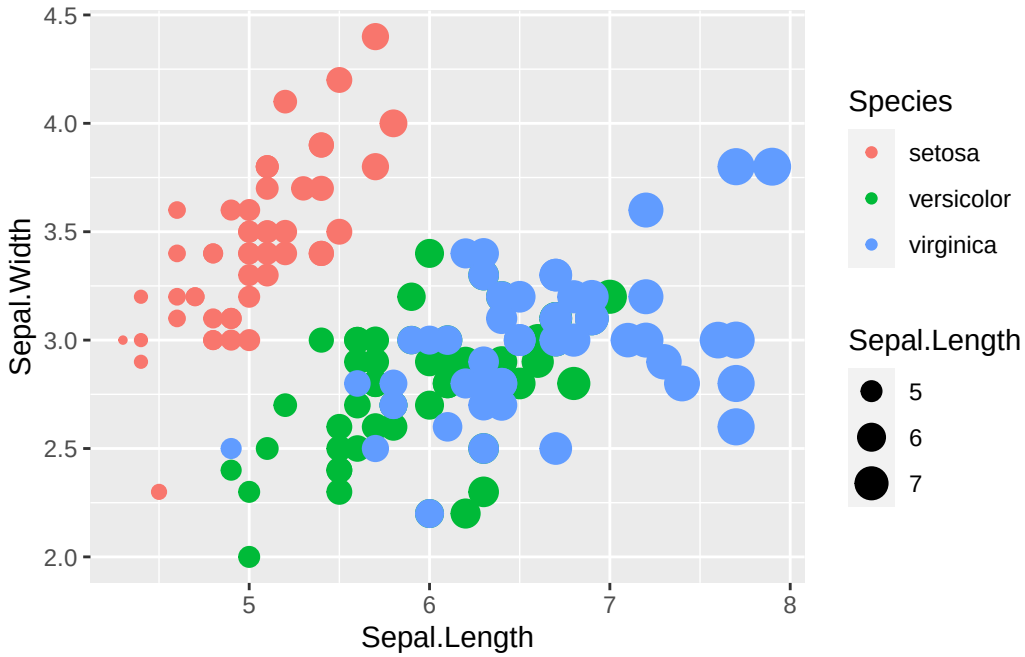
Si colocamos algunos de estos argumentos dentro de `geom_xxx()` vamos a obtener opciones estéticas fijas, que aplicarán solo a ese geoma y serán iguales para todos sus elementos. Este tipo de configuración no genera leyendas en el gráfico:

```
## Gráfico de puntos con estéticas fijas
iris %>%
  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  #especifico geoma
  geom_point(color = "seagreen", size = 3, alpha = .5)
```



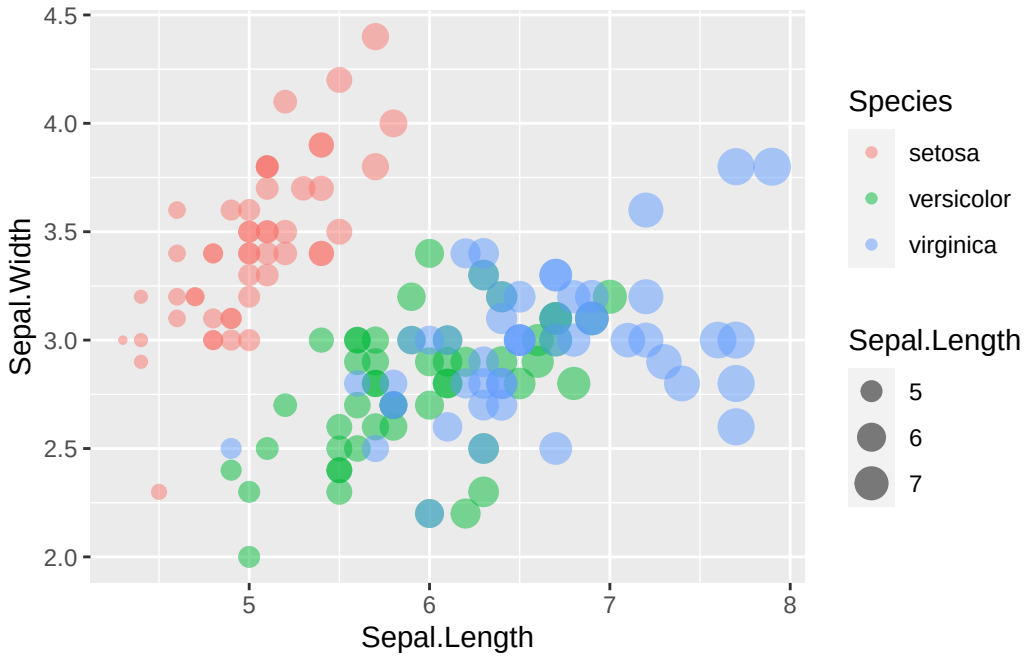
En cambio, si queremos estéticas dinámicas que generen elementos de leyenda, los argumentos estéticos irán dentro de `mapping = aes()` al inicio del gráfico y les asignaremos una variable del dataset en lugar de un color o tamaño estáticos.

```
## Gráfico de puntos con estéticas dinámicas
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species, size = Sepal.Length)) +

  # especifico geoma
  geom_point()
```



Se pueden combinar estéticas fijas y dinámicas en un mismo gráfico

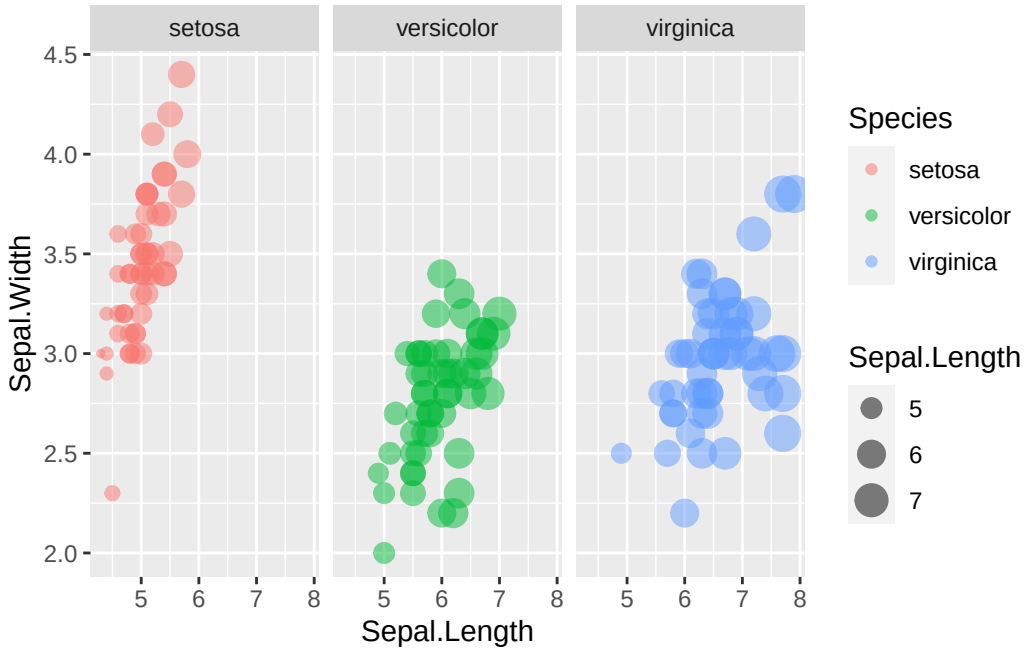
```
## Gráfico de puntos con estéticas fijas y dinámicas
iris %>%

  # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species, size = Sepal.Length)) +

   # especifico geoma
  geom_point(alpha = .5)
```



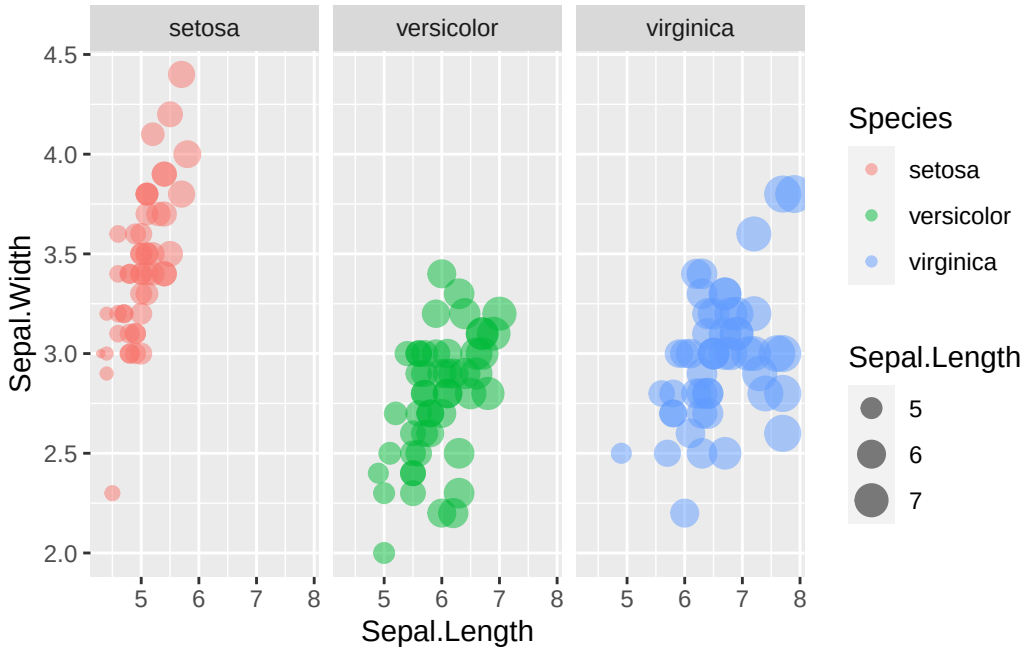
Facets

También podemos subdividir el gráfico en paneles o minigráficos en función de una o más variables categóricas usando los comandos `facet_wrap(~variable)` o `facet_grid()`:

```
## Paneles con facet_wrap()
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species, size = Sepal.Length)) +
  # especifico geoma
  geom_point(alpha = .5) +

  # minigráficos por especie
  facet_wrap(~ Species)
```



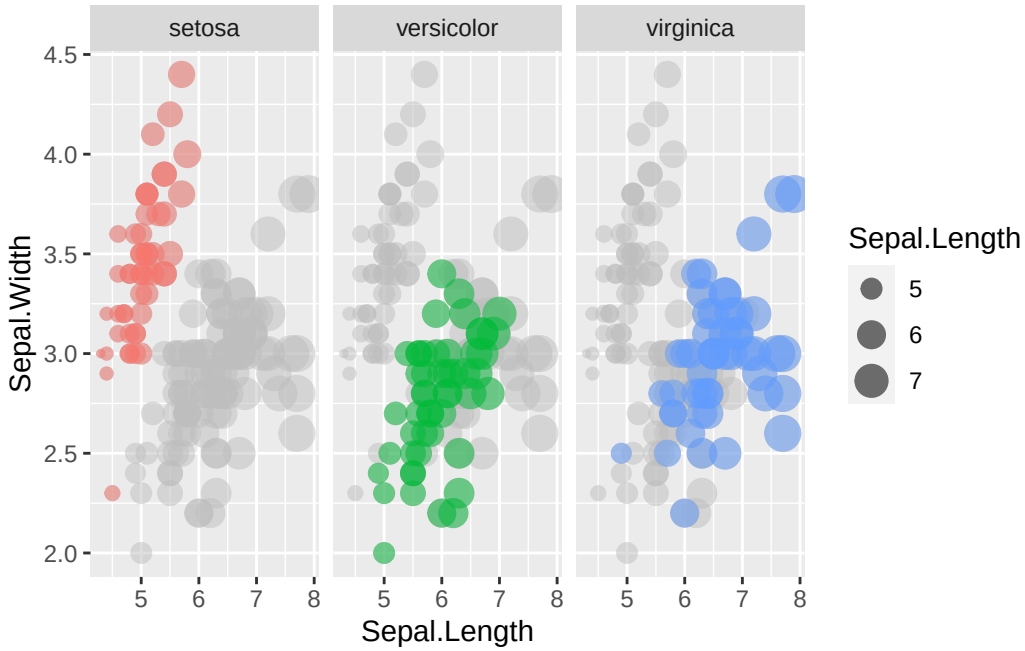

```
## Paneles con facet_grid()
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species, size = Sepal.Length)) +
  # especifico geoma
  geom_point(alpha = .5) +

  # minigráficos por especie
  facet_grid(~ Species)
```



Extensiones de `ggplot2`

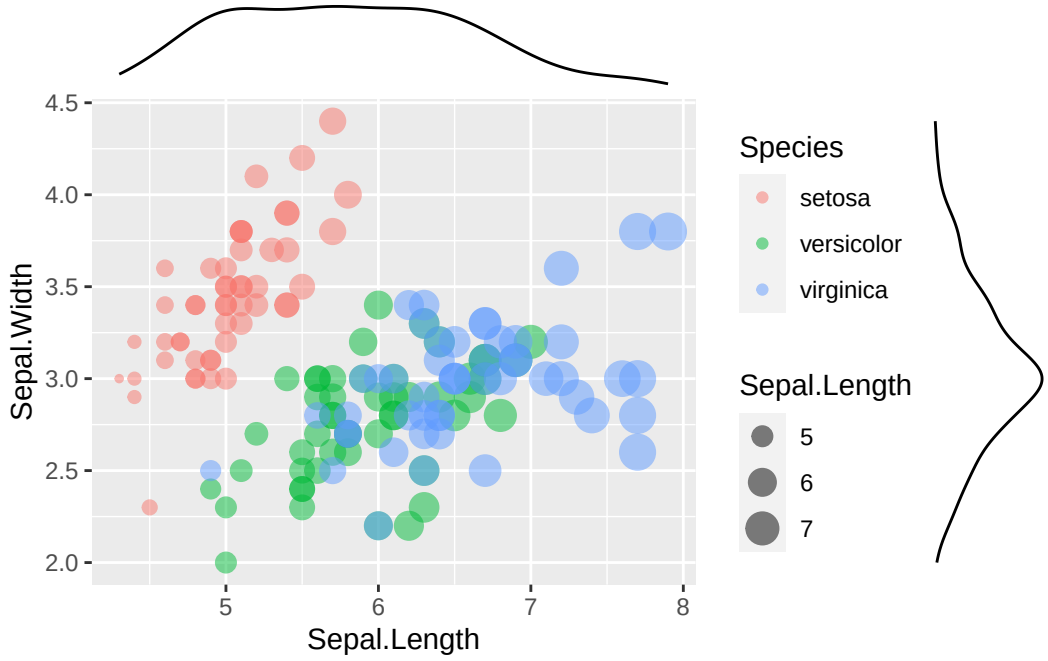
Existen paquetes complementarios a `ggplot2` que permiten expandir sus funciones. Por ejemplo `gghighlight` permite resaltar un conjunto específico de datos:

```
library(gghighlight)

## Paneles con facet_wrap() y resaltado
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species, size = Sepal.Length)) +
  # especifico geoma
  geom_point(alpha = .5) +

  # minigráficos por especie
  facet_wrap(~ Species) +

  # añade resaltado con gghighligth
  gghighlight()
```



El paquete `ggExtra` permite añadir gráficos de distribución marginales mediante la función `ggmarginal()`. Para usarl, primero debo asignar el ggplot a un objeto:

```
library(ggExtra)

## Asigno gráfico a un objeto
g1 <- iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species, size = Sepal.Length)) +

  # especifico geoma
  geom_point(alpha = .5)

## Distribución marginal
ggMarginal(g1)
```

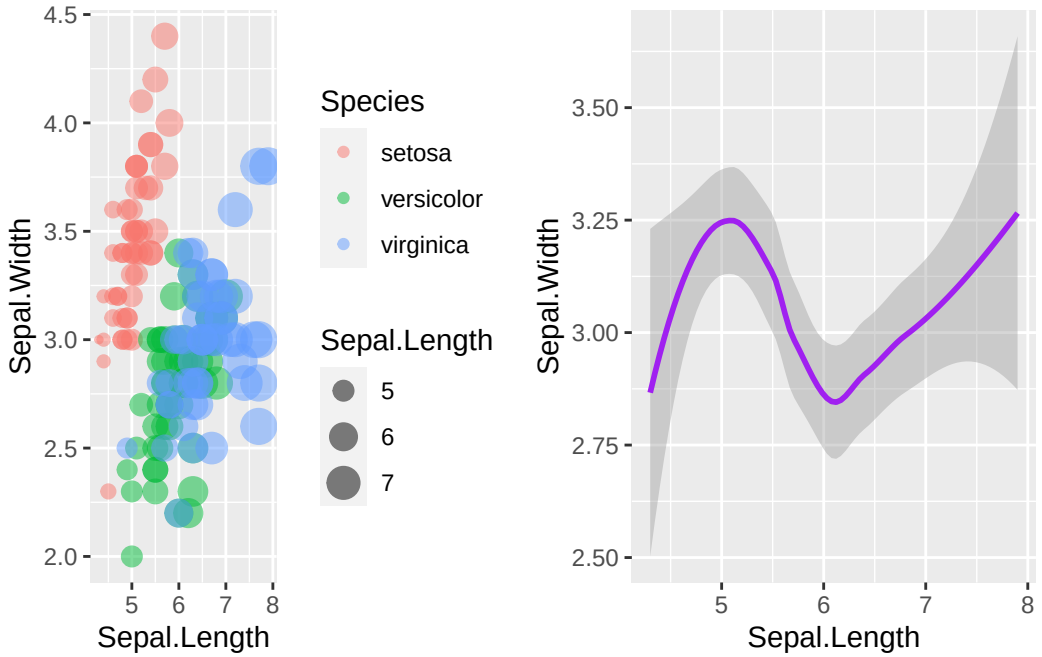


O podemos jugar con las opciones del gráfico marginal ejecutando el comando `ggMarginalGadget()` que nos abrirá una ventana emergente donde podemos previsualizar como se vería el gráfico con distintas opciones de configuración.

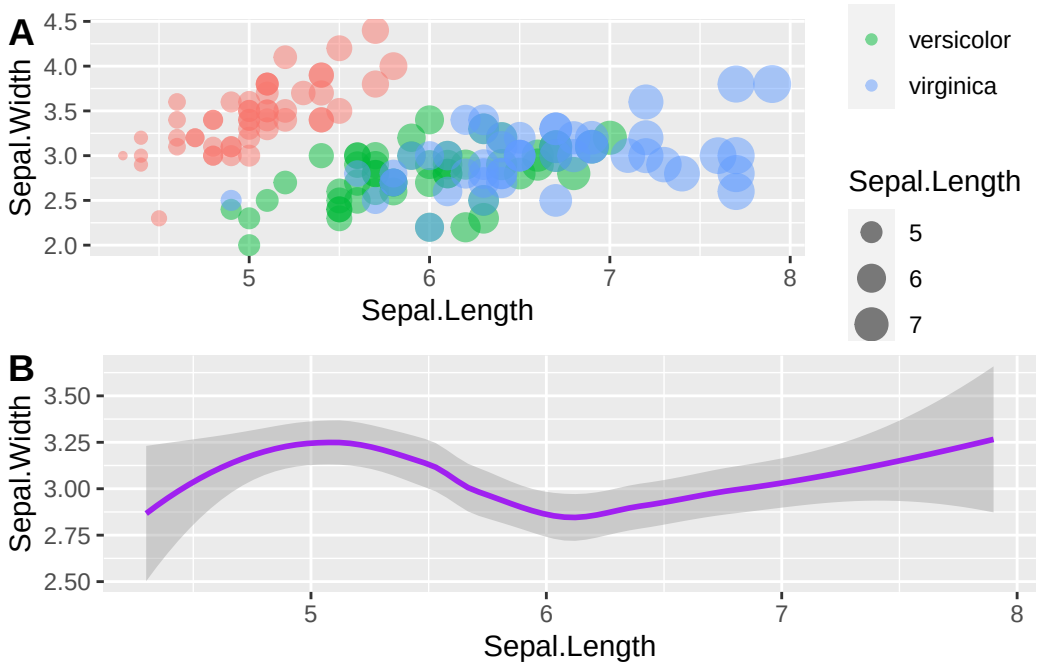
Otro paquete útil es `cowplot`, que permite unir distintos gráficos en paneles con la función `plot_grid()`. La misma se diferencia de los facets ya que en lugar de subdividir un gráfico según niveles de alguna variable, une gráficos de distintos tipos y/o tablas. Los gráficos a unir se pueden asignar a objetos (recomendado si son más de dos o tienen muchas opciones de configuración) o generarse adentro del mismo `plot_grid()`

```
library(cowplot)

## Asigno gráfico 1 a un objeto
g1 <- iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species, size = Sepal.Length)) +

  # especifico geoma
  geom_point(alpha = .5)

## Asigno gráfico 2 a un objeto
g2 <- iris %>%
```

```
   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +

  # especifico geoma
  geom_smooth(color = "purple")

## Combino gráficos
plot_grid(g1, g2)
```




```
## Especifico número de columnas y filas y añado etiquetas
plot_grid(g1, g2, nrow = 2, labels = "AUTO")
```



Opciones avanzadas de formato

Hasta aquí vimos como generar gráficos, colocar la geometría, subdividir en paneles y algunos paquetes que permiten realizar tareas extra de gran utilidad. Todo esto nos sirve para generar versiones preliminares de nuestros gráficos, pero si quisiéramos publicar resultados y/o compartir con nuestros colegas, deberíamos mejorarles el aspecto para que sean más llamativos e informativos.

Esto lo podemos hacer con comandos de `ggplot2` que permiten editar los ejes X e Y, cambiar las paletas de colores predefinidas, cambiar las etiquetas de ejes y agregar títulos, subtítulos o captions y modificar el tema predeterminado, entre otras.

Modificar escalas de los ejes

Para modificar los ejes X e Y del gráfico usaremos las funciones de la familia `scale_x_xxx()` y `scale_y_xxx()`, respectivamente. Las mismas pueden tomar los valores:

`scale_x__discrete()` y `scale_y_discrete()` - variables numéricas discretas o variables categóricas

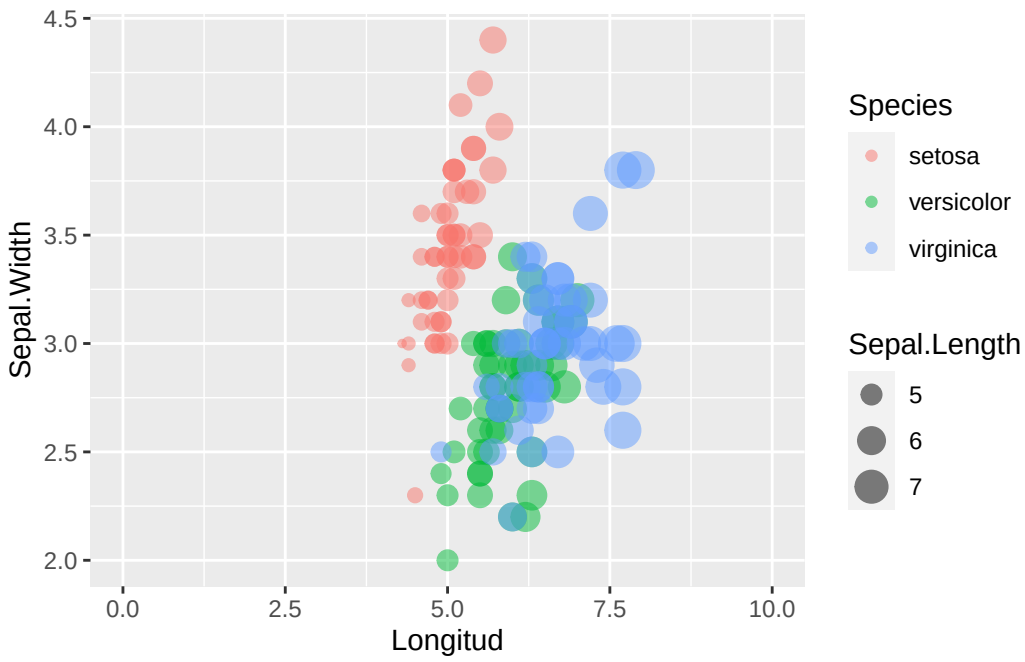
`scale_x_date()` y `scale_y_date()` - variables de tipo fecha

`scale_x_datetime()` y `scale_y_datetime()` - variables de tipo fecha y hora

La lista completa puede consultarse en el siguiente [LINK]

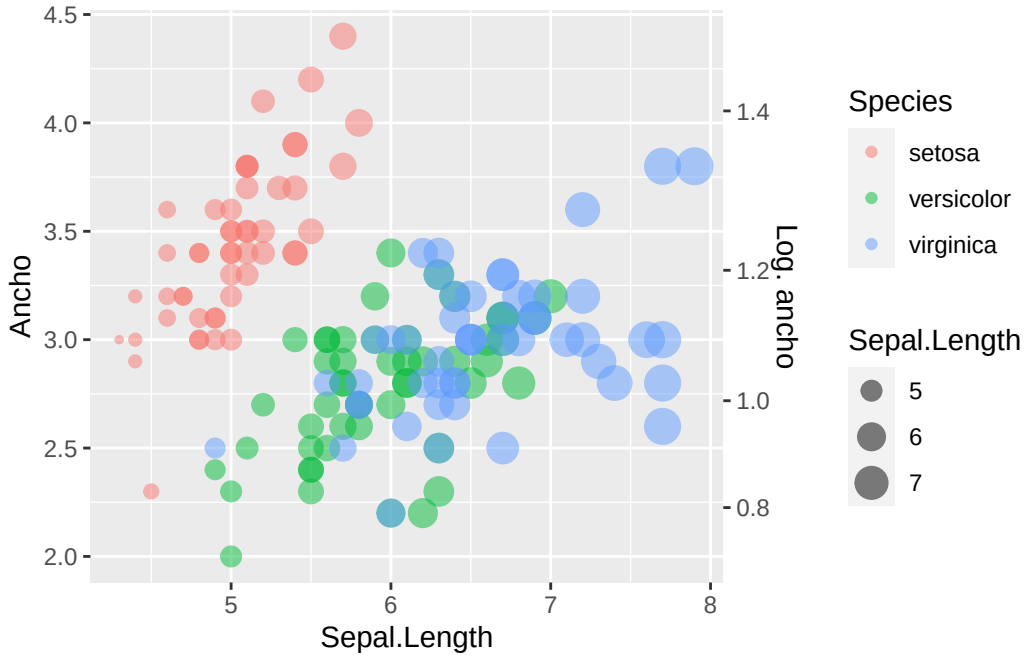
Veremos algunos ejemplos usando como base el objeto `g1` que generamos anteriormente:

```
## Modifico eje X
g1 +
  scale_x_continuous(
    # cambia nombre del eje
    name = "Longitud",

    # cambia rango
    limits = c(0, 10),

    # cambia puntos de corte
    breaks = c(0, 2.5, 5, 7.5, 10))
```




```
## Modifico eje Y
g1 +
  scale_y_continuous(
    # cambia nombre
    name = "Ancho",

    # añade eje Y secundario con el logaritmo del ancho
```

```
    sec.axis = sec_axis(trans = log, name = "Log. ancho")
  )
```



Hay que tener en cuenta que si intentamos asignar una escala diferente a la de la variable representada en el eje, R nos dará un mensaje de error:

```
g1 +
  scale_y_datetime()
```

```
Error in `self$trans$transform()`:
! `transform_time()` works with objects of class <POSIXct> only
```

Escalas de colores

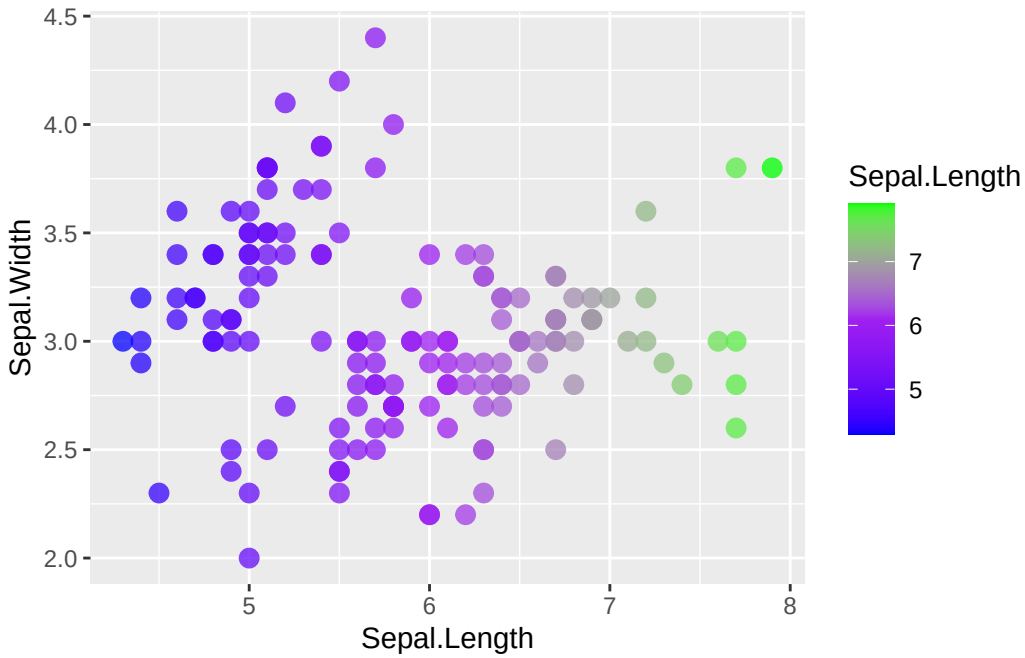
Otra familia de funciones interesantes es `scale_color_xxx()` y `scale_fill_xxx()` que nos permiten modificar las escalas de color de línea/puntos o de relleno de los gráficos ya sea definiendo manualmente una paleta o usando paletas predefinidas.

Algo que debemos tener en cuenta al seleccionar las escalas de colores que utilizaremos en nuestros gráficos, es seguir las reglas de publicación de la institución, congreso o *journal* donde pensemos compartir nuestros resultados. Algunas de estas especificaciones involucran utilizar paletas de colores que se distingan claramente al imprimir en escala de grises, así

como aquellas que sean aptas para personas con daltonismo u otras limitaciones visuales. Los paquetes complementarios `viridis` y `scico` ofrecen paletas de colores aptas para personas con daltonismo, mientras que `RColorBrewer` incluye tanto este tipo de paletas como esquemas de colores en gradientes y combinaciones predefinidas.

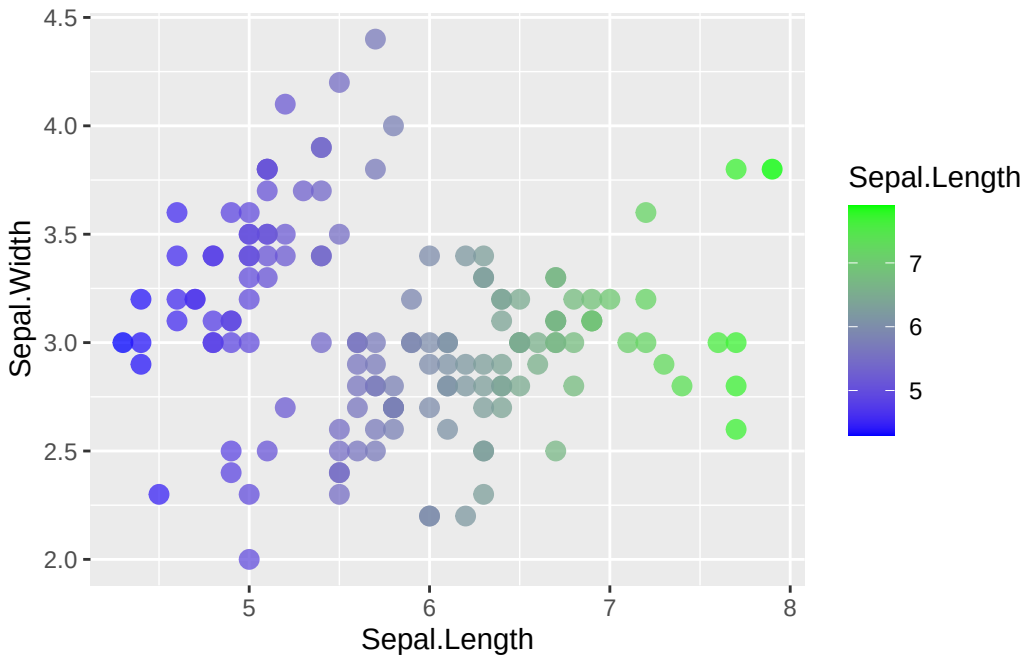
Veamos un ejemplo práctico:

```
## Escala de color: gradiente manual (variable continua)
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Sepal.Length)) +
  # especifico geoma
  geom_point(alpha = .75, size = 3) +

  # escala de colores
  scale_color_gradientn(colors = c("blue","purple","green"))
```

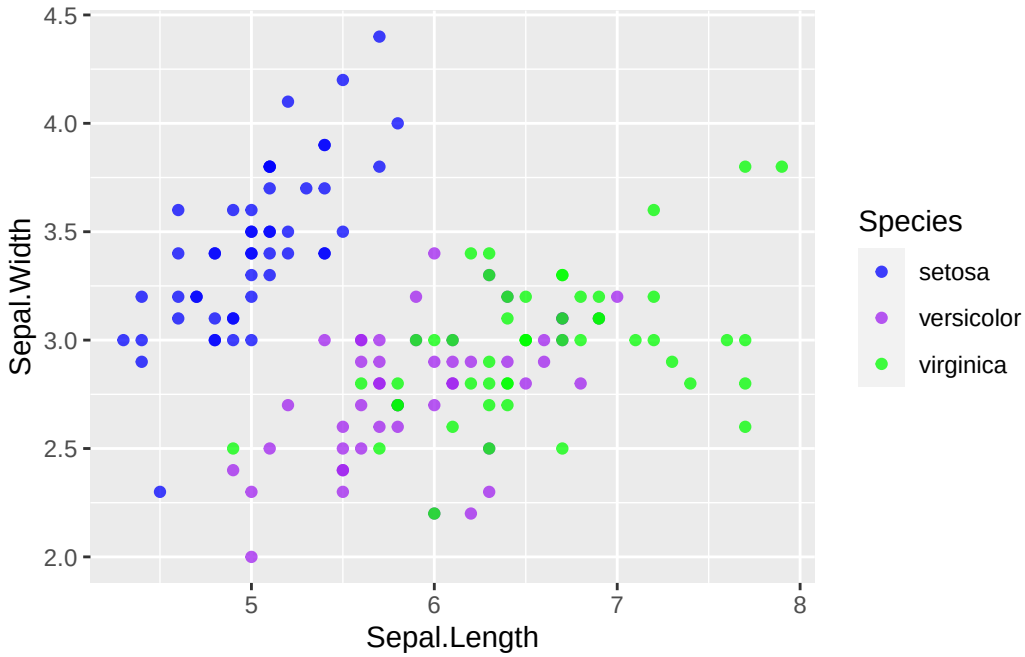



```
## Escala de color: gradiente manual (variable continua)
iris %>%

   # inicio gráfico
```

```
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Sepal.Length)) +
  # especifico geoma
  geom_point(alpha = .75, size = 3) +

  # escala de colores
  scale_color_gradient(low = "blue", high = "green")
```

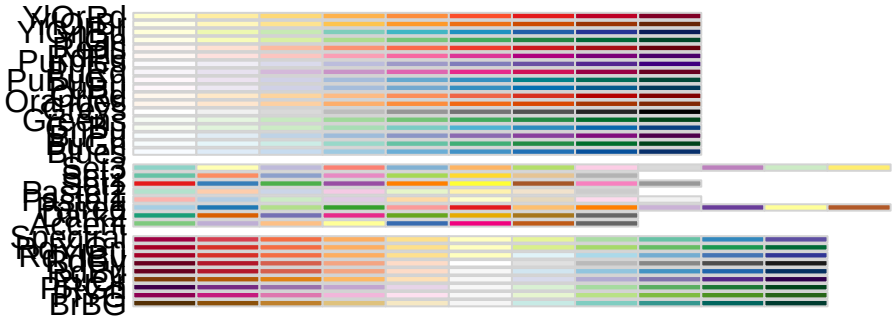



```
## Escala de color: colores definidos manualmente (variable discreta)
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species)) +
  # especifico geoma
  geom_point(alpha = .75) +

  # escala de colores
  scale_color_manual(values = c("blue","purple","green"))
```

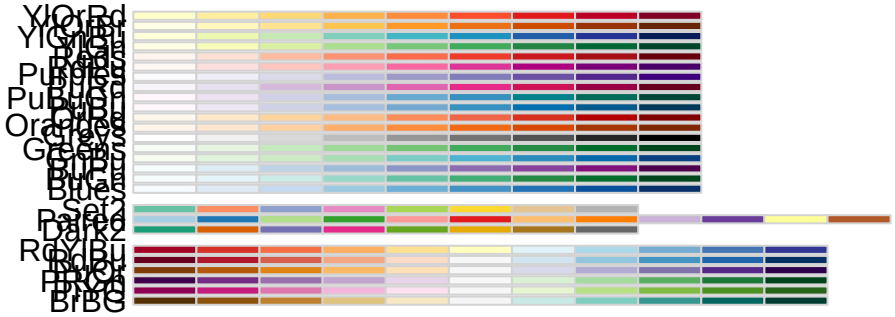


Las últimas versiones de `tidyverse` instalan por defecto los paquetes `RColorBrewer` y `viridis` por lo que no es necesario cargarlos para su uso. De todos modos, podemos previsualizar las paletas disponibles con el siguiente comando:

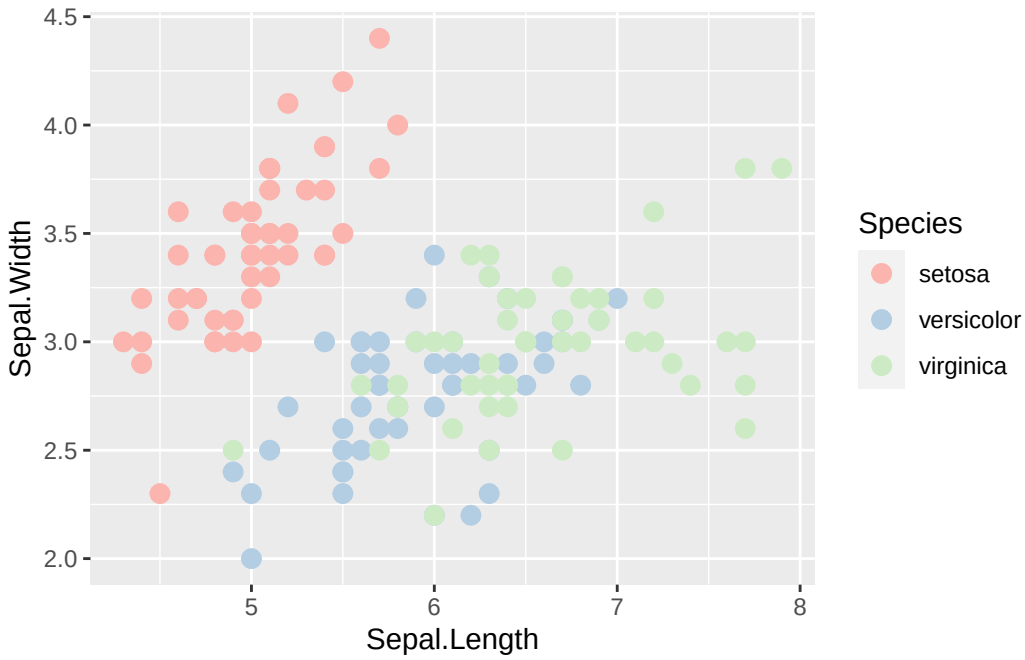
```
##  Muestra todas las paletas de RColorBrewer
RColorBrewer::display.brewer.all()
```



```
## Muestra solo paletas aptas daltonismo
RColorBrewer::display.brewer.all(colorblindFriendly = T)
```

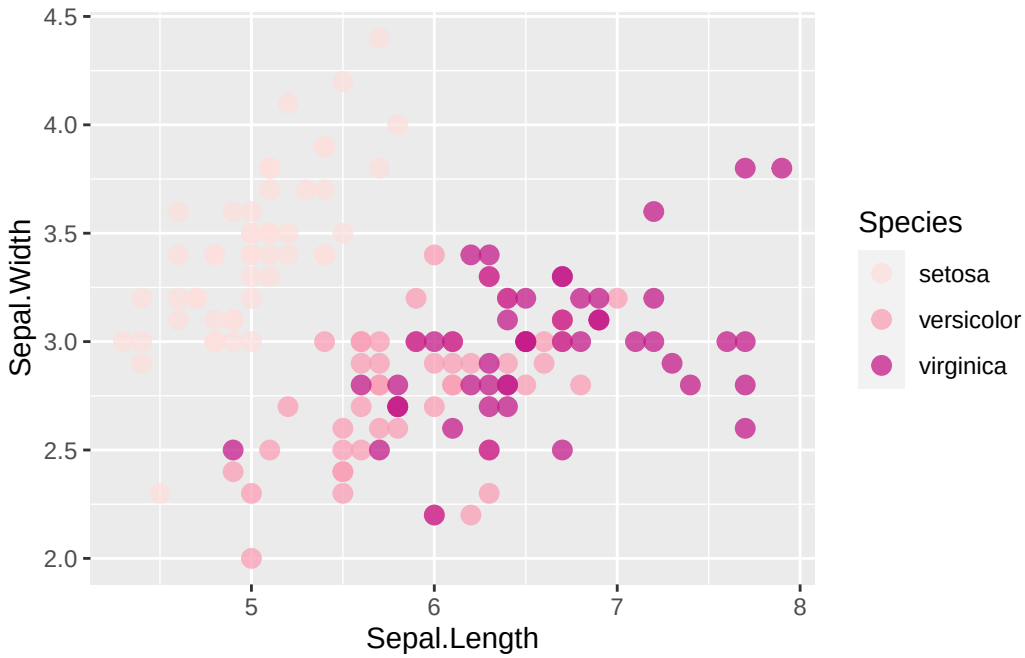


```r
## Escala de color discreta: RColorBrewer
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species)) +
  # especifico geoma
  geom_point(size = 3) +

  # escala de colores
  scale_color_brewer(palette = "Pastel1")
```



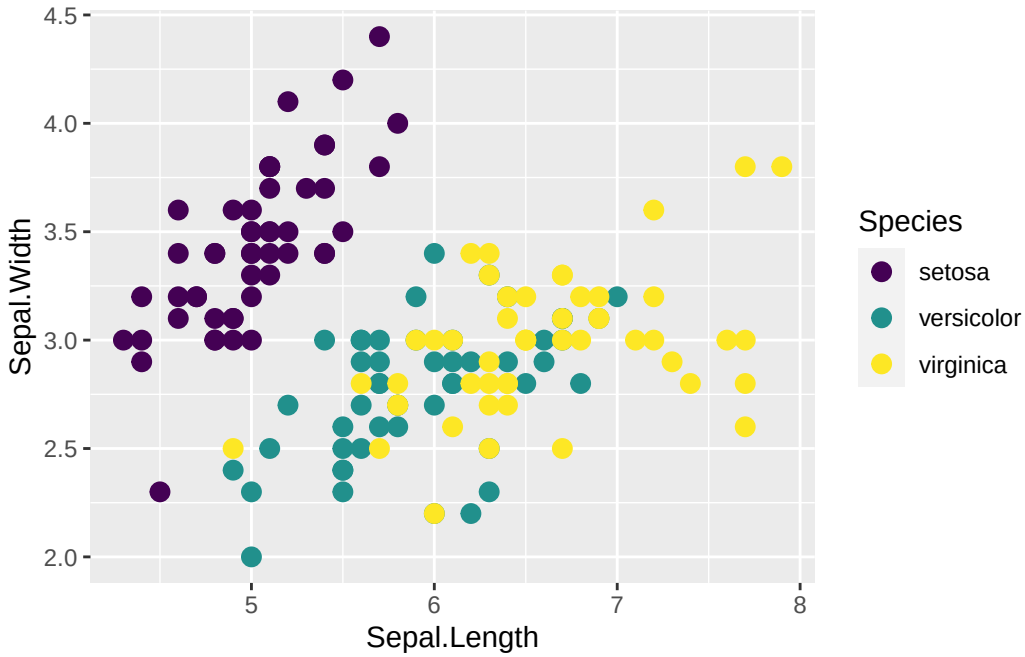

```r
## Escala de color discreta: RColorBrewer (colorblind-friendly)
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species)) +
  # especifico geoma
  geom_point(alpha = .75, size = 3) +

  # escala de colores
```

```
  scale_color_brewer(palette = "RdPu")
```

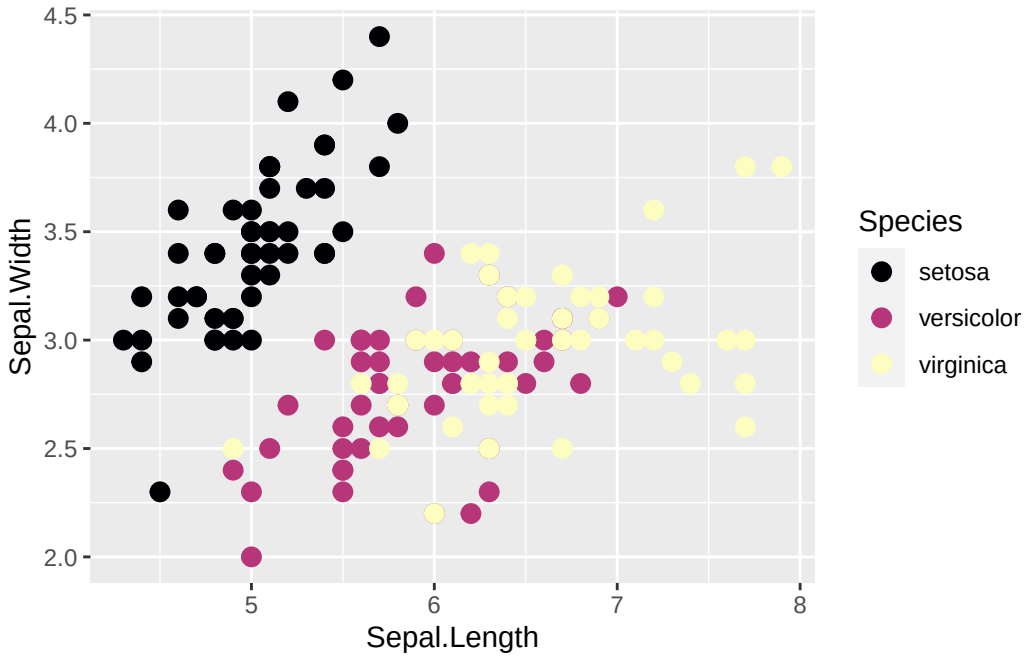


Las paletas del paquete `viridis` son todas aptas para daltonismo y tienen varias opciones que se pueden seleccionar mediante el argumento `option = "paleta"`. Si no especifico este argumento, el paquete usa por defecto la paleta `"viridis"`. La lista completa de paletas se puede consultar en este [LINK]. Veamos algunos ejemplos:

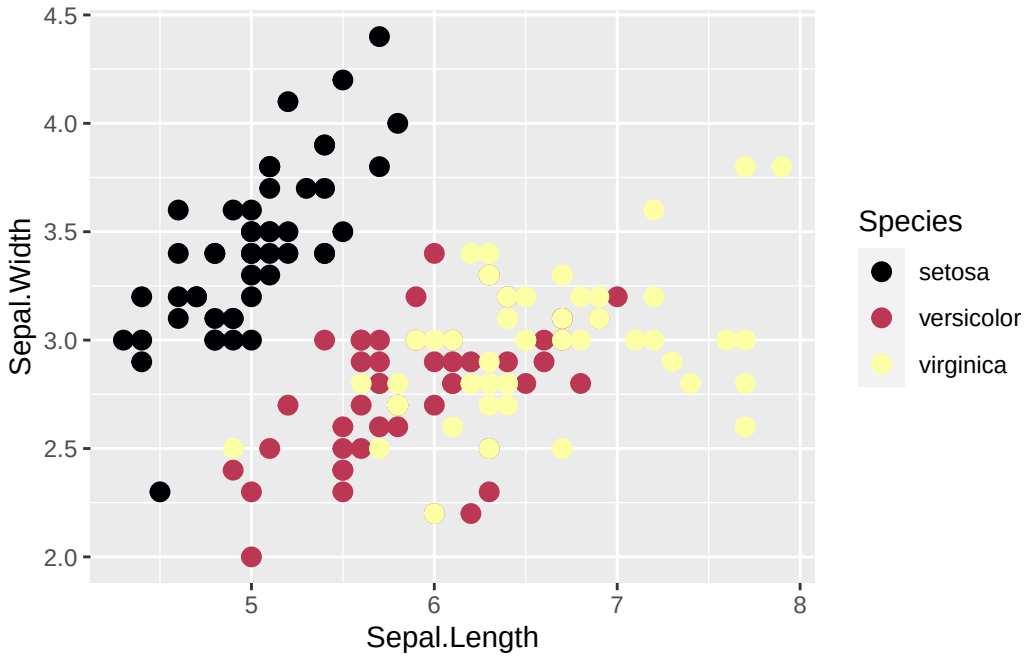
```
## Escala de color discreta: viridis
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species)) +
  # especifico geoma
  geom_point(size = 3) +

  # escala de colores
  scale_color_viridis_d()
```



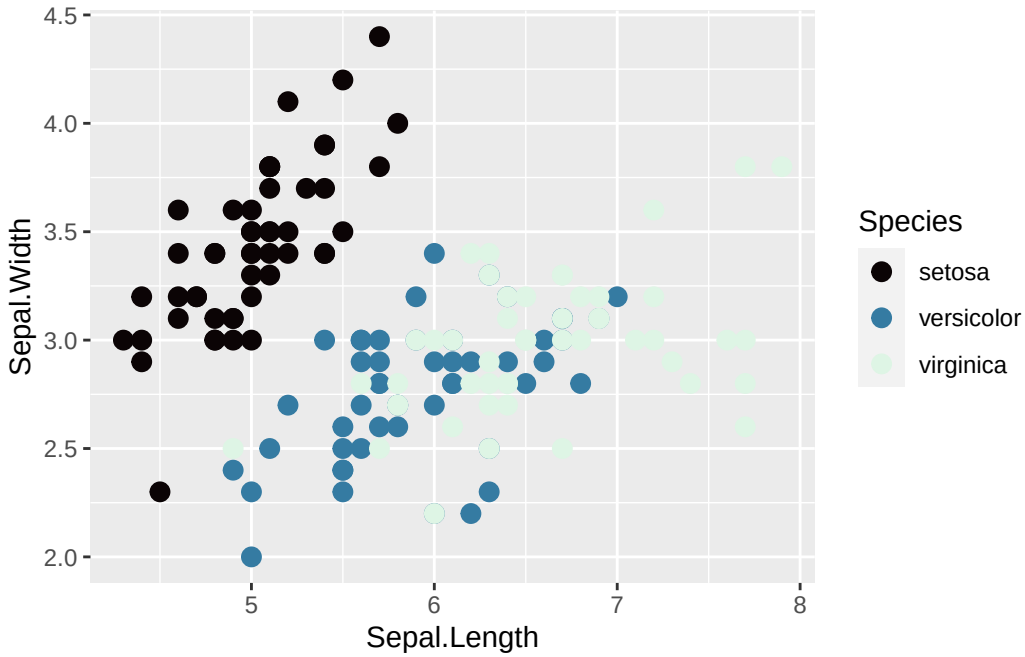

```
## Escala de color discreta: magma
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species)) +
  # especifico geoma
  geom_point(size = 3) +

  # escala de colores
  scale_color_viridis_d(option = "magma")
```




```
## Escala de color discreta: inferno
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species)) +
  # especifico geoma
  geom_point(size = 3) +

  # escala de colores
  scale_color_viridis_d(option = "inferno")
```



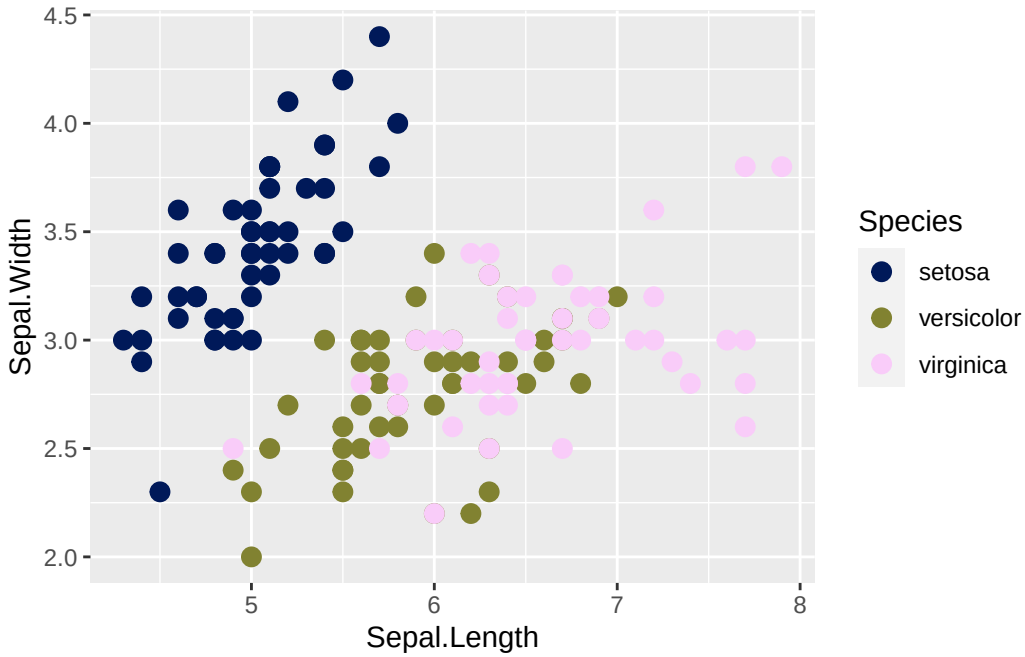

```
## Escala de color discreta: mako
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species)) +
  # especifico geoma
  geom_point(size = 3) +

  # escala de colores
  scale_color_viridis_d(option = "mako")
```



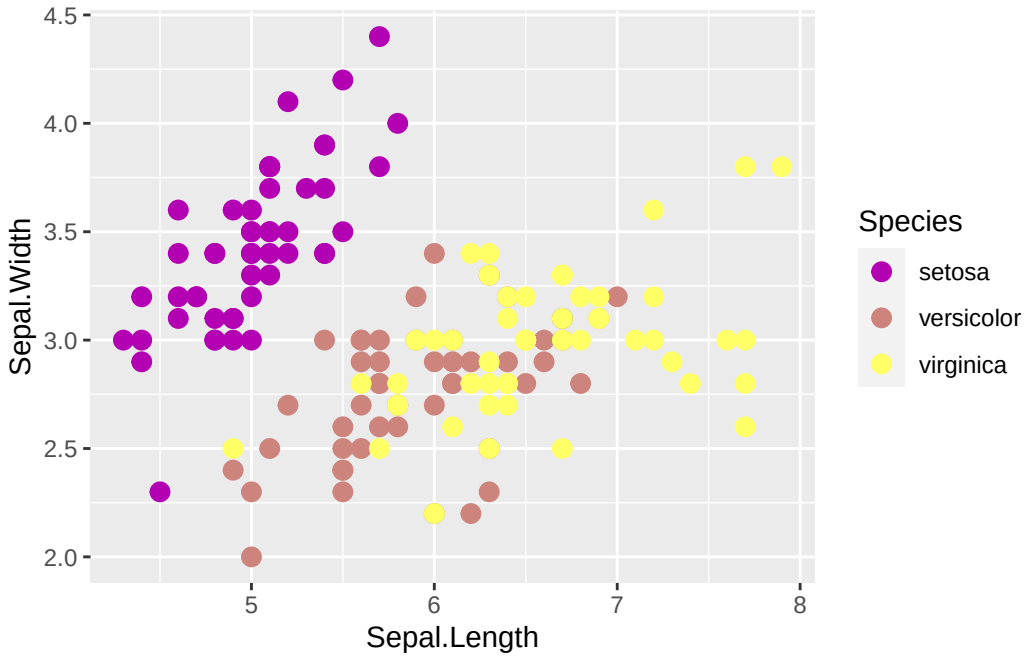
El paquete `scico` trae varias paletas *colorblind-friendly* aptas para publicaciones científicas. A diferencia de los dos paquetes anteriores, debo instalarlo y cargarlo antes de usarlo. Si no especifico nada en el argumento `palette = "paleta"` usará por defecto la paleta `"batlow"`. La lista completa de paletas disponibles se puede consultar en este [LINK] o ejecutando el siguiente comando:

```
scico::scico_palette_show()
```


```
library(scico)

## Escala de color discreta: scico batlow
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species)) +
  # especifico geoma
  geom_point(size = 3) +

  # escala de colores
  scale_color_scico_d()
```




```
## Escala de color discreta: scico buda
iris %>%

   # inicio gráfico
  ggplot(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                       color = Species)) +
  # especifico geoma
  geom_point(size = 3) +

  # escala de colores
  scale_color_scico_d(palette = "buda")
```



También existen sitios web como HTML Color Codes donde pueden generar paletas personalizadas o usar algunas preexistentes y copiarlas a R en formato hexadecimal.

Títulos, subtítulos y *captions*

Cuando vimos la configuración de los ejes, pudimos cambiar manualmente sus nombres desde `scale_eje_xxx()`, pero también existe el comando `labs()` donde podemos especififar no solamente los nombres de los ejes sino agregar títulos y pies de gráfico (*captions).* Veamos un ejemplo a partir del objeto `g1` que creamos anteriormente:

```
g1 +
  labs(y = "Ancho", x = "Longitud",
       title = "Relación longitud-ancho del sépalo",
       caption = "Figura 1. Relación entre longitud y ancho del sépalo \n
       en tres especies de plantas del género Iris")
```

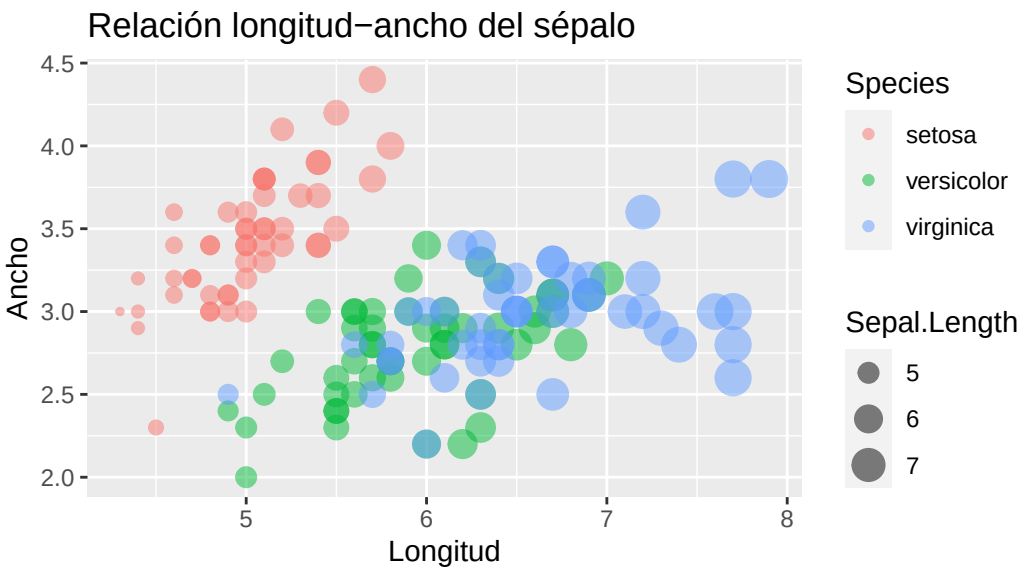


Figura 1. Relación entre longitud y ancho del sépalo

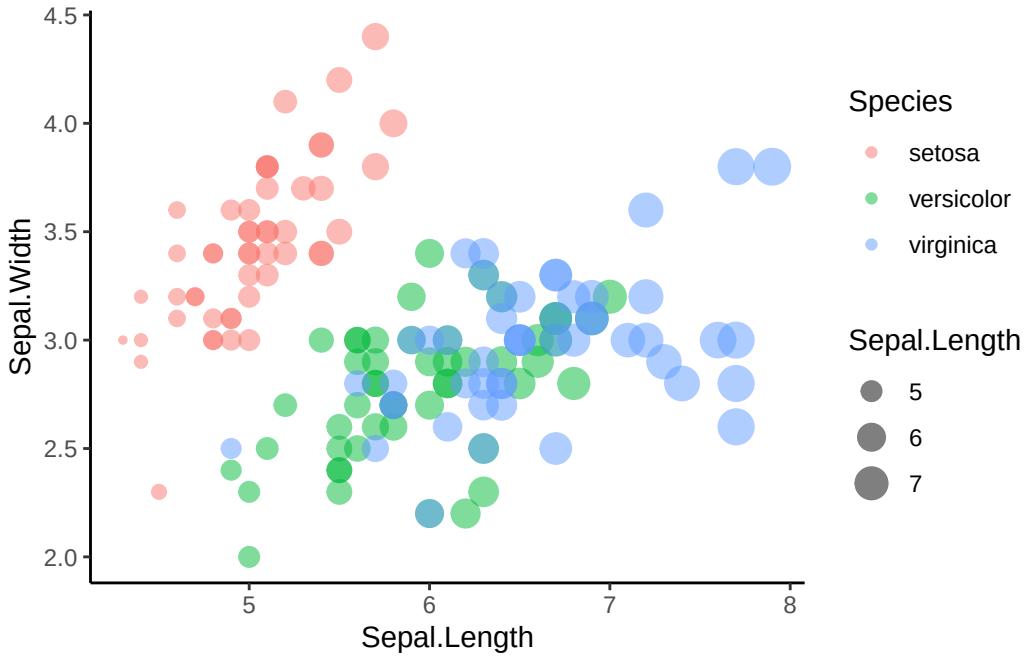
en tres especies de plantas del género Iris

En este ejemplo, usamos la expresión `"\n"` para indicar que queremos separar el texto en dos renglones. También podemos insertar etiquetas de texto dinámicas con la función `str_glue(texto, código)`.

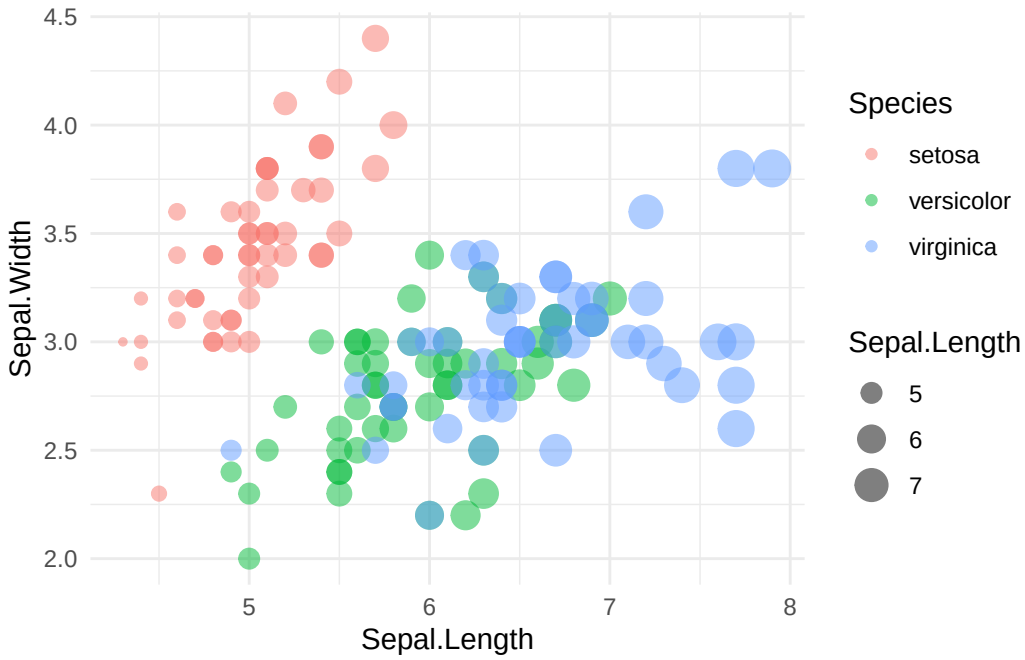
Temas

Finalmente, podemos cambiar el formato visual del gráfico utilizando temas predeterminados con los comandos `theme_xxx()`:

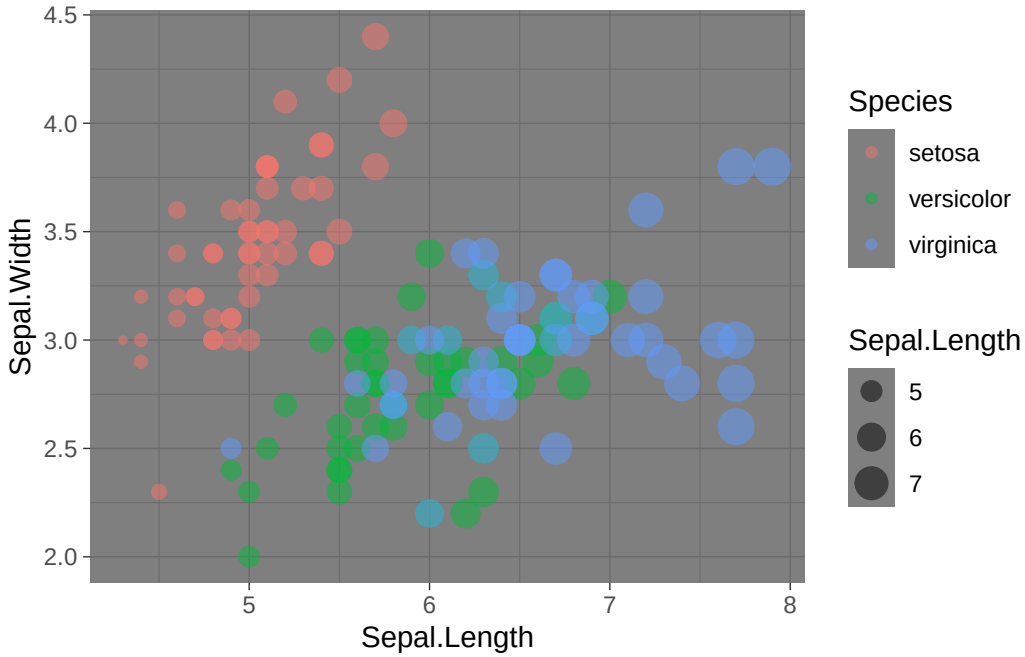
```
## Tema predeterminado: classic
g1 +
  theme_classic()
```




```
## Tema predeterminado: minimal
g1 +
  theme_minimal()
```

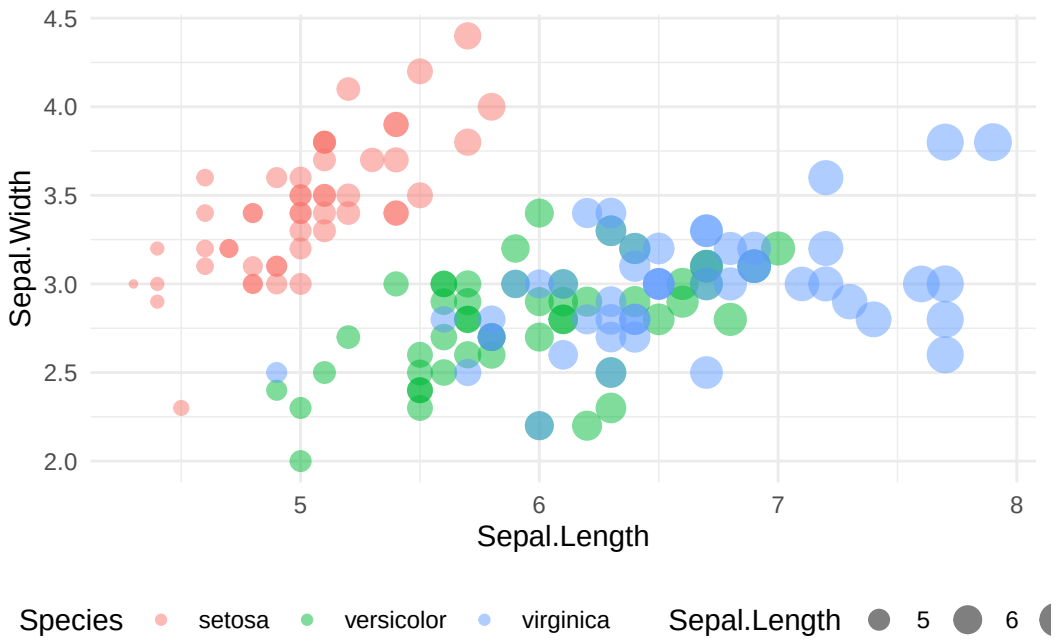


```
## Tema predeterminado: dark
g1 +
  theme_dark()
```

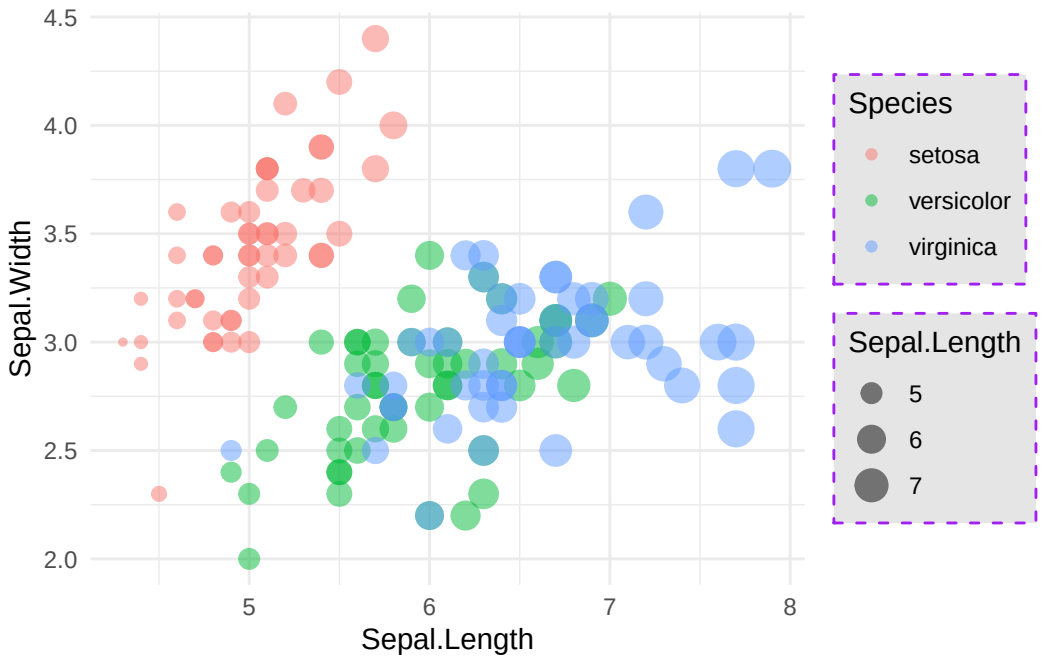


También podemos editar elementos del tema usando el comando `theme()`. Es importante que si vamos a usar un estilo predeterminado y queremos hacerle alguna modificación, usemos `theme_xxX()`y luego `theme()`:

```
# Leyenda debajo del gráfico
g1 +
  theme_minimal() +
  theme(legend.position = "bottom")
```




```
# cambio color de fondo de la leyenda
g1 +
  theme_minimal() +
  theme(legend.background = element_rect(color = "purple", fill = "grey90",
                                         linetype = "dashed"))
```


4.5
4.0
3.5
3.0
2.5
2.0
Sepal.Width
5
6
7
8
Sepal.Length
Species
setosa
versicolor
virginica
Sepal.Length
5
6
7