

1 Introducción

1.1 Principios para crear gráficos

1.2 Elementos de un gráfico

2 Sistemas de gráficos en **R**

3 Gráficos con ggplot2

3.1 ¿Cómo conseguir ggplot2 ?

3.2 Gramática de gráficos

3.2.1 Componentes en ggplot2

3.3 Primer gráfico paso a paso

3.4 Geometrías: los bloques para construir gráficos

3.4.1 Gráficos de puntos

3.4.2 Boxplot o diagramas de cajas

3.4.3 Gráfico de tipo violín

3.4.4 Histogramas, polígonos de frecuencia y densidad

3.4.5 Gráficos de barras o columnas

3.4.6 Gráficos de medias e intervalos

3.4.7 Gráficos de torta

3.4.8 Gráficos de texto

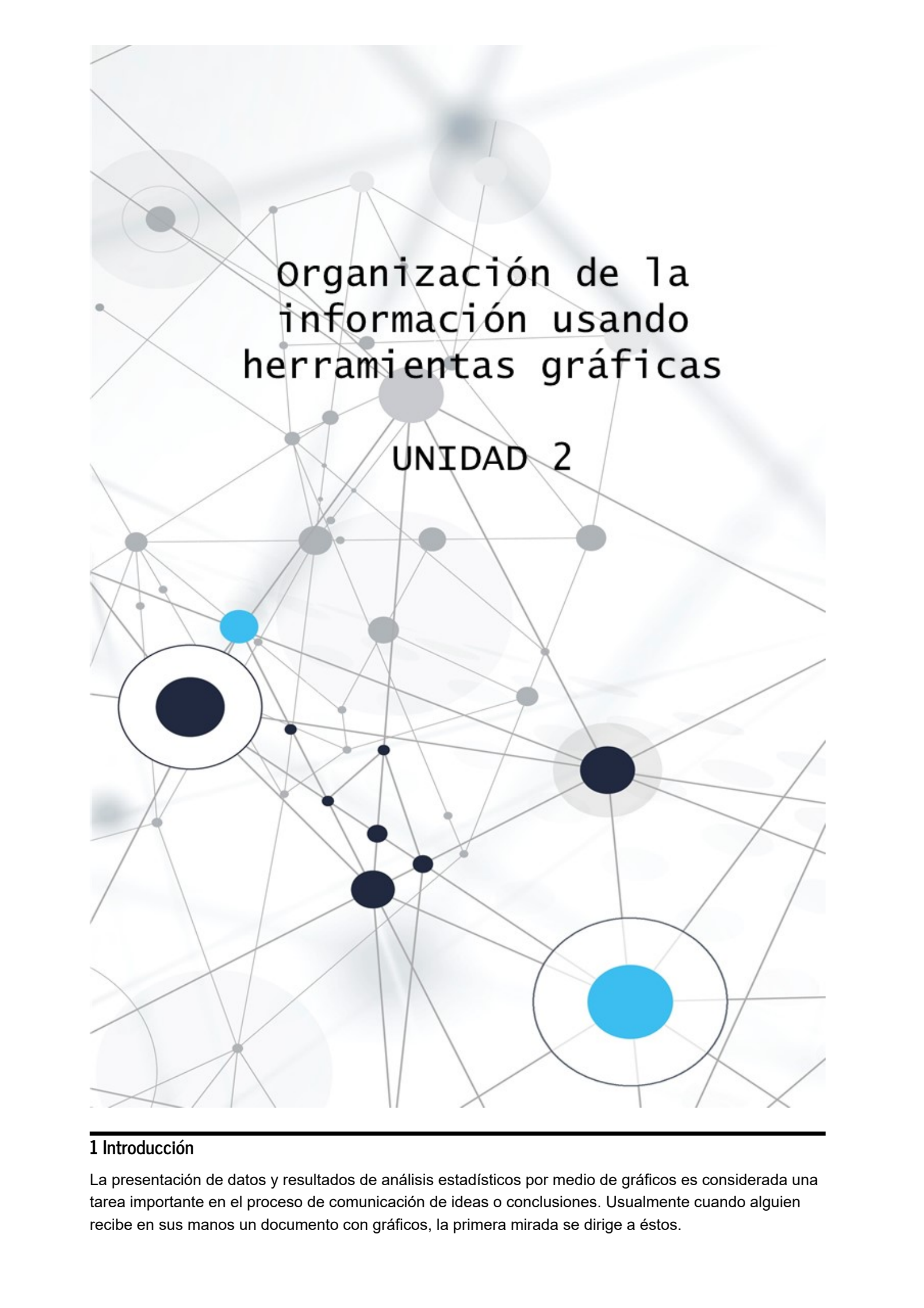
3.4.9 Algunos extras

3.5 Sistemas de coordenadas

3.6 Gráficos condicionales o por paneles: **facets**

3.7 Temas

3.8 Librería **plotly** : gráficos interactivos y más llamativos



Organización de la información usando herramientas gráficas

UNIDAD 2

1 Introducción

La presentación de datos y resultados de análisis estadísticos por medio de gráficos es considerada una tarea importante en el proceso de comunicación de ideas o conclusiones. Usualmente cuando alguien recibe en sus manos un documento con gráficos, la primera mirada se dirige a éstos.

Los gráficos estadísticos son importantes en los distintos momentos del análisis de los datos. En una primera instancia son muy útiles para complementar medidas de resumen y tomar conocimiento de la estructura de los mismos, descubrir patrones o tendencias, o bien detectar situaciones anómalas. Si el estudio es observacional o exploratorio, los gráficos pueden ayudarnos a generar futuras preguntas de investigación. Luego en la parte de modelado, son importantes para verificar el desempeño de los modelos y verificar sus supuestos. Por último, en la etapa final, un gráfico correctamente diseñado puede ayudar a resumir y comunicar los principales hallazgos del estudio.

La siguiente cita de John Tukey (https://es.wikipedia.org/wiki/John_W._Tukey) resume lo anterior:

Un gráfico puede valer más que mil palabras, pero puede tomar muchas palabras para hacerlo John W. Tukey

En esta unidad veremos como esas *palabras* requeridas para hacer el gráfico se traducen en líneas de código en **R**.

1.1 Principios para crear gráficos

Una ventaja de los gráficos es que pueden mostrarnos cosas que de otra forma es muy difícil o imposible ver, ésta es una de las razones por las cuales casi todos los análisis estadísticos deberían comenzar con gráficos. Correa y Gonzales (2002) (<https://cran.r-project.org/doc/contrib/grafi3.pdf>), enuncian los siguientes principios a considerar al momento de presentar un gráfico:

Entendibilidad ¿Nos permite el gráfico visualizar las relaciones entre las variables? ¿Interactúan los elementos en el gráfico para maximizar nuestra percepción de las relaciones entre las variables?

Claridad ¿Son los elementos del gráfico claramente distinguibles? ¿Son los elementos más importantes del gráfico visualmente prominentes?

Consistencia ¿Son los elementos de los gráficos consistentes con su uso en gráficos anteriores? ¿Existen nuevos elementos del gráfico que requieren una descripción adicional?

Eficiencia ¿Están los elementos del gráfico representando eficientemente los datos? ¿Hay elementos en el gráfico que sirven a más de un propósito?

Necesidad ¿Es el gráfico una forma útil de representar estos datos? ¿Es cada elemento en el gráfico necesario?

1.2 Elementos de un gráfico

También los mismos autores identifican los elementos básicos de un gráfico estadístico:

- Títulos: principal y secundarios (si son necesarios)
- Descripción del Gráfico, generalmente en la descripción al pie de la figura.
- Región de datos y símbolos
- Ejes (horizontal, vertical o ambos)
- Escalas con las unidades de los ejes
- Leyendas con el código de colores, marcas, etc.

2 Sistemas de gráficos en R

R cuenta con dos sistemas para graficar: `graphics` y `grid`. Ambos vienen por defecto con la instalación de **R**.

El paquete `graphics`, también conocido como `base plot system`. Este paquete provee la función genérica `plot()` para hacer gráficos simples, y otras funciones para gráficos específicos (`hist()`, `barplot()`, `boxplot()`, etc.). Usa un enfoque de *papel y lápiz* por capas donde el gráfico final es una sumatoria de capas que se agregan una a la vez sin posibilidad de modificarse luego. Generalmente es OK para gráficos simples o exploratorios. Para gráficos más complejos (con subgrupos o multipanel) requiere programar más. Una desventaja es la sintaxis poco consistente.

El paquete `grid` agrega funcionalidades al sistema base para definir paneles, escalas, etc., pero no se usa directamente. Sobre estas funcionalidades se desarrollaron dos paquetes muy importantes: `lattice` y `ggplot2`

El paquete `lattice`, desarrollado por Deepayan Sarkar, implementa gráficos tipo `trellis` (multipanel). `lattice` tiene un sintaxis más coherente y en vez de tener un enfoque por capas, todos los componentes del gráfico se declaran en una función. muy conveniente para gráficos condicionales pero complicada para combinar gráficos o hacer ajustes finos.

El paquete `ggplot2`, desarrollado por Hadley Wickham (<http://hadley.nz/>), está basado en la filosofía *Gramática de gráficos* (*grammar of graphics*, por eso `gg`). Combina los dos enfoques: *por capas* y *función*. Uno provee los datos, indica que variables asignar a las estéticas (ejes, escalas, colores, símbolos) y las geometrías o formas que se quieren graficar y `ggplot2` se encarga del resto. Se puede ir agregando capas. Es muy potente para la exploración y visualización de datos en formato de tabla con filas (observaciones) y columnas (variables).

En esta unidad vamos a desarrollar las funciones de `ggplot2` que forma parte del paquete `tidyverse` (<https://ggplot2.tidyverse.org/>)



ggplot2



Overview

ggplot2 is a system for declaratively creating graphics, based on [The Grammar of Graphics](#). You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

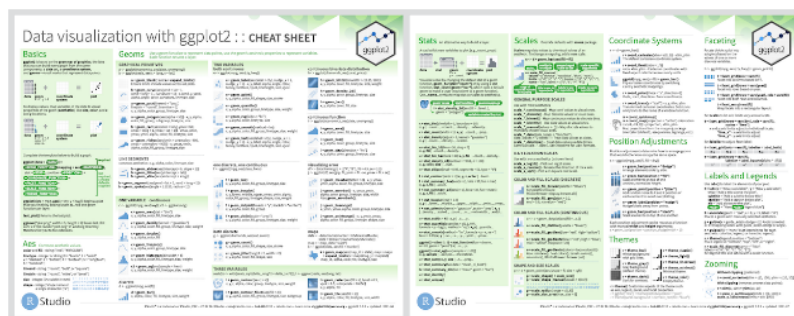
Installation

```
# The easiest way to get ggplot2 is to install the whole tidyverse:
install.packages("tidyverse")

# Alternatively, install just ggplot2:
install.packages("ggplot2")

# Or the development version from GitHub:
# install.packages("pak")
pak::pak("tidyverse/ggplot2")
```

Cheatsheet



Usage

It's hard to succinctly describe how ggplot2 works because it embodies a deep philosophy of visualisation. However, in most cases you start with `ggplot()`, supply a dataset and aesthetic mapping (with `aes()`). You then add on layers (like `geom_point()` or `geom_histogram()`), scales (like `scale_colour_brewer()`), faceting specifications (like

LINKS

[View on CRAN](#)
[Browse source code](#)
[Report a bug](#)
[Learn more](#)
[Extensions](#)

LICENSE

[Full license](#)
[MIT](#) + file [LICENSE](#)

COMMUNITY

[Contributing guide](#)
[Code of conduct](#)

CITATION

[Citing ggplot2](#)

DEVELOPERS

[Hadley Wickham](#)
Author
[Winston Chang](#)
Author
[Lionel Henry](#)
Author
[Thomas Lin Pedersen](#)
Author, maintainer
Kohske Takahashi
Author
[Claus Wilke](#)
Author
[Kara Woo](#)
Author
[Hiroaki Yutani](#)
Author

3 Gráficos con ggplot2

Como dijimos antes, ggplot2 es un paquete **R** para producir gráficos estadísticos implementando la *gramática de los gráficos*. Esto permite crear una gran variedad de gráficos combinando diversos componentes, evitando así las limitaciones de gráficos predefinidos. También ggplot2 es compatible con otros paquetes que permiten crear gráficos interactivos o animaciones.

El modelo en el que se basa ggplot2 lo hace realmente fácil de aprender: hay un conjunto simple de principios básicos y muy pocos casos especiales. También la documentación existente es muy buena y detallada (guía de referencia (<https://ggplot2.tidyverse.org/reference/index.html>), libro (<https://ggplot2-book.org/index.html>)), y la comunidad (<http://groups.google.com/group/ggplot2>) de usuarios es muy activa.

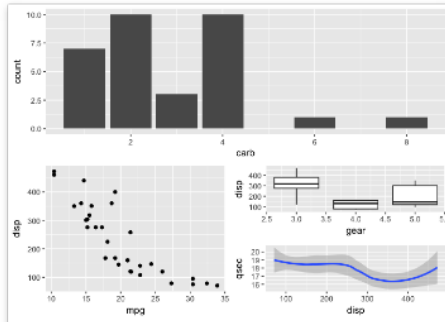
En la web existen varias galerías donde se pueden ver ejemplos (<https://www.r-graph-gallery.com/>) así como algunas extensiones (<https://exts.ggplot2.tidyverse.org/gallery/>) avanzadas. Estos recursos son muy útiles ya que contienen el código fuente para reproducir los gráficos.

127 registered extensions available to explore

Sort Author Filter Tag Filter CRAN Only

Github stars ▼ Text Filter ▼ ▼

Showing 107 of 127



patchwork Star 2252

Easy composition of ggplot plots using arithmetic operators

author: thomasp85

tags: visualization, composition

libraries:



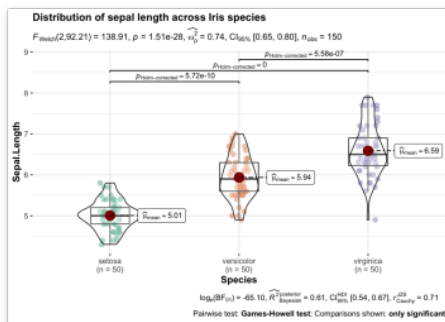
gganimate Star 1884

A Grammar of Animated Graphics.

author: thomasp85

tags: visualization, general

libraries:



ggstatsplot Star 1749

'ggstatsplot' provides a collection of functions to enhance 'ggplot2' plots with results from statistical tests.

author: IndrajeetPatil

tags: visualization, statistics

libraries:



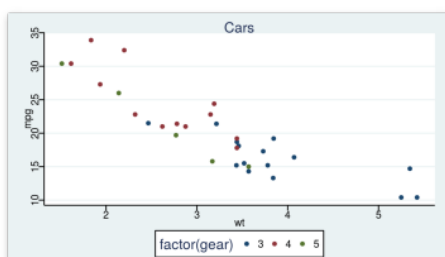
esquisse Star 1665

Explore and Visualize Your Data Interactively with ggplot2

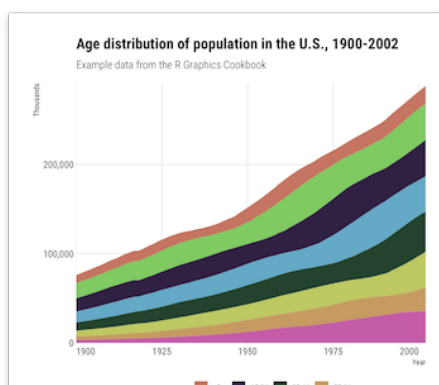
author: dreamrs

tags: visualization, interface

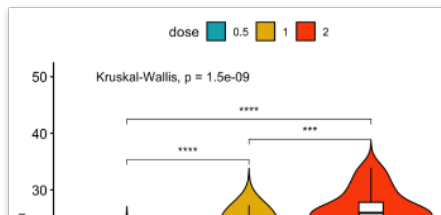
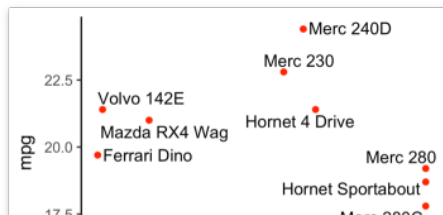
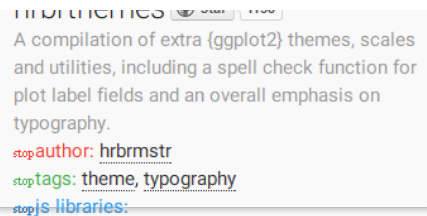
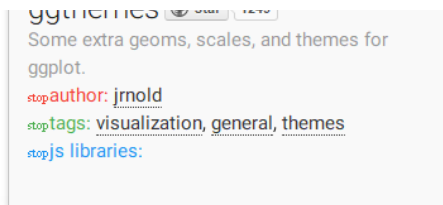
libraries:



gathemes Star 1249



hrbrthemes Star 1138



3.1 ¿Cómo conseguir ggplot2 ?

ggplot2 es parte del meta-paquete tidyverse por lo tanto para instalarlo por primera vez en la computadora tenemos las siguientes alternativas:

```
# Sólomente el paquete ggplot2
install.packages("ggplot2")
```

```
# O junto con la familia tidyverse
install.packages("tidyverse")
```

Lo anterior se debe realizar por única vez si el paquete no está previamente instalado en la máquina. Para usar las funciones en una sesión de trabajo hay que cargarlo con `library()` :

```
# Solo
library("ggplot2")
```

```
# O junto con la familia tidyverse
library("tidyverse")
```

En resumen:



Images sourced from <https://www.wikihow.com/Change-a-Light-Bulb>

R va a avisarnos en la consola que esta enmascarando (reemplazando) algunas funciones que ya estaban en el entorno, o bien el paquete nos devuelve algún mensaje. A menos que diga `Error ...`, eso está bien.

3.2 Gramática de gráficos

En líneas generales el concepto de gramática de gráficos es una forma describir los componentes de un gráfico de manera tal que permita su generalización. Este concepto fue propuesto por Leland Wilkinson (<https://www.springer.com/in/book/9780387245447>) quien describe una serie de reglas que definen como estos componentes del gráfico *mapean* las distintas características de los datos (variables, observaciones). `ggplot2` implementa una variante *por capas* de este paradigma (`gg` es por *grammar of graphics*). Como resultado, se crean una serie de capas que permiten describir y construir visualizaciones de manera estructurada en cuanto a representación de los elementos pero a su vez flexible para generar combinaciones nuevas.

3.2.1 Componentes en `ggplot2`

Los gráficos de `ggplot2` se definen por la combinación de capas (`layers`), escalas (`scales`), coordenadas (`coords`) y facetas (`facets`). Adicionalmente a estos componentes se pueden aplicar temas (`themes`) que permiten controlar los detalles del diseño de la visualización.

Los `layers` constan de 5 elementos: los datos (`data`), los elementos de mapeo (`mapping`), la transformación estadística (`stat`), la geometría (`geom`), y el ajuste de posición (`position`). Generalmente, sobre todo para gráficos simples, `data` y `mapping` se definen una vez para todo el gráfico dentro de la función `ggplot()` . En otras situaciones se da a nivel de cada `layer` . Los `layers` se construyen con las funciones `geom_*` y `stat_*` que veremos más adelante.

- El `data` es un set de datos que contiene la información que se desea visualizar en la capa. Generalmente es un objeto tipo `data.frame` o similar, e.g. `tibble` . En algunos casos se pueden combinar varios set de datos definidos para cada capa.
- Los elementos de mapeo son definidos mediante `aes()` para indicar la forma en que las variables y observaciones van a ser representadas en la visualización mediante atributos estéticos (ejes, líneas, colores, rellenos, etc.).
- Los `geoms` representan la parte visual un gráfico: puntos, líneas, polígonos, etc.
- Las `stats` resumen los datos aplicando funciones estadísticas, e.g. promedio, agrupamiento y conteo de observaciones, o ajuste de un modelo lineal o suavizado, etc.
- Los ajustes de posición (`position`) permiten controlar la posición de los elementos `geoms` dentro de un `layer` para evitar su superposición o
- Las `scales` asignan los valores del espacio de datos a valores en el espacio de los elementos estéticos (`aesthetics` o `aes`). Por ejemplo, el uso de un color, forma o tamaño de un `geom` puede ser controlado por un atributo de los datos. Las escalas también definen las leyendas y los ejes.
- Sistema de coordenadas (`coord`) que define que variables definirán el espacio del gráfico y como se representarán, e.g. coordenadas cartesianas, polares, etc.
- Paneles (`facets`) es un elemento que permite especificar una o más variables para dividir el gráfico en paneles y así mostrar subgrupos de datos. Esto permite ver visualizar relaciones condicionales entre variables, e.g. $y \sim x \mid z$, es decir, que pasa con la variable `x` e `y` cuando cambia `z` .

Adicionalmente a estos componentes se pueden aplicar temas (`themes`) que permiten controlar los detalles del diseño de la visualización, tipografía, posición de algunos objetos, paleta de colores, etc. Los valores predeterminados de `ggplot2` son un buen punto de partida pero existen opciones predefinidas que pueden modificarse para generar un tema particular.

3.3 Primer gráfico paso a paso

Veamos con un ejemplo como se combinan los componentes anteriormente vistos para realizar un gráfico simple. Para esto vamos a usar el set de datos `iris` que viene por defecto en **R**. Este set de datos contiene mediciones de la estructura floral de tres especies del género *Iris*.

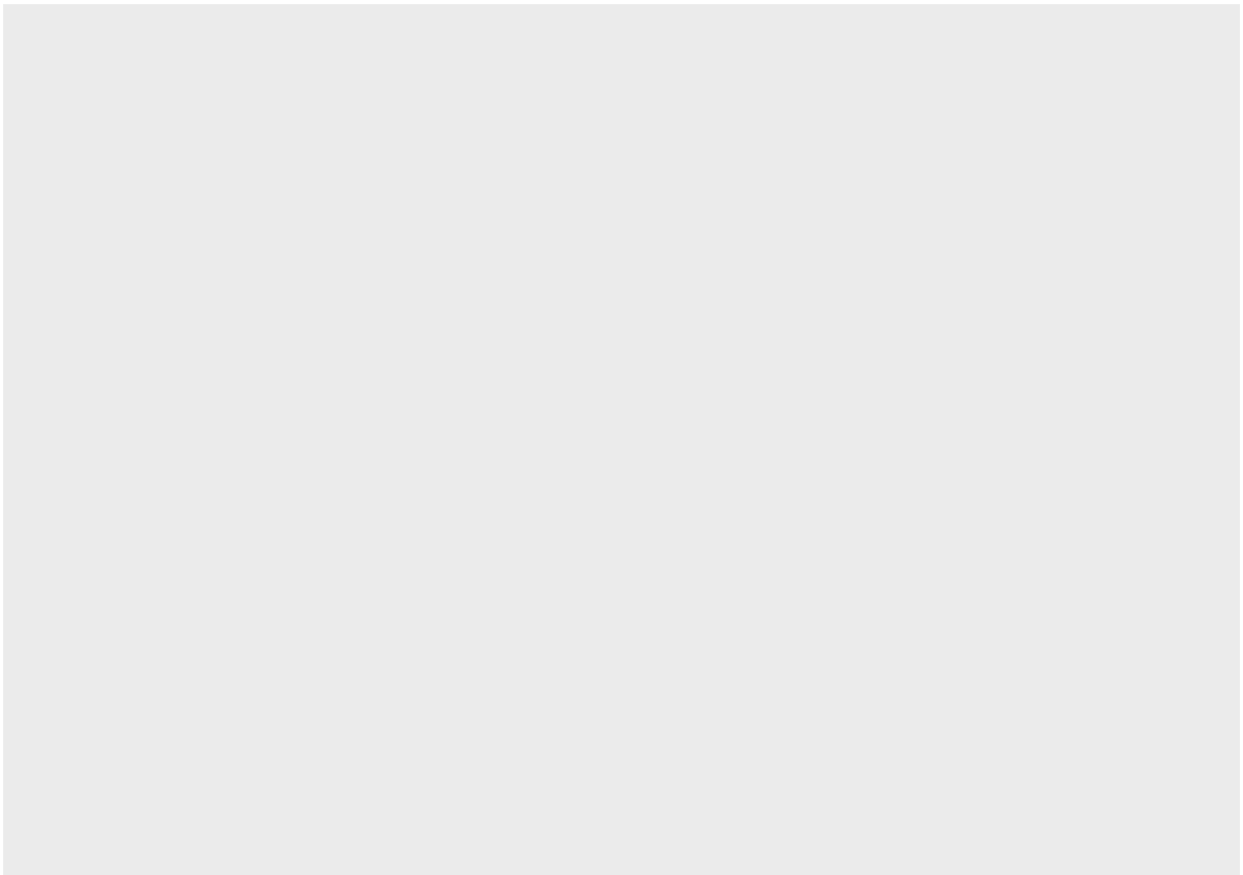
```
data(iris)
head(iris)
```

```
##   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
## 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
```

Nuestro primer gráfico tendrá como objetivo mostrar la relación que existe entre las características de la flor `Sepal.Width` y `Sepal.Length`, y potencialmente ver si esta es similar entre especies. Veamos paso por paso como se construye el gráfico.

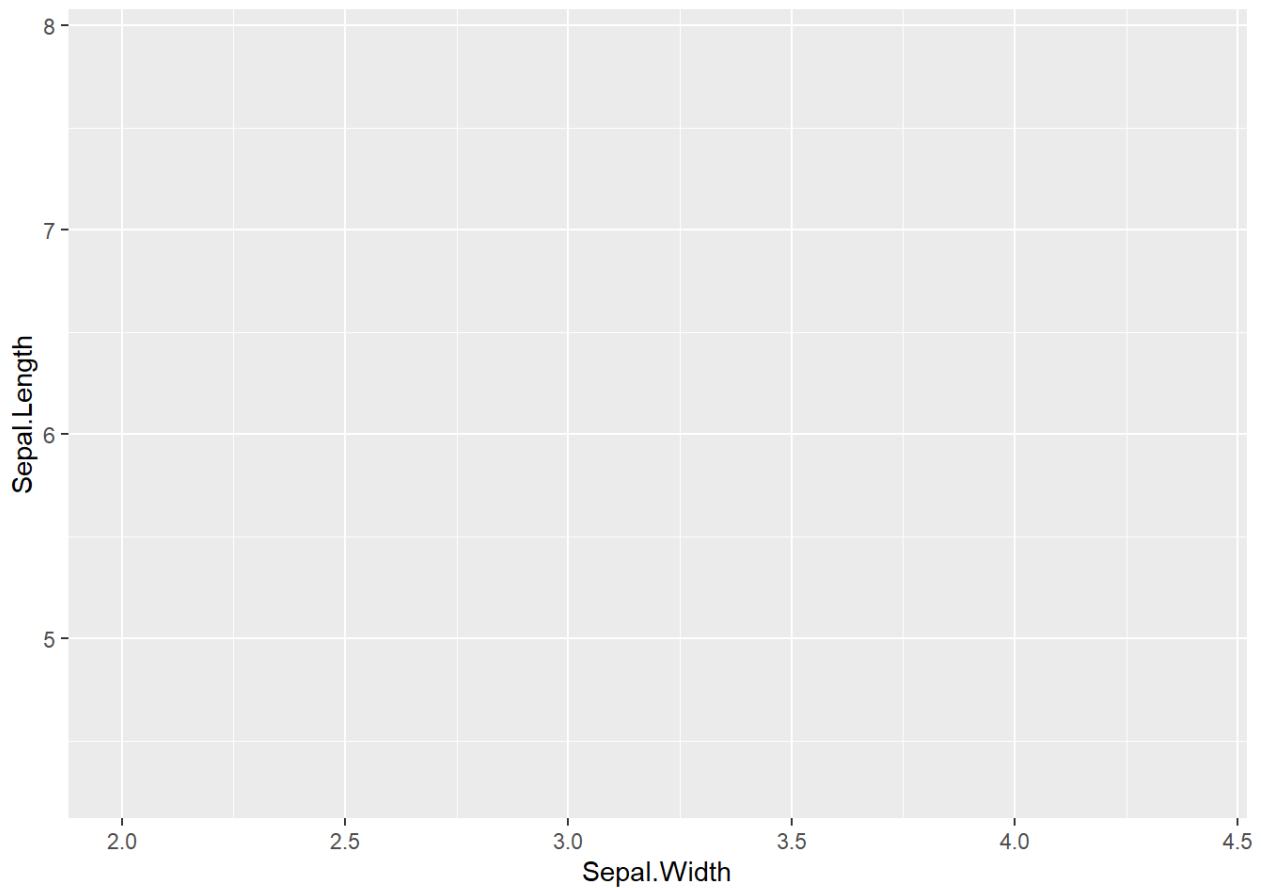
Primero definimos el set de datos que usaremos:

```
ggplot(data = iris)
```



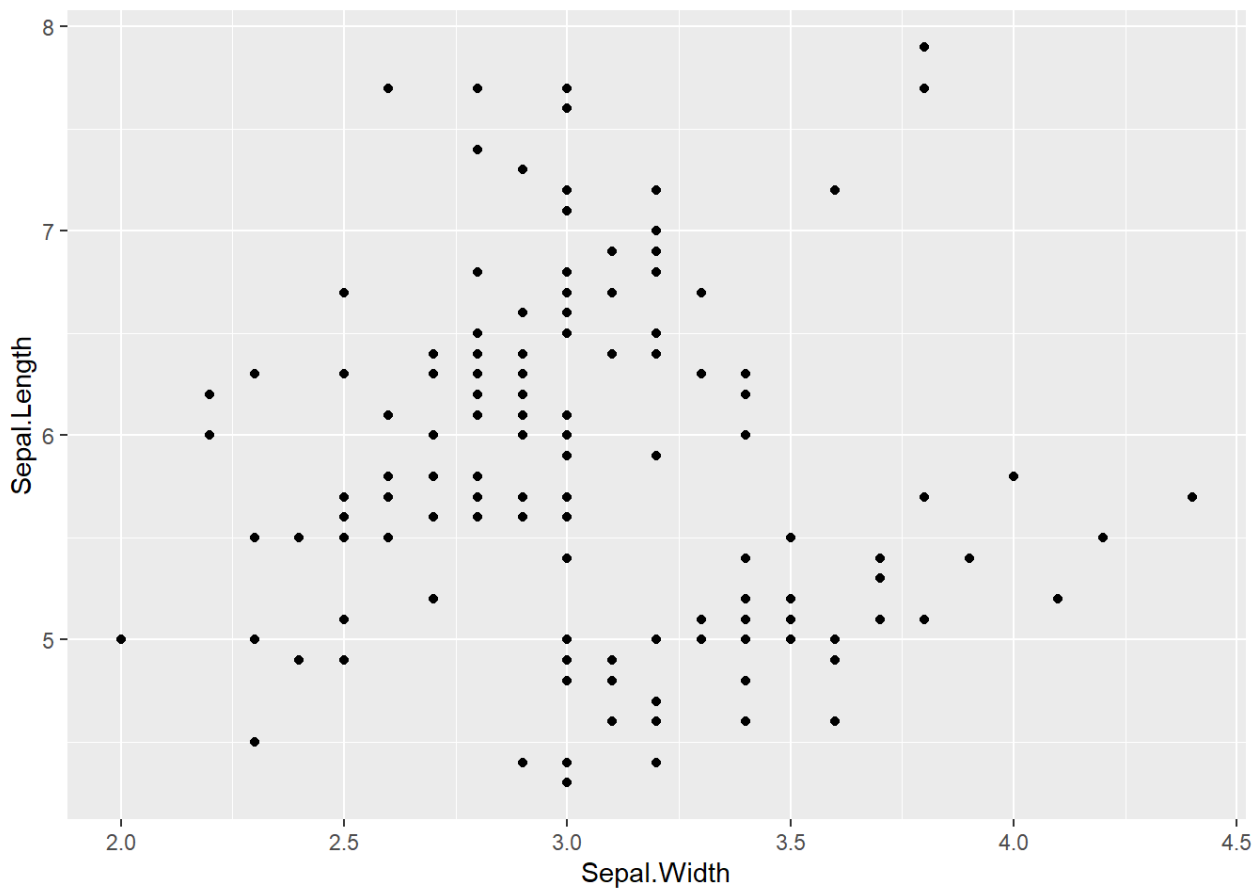
Como vemos esto no produjo nada ya que no indicamos cuales son las variables que queremos graficar y cómo graficarlas. Nuestro `layer` solo tiene la información de `data`. Agreguemos ahora la información de `mapping` usando `aes()`. Usando el operador `+` podemos concatenarlo al comando anterior.

```
ggplot(data = iris) +  
  aes(x = Sepal.Width, y = Sepal.Length)
```



Aquí vemos que, si bien no hemos graficado nada, la información suministrada permite a `ggplot` identificar los ejes, definir el espacio de coordenadas (cartesianas por defecto) y proponer unos límites en función del rango de valores. Agreguemos ahora la geometría: en este caso tiene sentido usar `geom_point()` ya que queremos mostrar un punto por observación

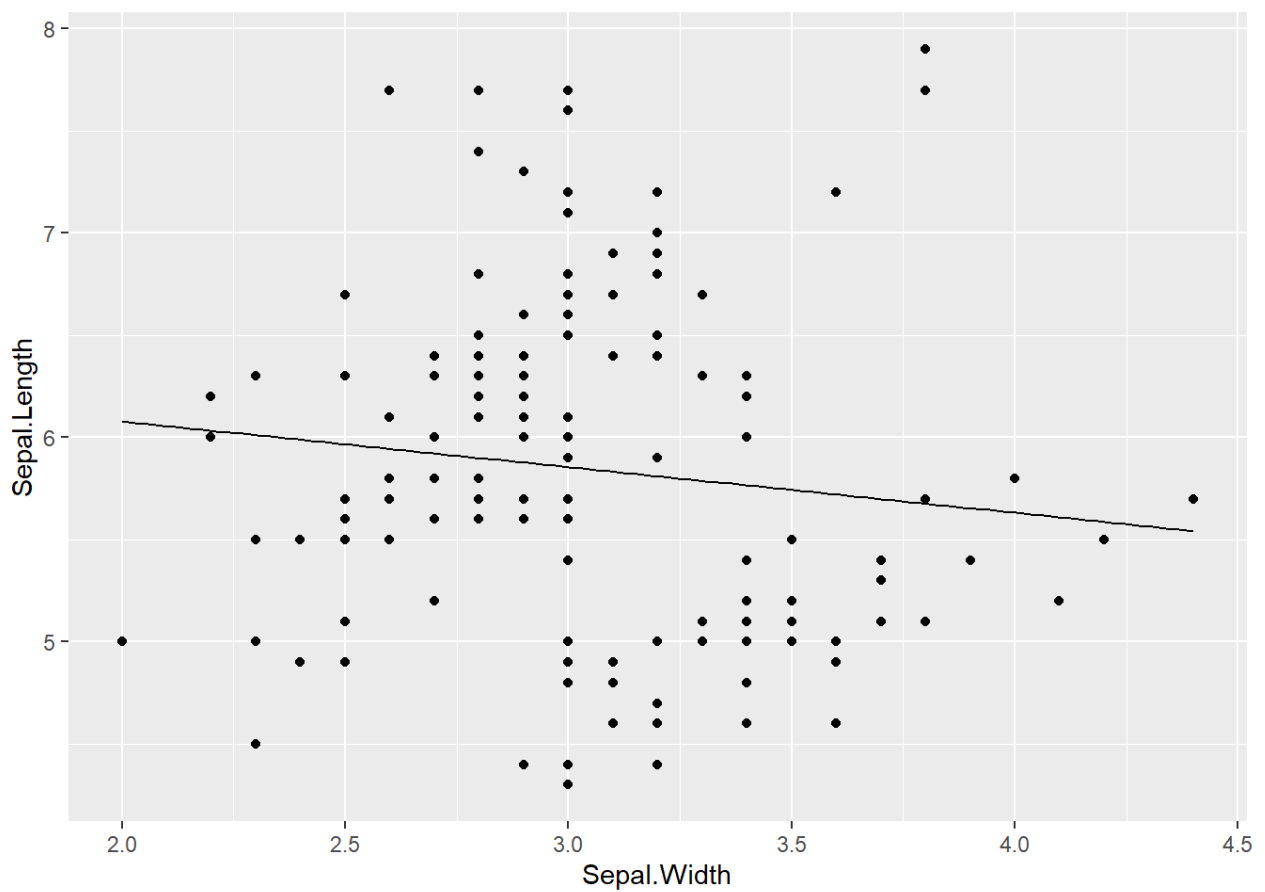
```
ggplot(data = iris) +  
  aes(x = Sepal.Width, y = Sepal.Length) +  
  geom_point()
```



Como vemos ahora el gráfico va tomando forma. Este tipo de gráficos se llama *gráfico de dispersión* y muestra la relación entre ambas variables. Por defecto no se aplica ninguna transformación estadística lo que equivale a (`stat = "identity"`).

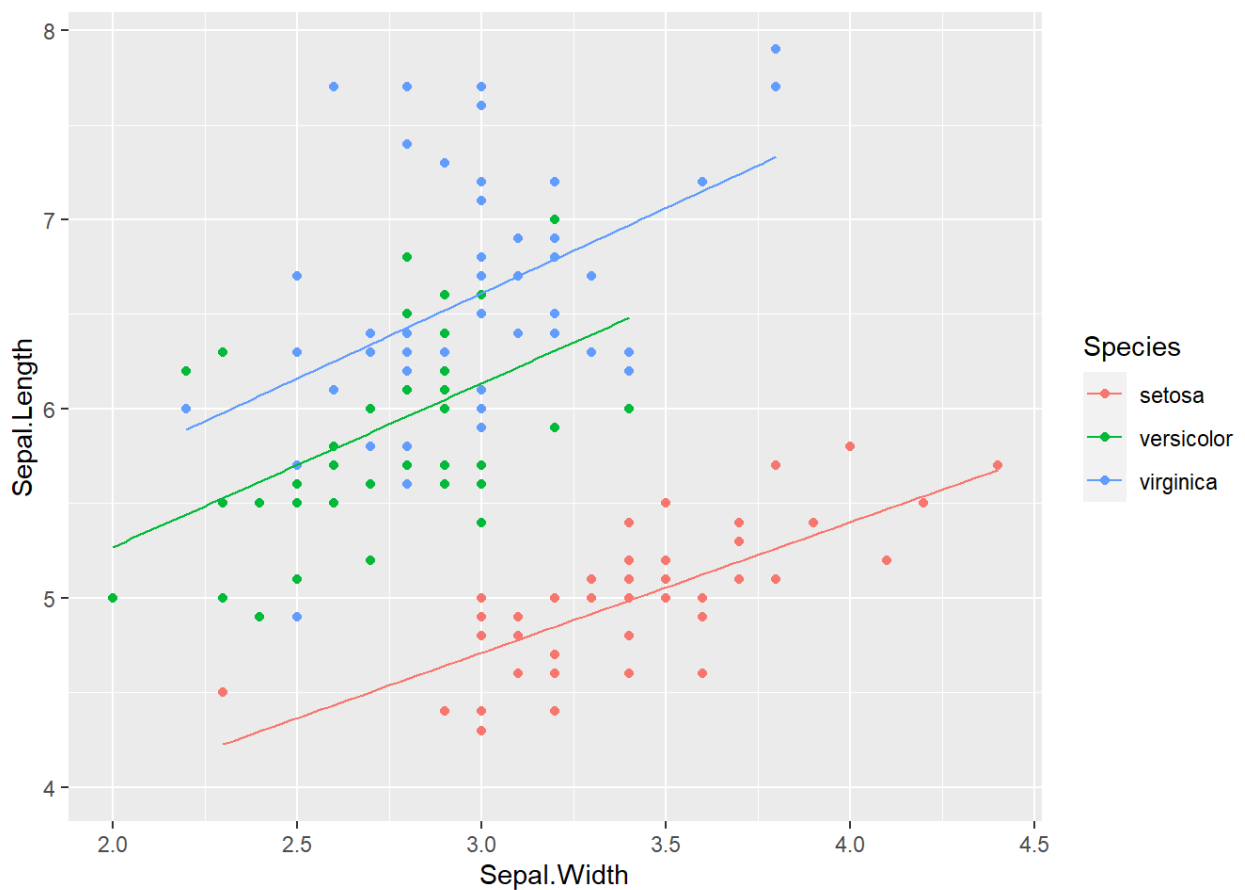
A este gráfico vamos a agregarle alguna función que permita resumir la relación entre ambas variables, por ejemplo un modelo de regresión. La mejor forma de representarlo sería una línea. Para eso vamos a agregar otro layer con `geom_line()` donde indicaremos una transformación de los datos `stat = smooth`.

```
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm')
```



Esta relación casi nula o negativa es para todo el set de datos y puede enmascarar algún patrón por especies. Veamos como agregar la información de `Species` al gráfico utilizando otros atributos estéticos como por ejemplo `color` :

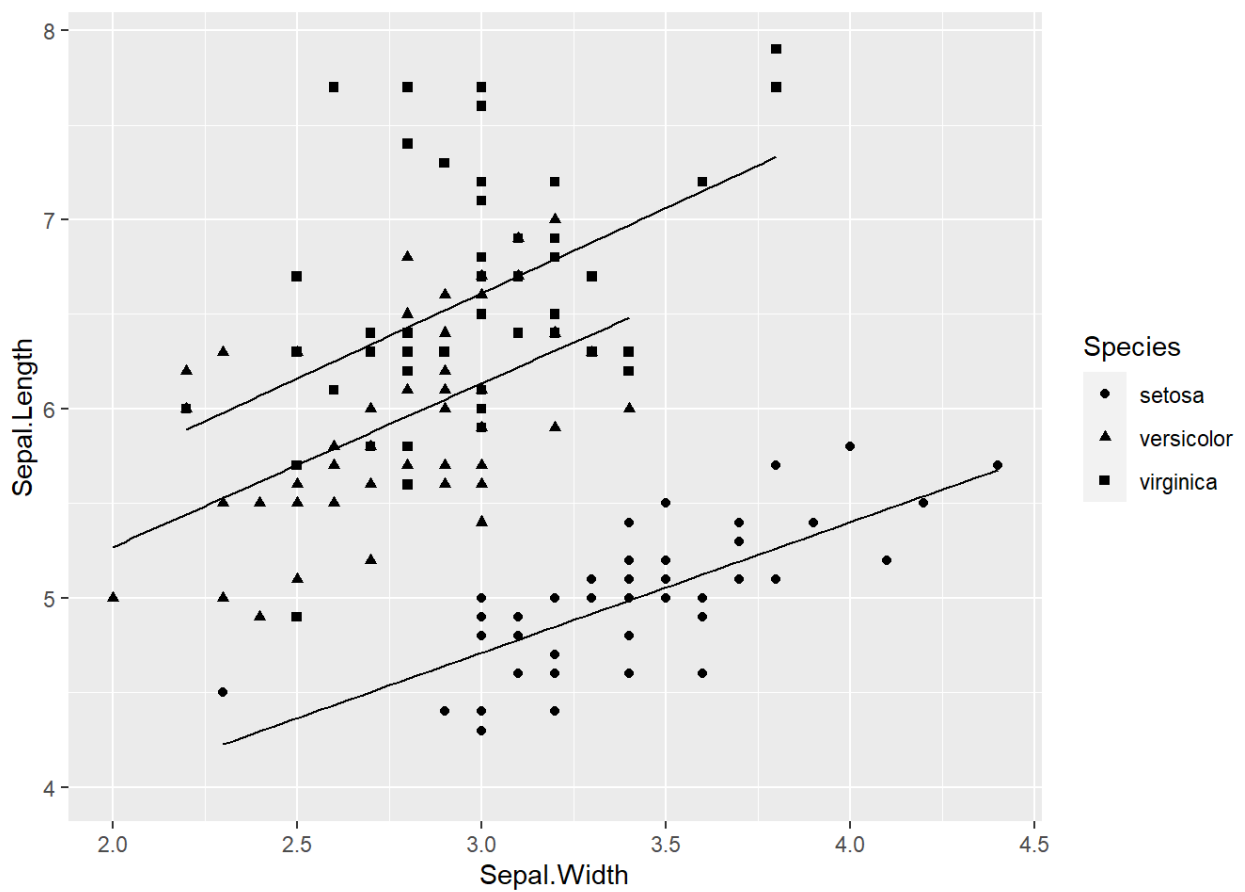
```
ggplot(data = iris) +  
  aes(x = Sepal.Width, y = Sepal.Length, color = Species) +  
  geom_point() +  
  geom_line(stat = 'smooth', method = 'lm')
```



De este gráfico surge que la relación en realidad es positiva para todas las especies!

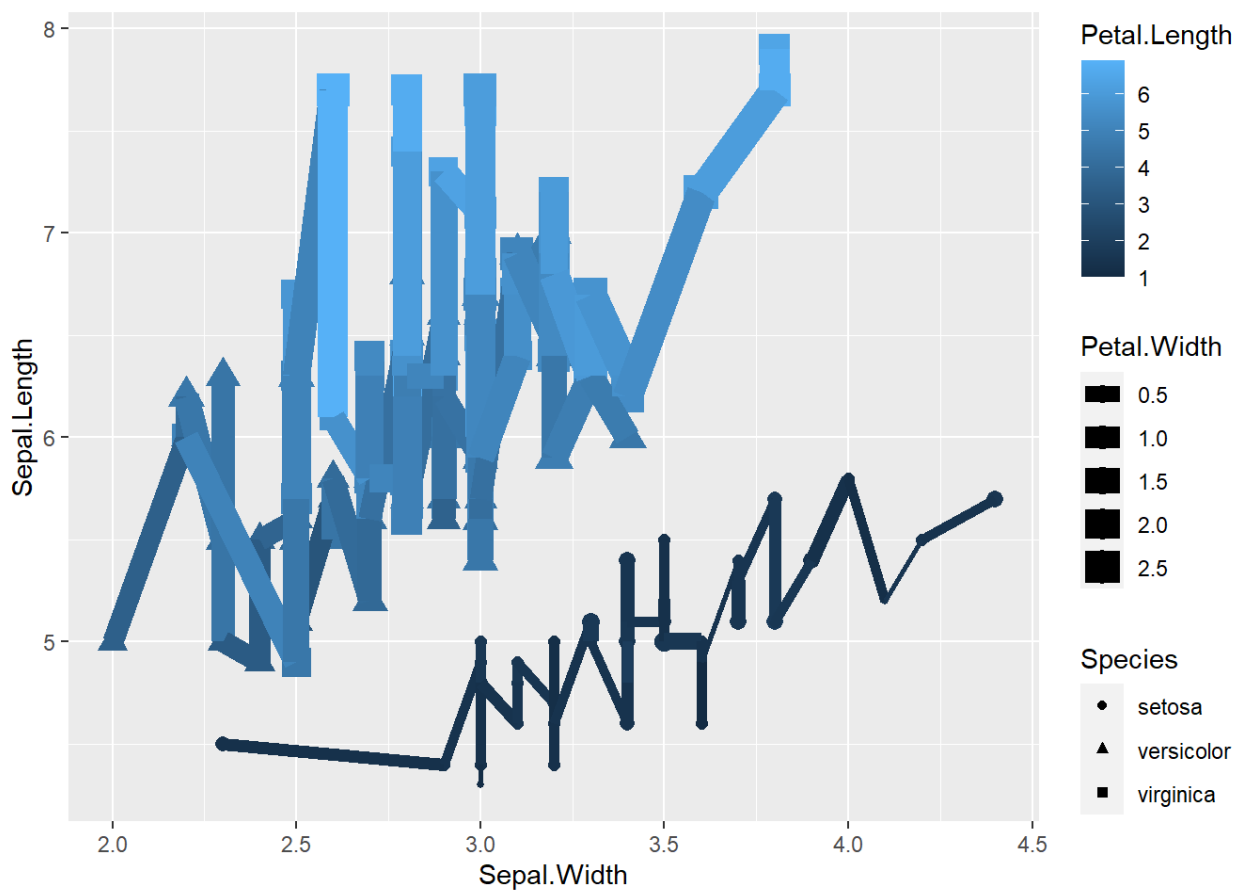
Dependiendo el tipo de `geom` tenemos distintos atributos estéticos para explorar: `color` y `alpha` (transparencia) para todos, `shape` y `size` para puntos, `linewidth` y `linetype` para líneas, y `fill` para barras, etc. Que tipo de atributo estético depende también de la naturaleza de la variable: continua o discreta. Veamos como queda mapear los valores de `Species` al atributo `shape` (forma):

```
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length, shape = Species) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm')
```



Las estéticas se pueden combinar para mostrar mas relaciones entre variables. Por ejemplo, además de `shape = Species` podríamos agregar la información de los pétalos como tamaño y color:

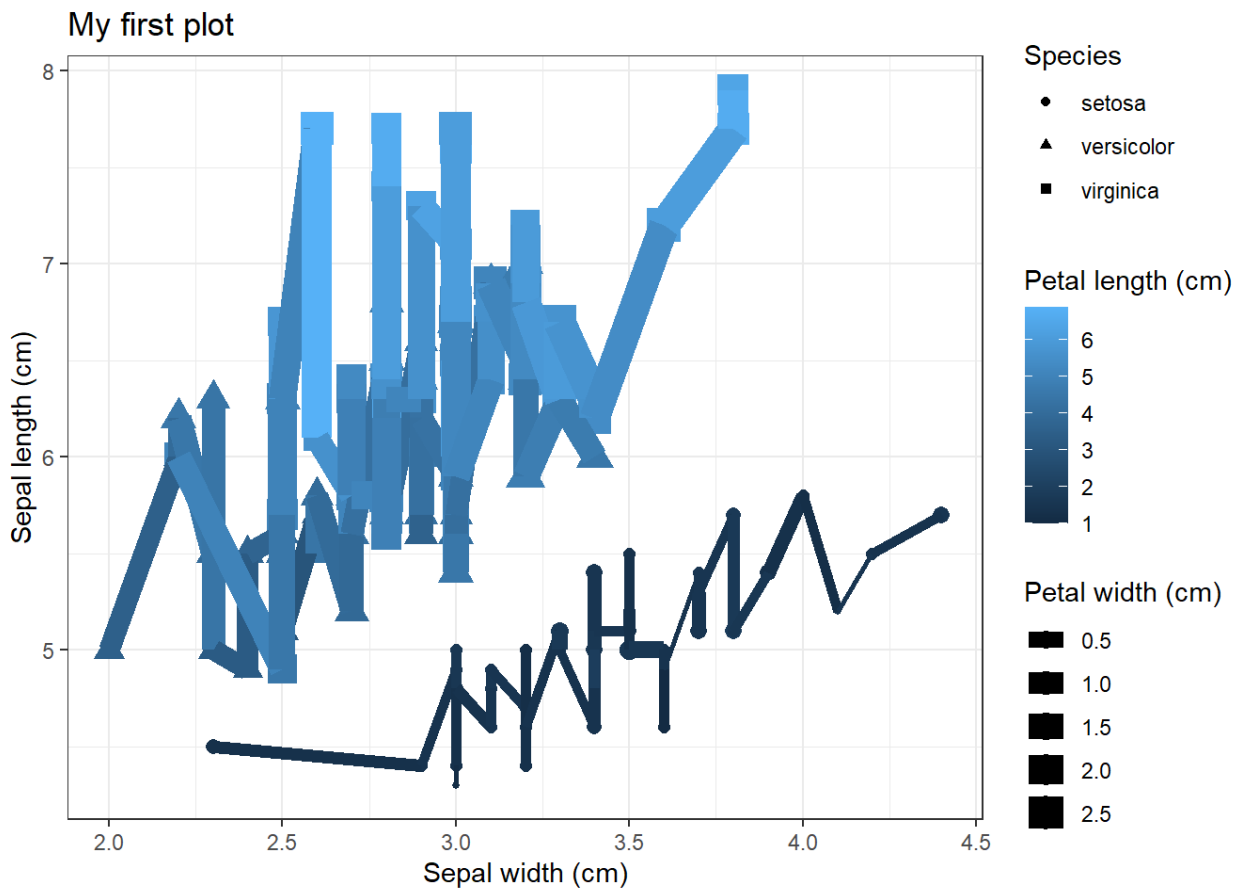
```
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length, shape = Species,
      size = Petal.Width, color = Petal.Length) +
  geom_point() +
  geom_line()
```



Claramente esto es una exageración pero muestra la potencialidad del gráfico. Siempre hay que tener en cuenta los principios vistos al principio, balance entre simplicidad del gráfico y la cantidad de información que queremos comunicar.

Finalmente vamos a ver como mejorar los nombres de los ejes, leyendas y agregar un título. Esto lo hacemos con `labs()`. También agregamos algún tema predefinido o como `theme_bw()`.

```
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length, shape = Species,
      size = Petal.Width, color = Petal.Length) +
  geom_point() +
  geom_line() +
  labs(x = "Sepal width (cm)", y = "Sepal length (cm)",
      color = "Petal length (cm)", size = "Petal width (cm)",
      shape = "Species", title = "My first plot") +
  theme_bw()
```



3.4 Geometrías: los bloques para construir gráficos

Existen muchas geometrías disponibles en el paquete `ggplot2`. El sitio web de referencia (<https://ggplot2.tidyverse.org/reference/>) contiene el listado exhaustivo de las opciones que tenemos para representar distintos tipos de datos. Como vimos antes, estos `geoms` son atajos del comando `layer()` para agregar capas a nuestro gráfico combinando `data`, `mapping`, `stats` y `position`. Pueden usarse como unico layer del gráfico o bien combinarse para crear visualizaciones mas complejas.

Algunos de `geoms` más comunes son:

- `geom_point()`, lo vimos anteriormente, cada dato se representa por un punto en el sistema de coordenas. Una variante es `geom_jitter()` que permite agregar una variabilidad aleatoria para evitar que se superpongan los puntos cuando el eje x es discreto.
- `geom_line()` dibujan líneas uniendo puntos de izquierda a derecha. Una variante es `geom_path()` que une puntos en función de su orden en el set de datos. Las líneas se utilizan normalmente para explorar cómo cambian las cosas con el tiempo.
- `geom_smooth()` permite ajustar distintos modelos para mostrar la tendencia de los datos en forma de linea con opción a agregar información de los errores. Es un atajo para combinar `geom_line()` y `stat_smooth()`.
- `geom_boxplot()` produce un diagrama de caja y bigotes para resumir la distribución de un conjunto de datos usando medidas de resumen robustas (mediana, cuartiles, etc.) y detectar valores atípicos.
- `geom_histogram()` y `geom_freqpoly()` muestran la distribución de variables continuas agrupándola en clases o intervalos. Si hay muchos datos `geom_density()` puede ser una alternativa.

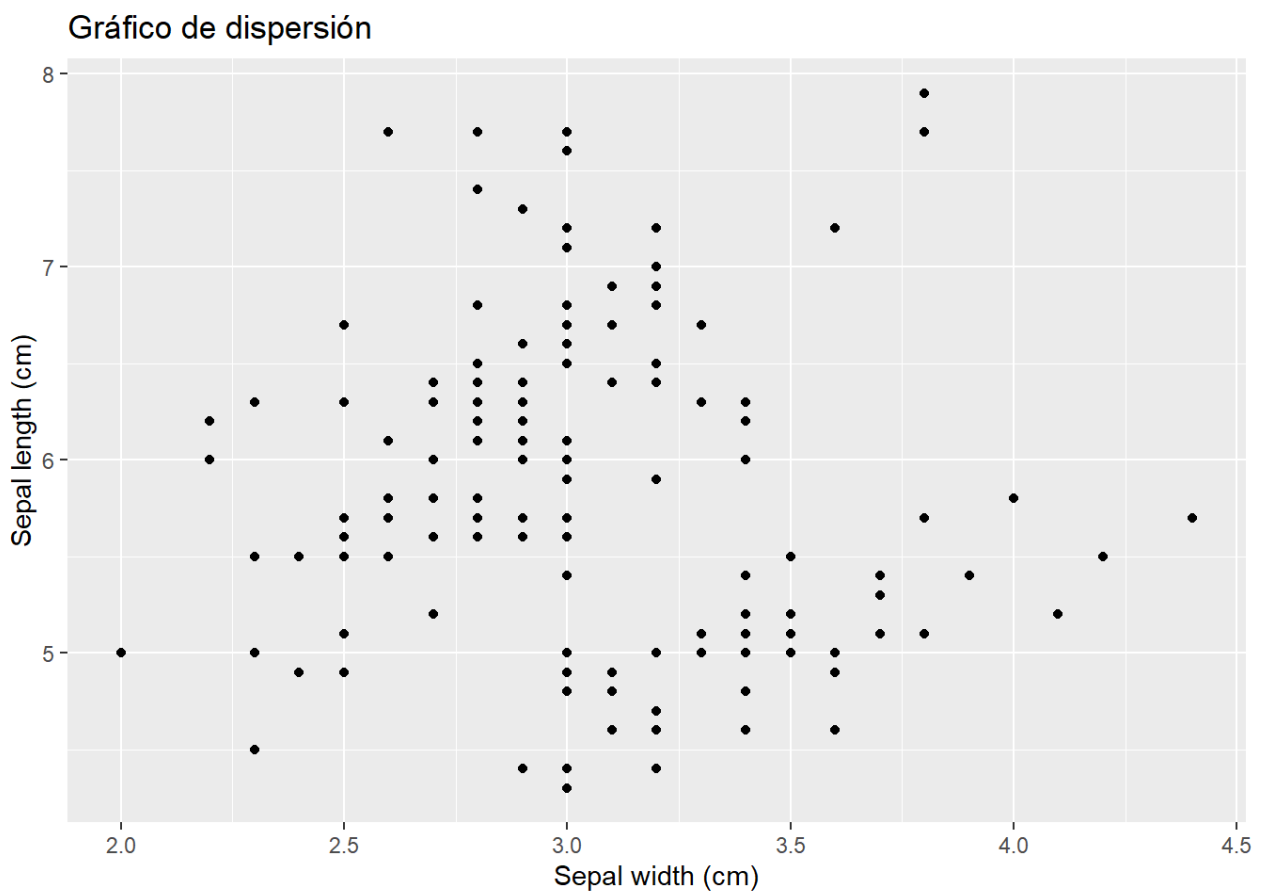
- `geom_bar()` y `geom_col()` nos permiten hacer gráficos de barras apiladas o a la par. La última es un atajo de la primera. Sirve para graficar frecuencias de variables categóricas o medidas de resumen de variables continuas por grupos.

Veamos en detalle estos gráficos y algunos más avanzados.

3.4.1 Gráficos de puntos

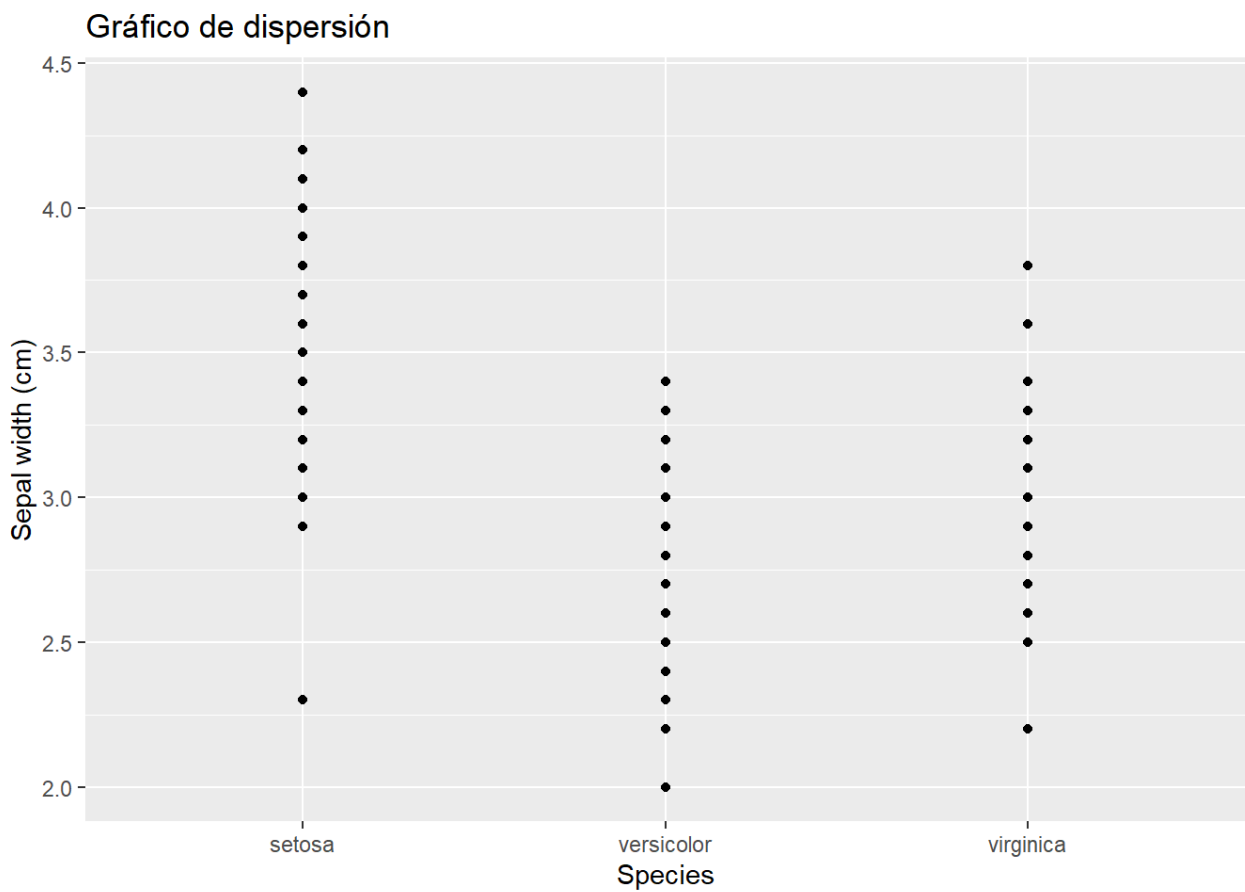
Este es el clásico gráfico de dispersión donde podemos representar como mínimo dos variables continuas que definen un espacio de coordenadas. Esto lo vimos al graficar la relación entre `Sepal.Width` y `Sepal.Length`

```
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length) +
  geom_point() +
  labs(x = "Sepal width (cm)", y = "Sepal length (cm)",
       title = "Gráfico de dispersión")
```



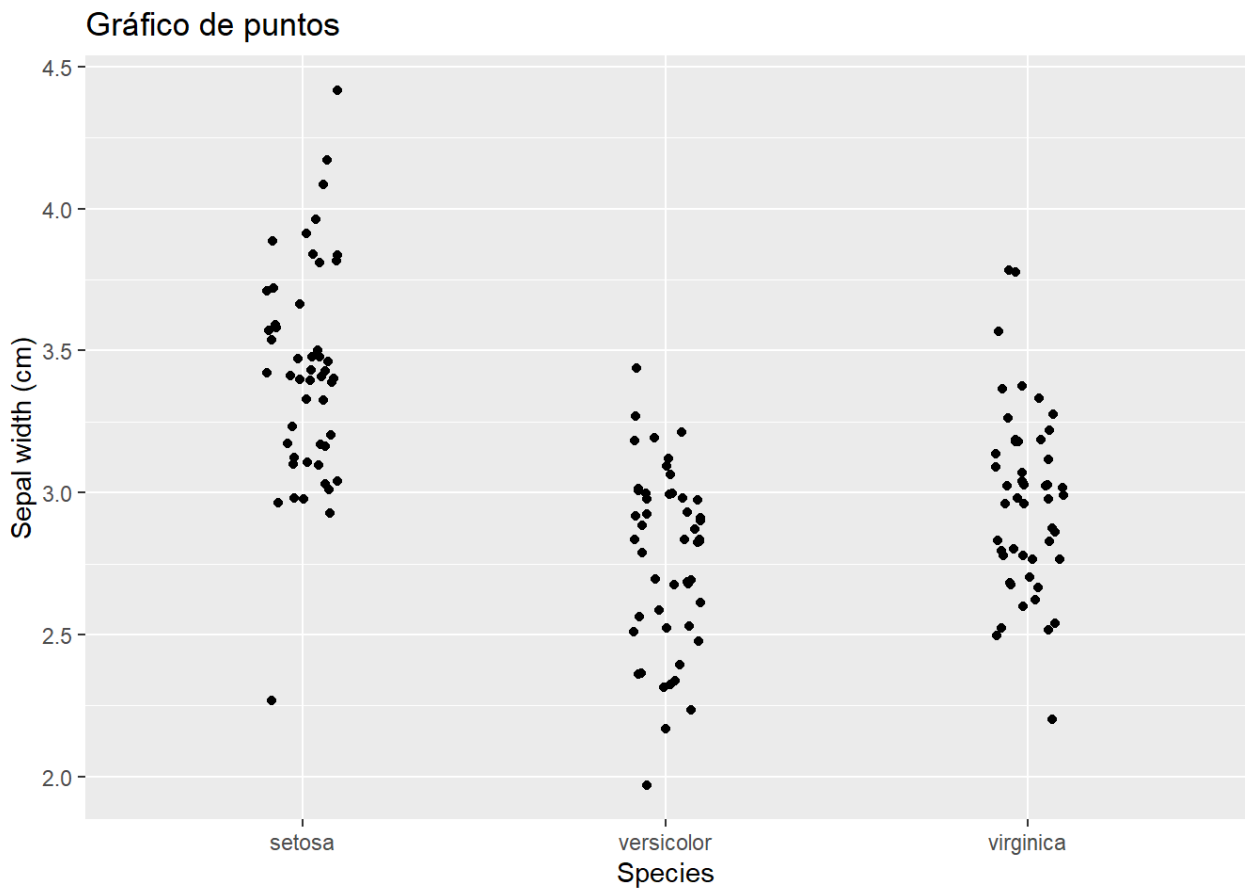
Alternativamente este gráfico de puntos puede mostrar la relación de una variable cuantitativa (e.g. `Sepal.Width`) con alguna cualitativa (e.g. `Species`)

```
ggplot(data = iris) +
  aes(x = Species, y = Sepal.Width) +
  geom_point() +
  labs(x = "Species", y = "Sepal width (cm)",
       title = "Gráfico de dispersión")
```



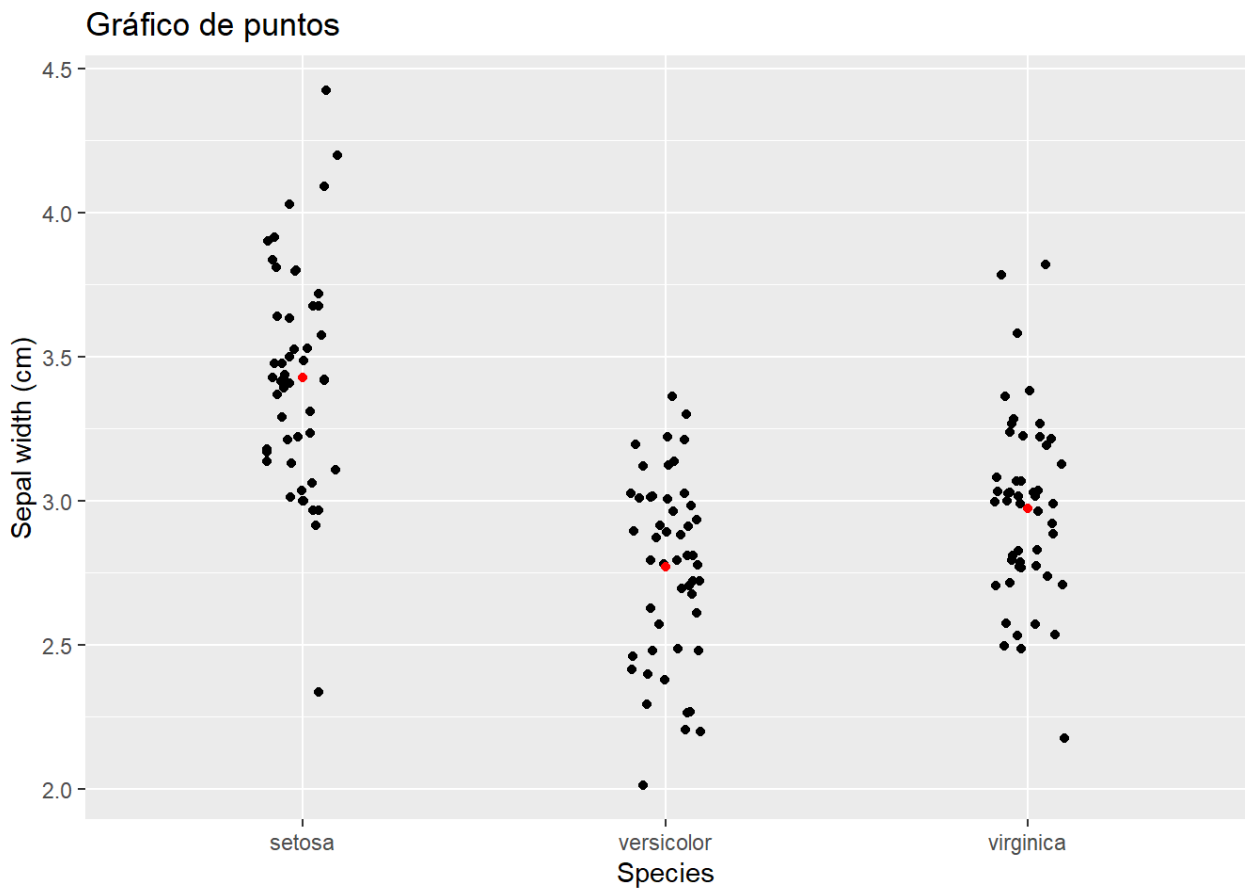
En este caso tiene sentido agregar un ruido aleatorio a la posición sobre el eje X para evitar la superposición de puntos.

```
ggplot(data = iris) +  
  aes(x = Species, y = Sepal.Width) +  
  geom_jitter(width = 0.1) +  
  labs(x = "Species", y = "Sepal width (cm)",  
        title = "Gráfico de puntos")
```



Sobre este gráfico podríamos agregar una estadística de resumen, por ejemplo la media:

```
ggplot(data = iris) +  
  aes(x = Species, y = Sepal.Width) +  
  geom_jitter(width = 0.1) +  
  geom_point(stat = "summary", fun = mean, color = "red") +  
  labs(x = "Species", y = "Sepal width (cm)",  
       title = "Gráfico de puntos")
```



Este tipo de gráficos nos da pie para pensar en agregar otras medidas de resumen que es el gráfico que vemos a continuación.

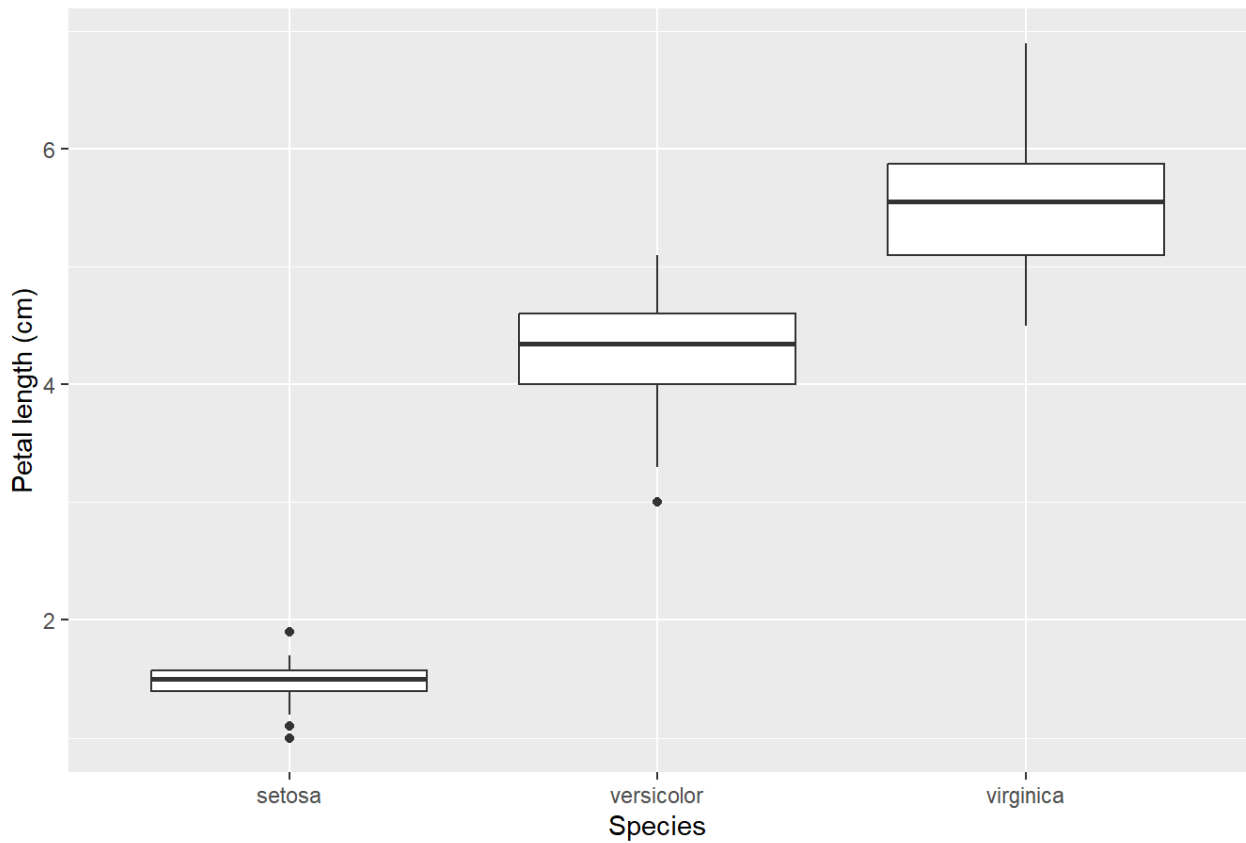
3.4.2 Boxplot o diagramas de cajas

Cuando un conjunto de datos incluye una variable categórica y una o más variables continuas, probablemente es de interés como cómo varían los valores de las variables continuas con los niveles de la variable categórica. El gráfico anterior de algún modo mostraba eso.

Cuando tenemos suficiente cantidad de datos para cada categoría, un gráfico muy útil es el gráfico de caja o *boxplot*. Este gráfico inventado por John Tukey (<https://vita.had.co.nz/papers/boxplots.pdf>) hace 40 años pero sigue siendo vigente y muy potente. Se construye a partir de medidas posición robustas: mediana, cuartiles, etc. Permite visualizar dispersión de los datos, la tendencia central y detectar valores atípicos. Veamos un ejemplo con `Petal.Length`.

```
ggplot(data = iris) +
  aes(x = Species, y = Petal.Length) +
  geom_boxplot() +
  labs(x = "Species", y = "Petal length (cm)",
       title = "Boxplot")
```

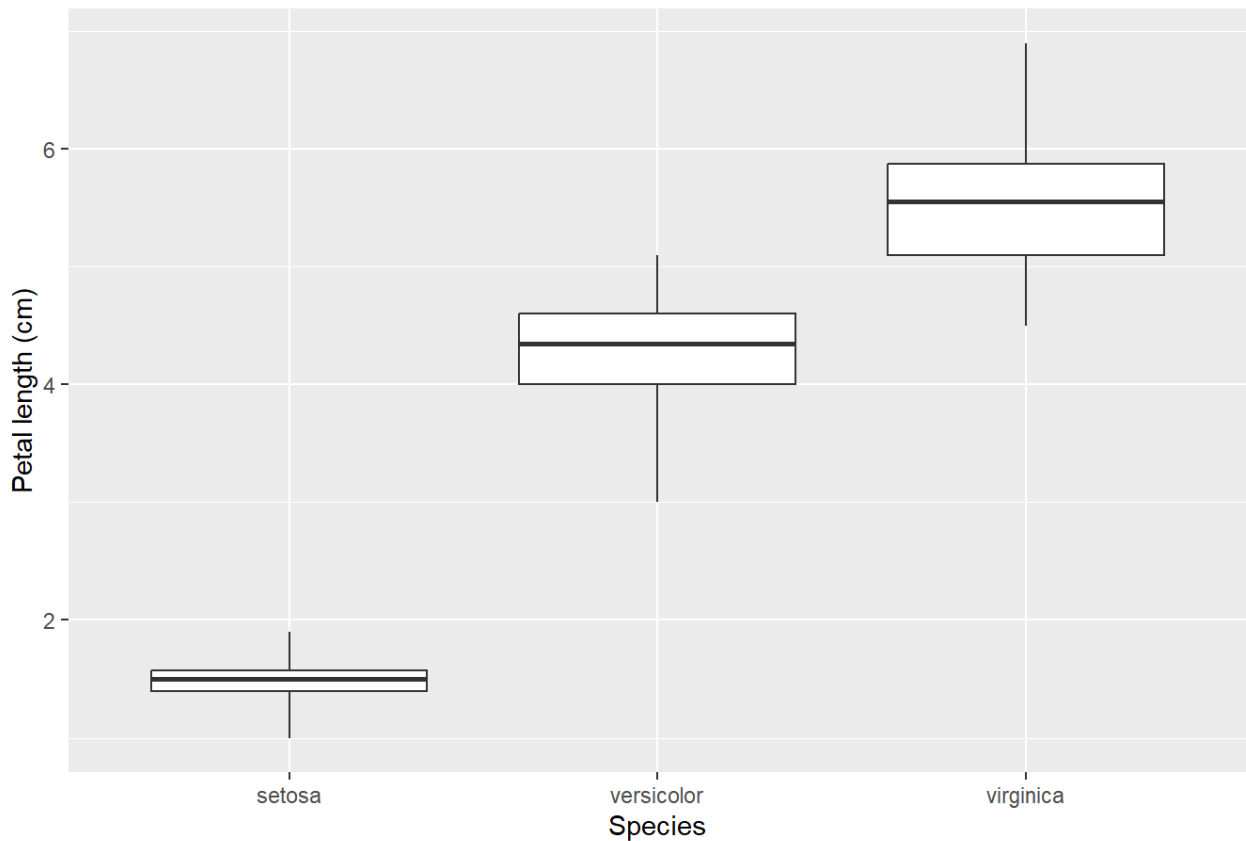
Boxplot



Por defecto `geom_boxplot()` usa un factor de escala de 1.5 y muestra los valores atípicos y *outliers* leves y extremos. Si solo queremos aquellos outliers *extremos*, es necesario indicar `coef = 3`

```
ggplot(data = iris) +  
  aes(x = Species, y = Petal.Length) +  
  geom_boxplot(coef = 3) +  
  labs(x = "Species", y = "Petal length (cm)",  
       title = "Boxplot")
```

Boxplot

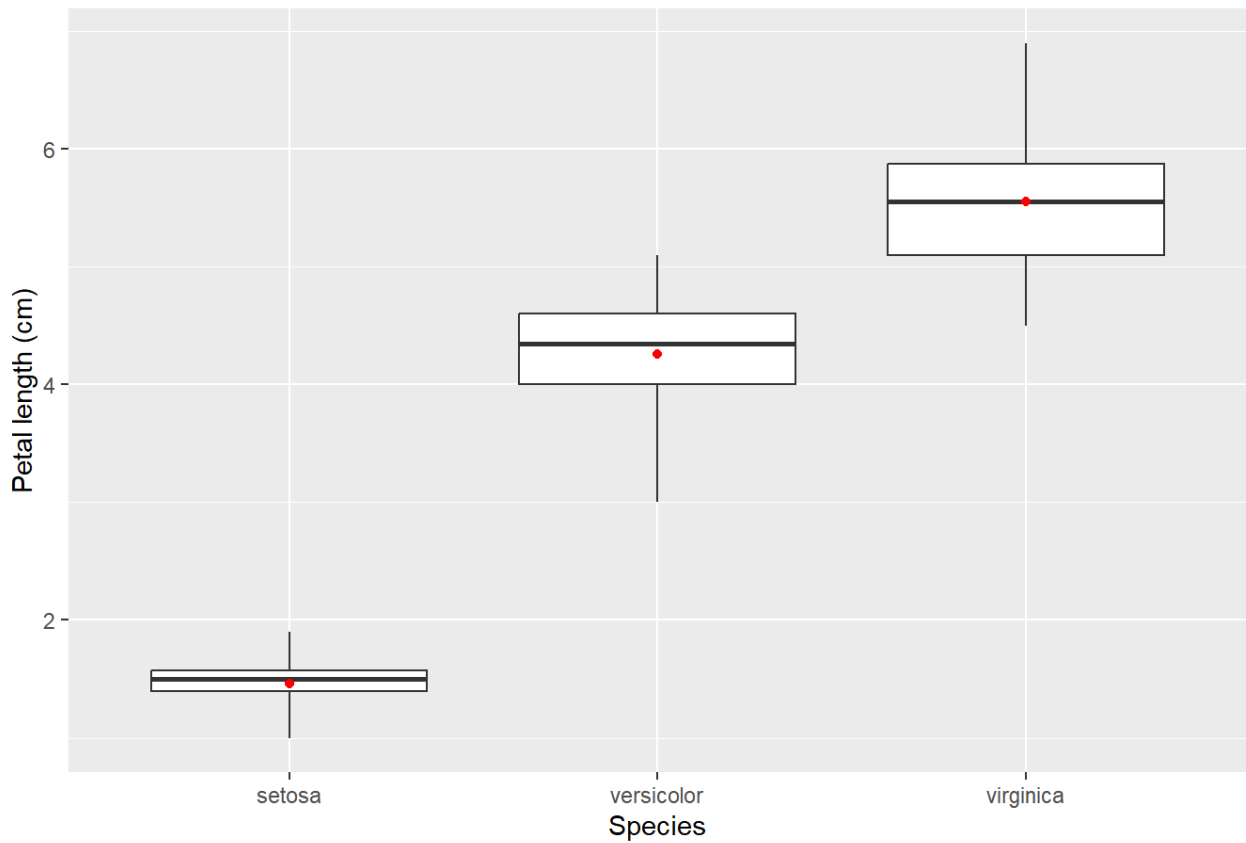


Como vemos los valores outliers antes indicados desaparecen porque son valores atípicos leves, i.e. están entre las vallas internas y externas.

El gráfico de caja puede combinarse con otros geoms, por ejemplo, agregando la media con `stat = 'summary', fun = mean`:

```
ggplot(data = iris) +  
  aes(x = Species, y = Petal.Length) +  
  geom_boxplot(coef = 3) +  
  geom_point(stat = "summary", fun = mean, color = "red") +  
  labs(x = "Species", y = "Petal length (cm)",  
       title = "Boxplot")
```

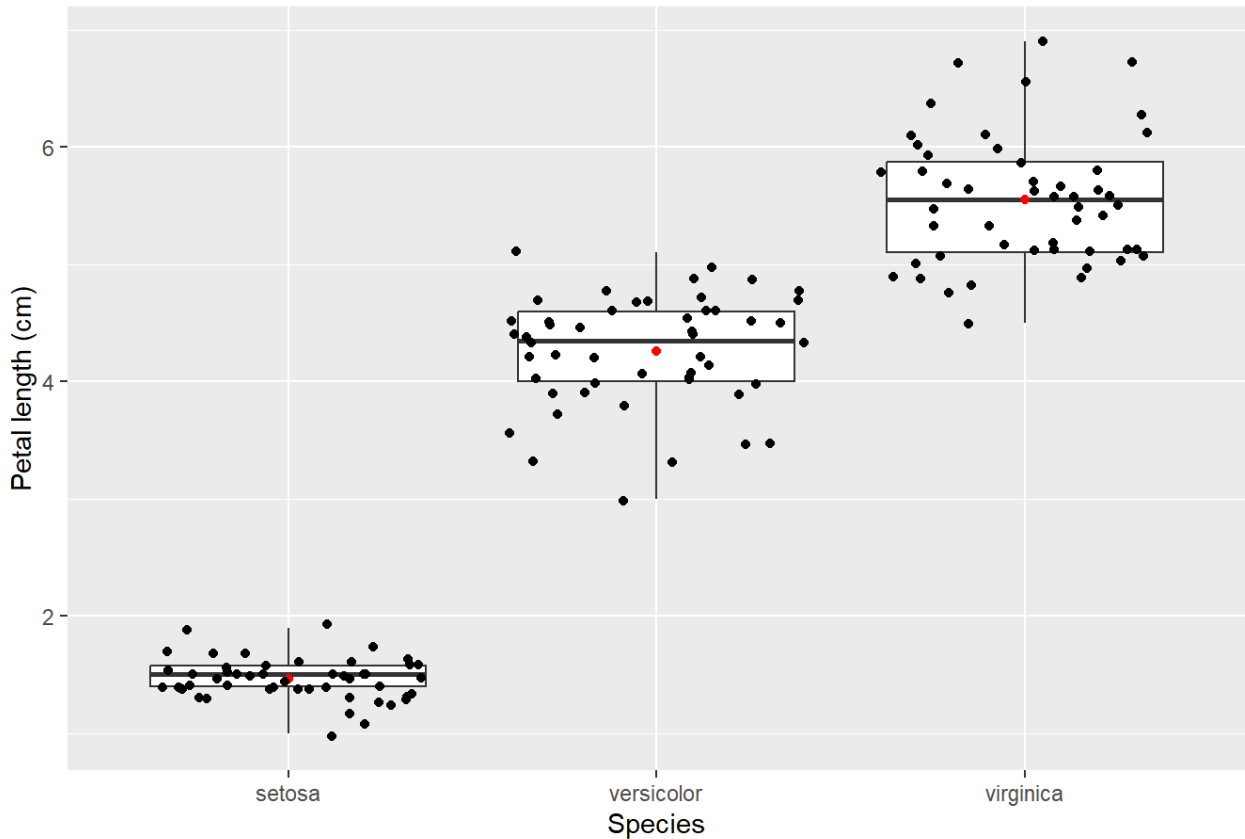
Boxplot



O podemos agregarle los puntos observados, con un poco de desplazamiento que vimos en `geom_jitter()` :

```
ggplot(data = iris) +  
  aes(x = Species, y = Petal.Length) +  
  geom_boxplot(coef = 3) +  
  geom_point(stat = "summary", fun = mean, color = "red") +  
  geom_jitter() +  
  labs(x = "Species", y = "Petal length (cm)",  
        title = "Boxplot recargado")
```

Boxplot recargado

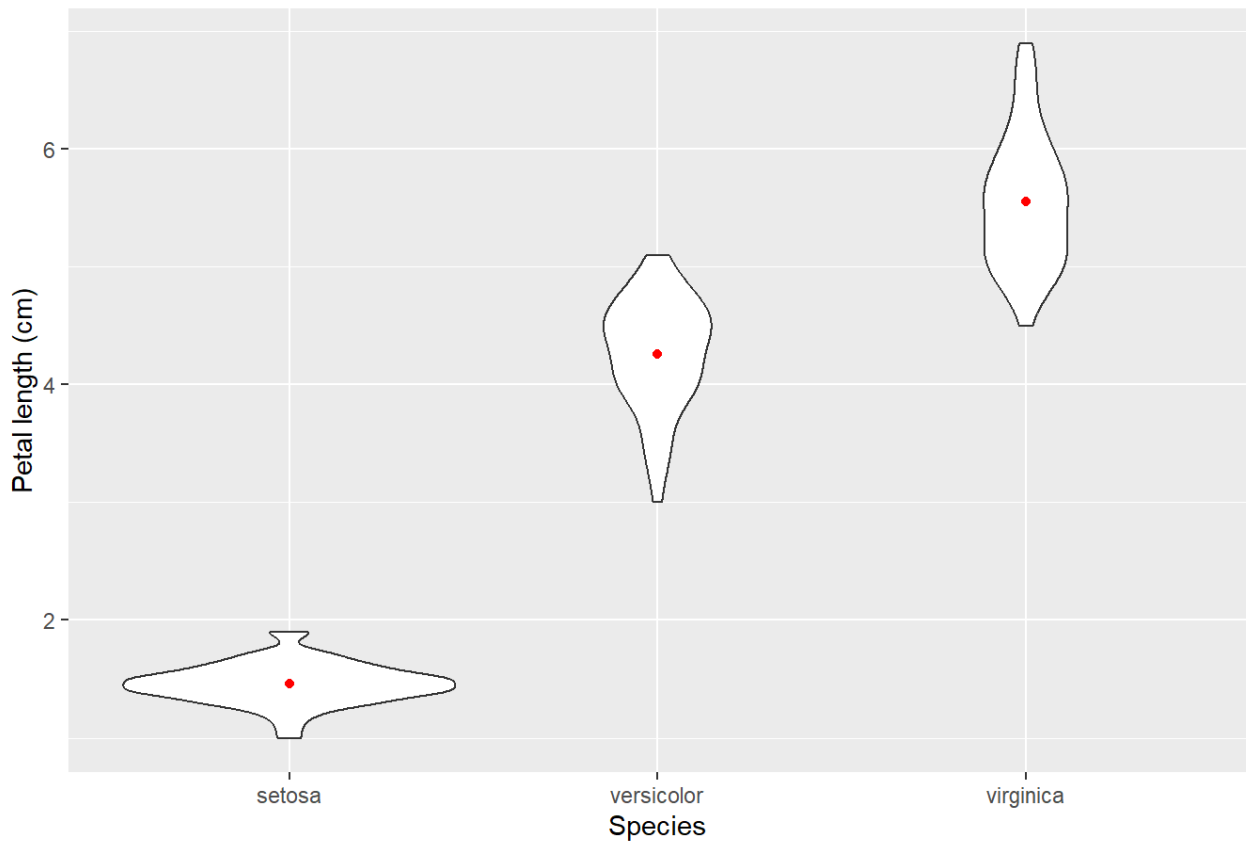


3.4.3 Gráfico de tipo violín

Siguiendo con la idea de ver la distribución de los datos, otra alternativa muy popular, sobre todo cuando los datos son muchos, es representar de manera compacta la *densidad* de la distribución, destacando las áreas donde se encuentran más puntos.

```
ggplot(data = iris) +  
  aes(x = Species, y = Petal.Length) +  
  geom_violin() +  
  geom_point(stat = "summary", fun = mean, color = "red") +  
  labs(x = "Species", y = "Petal length (cm)",  
       title = "Violin plot")
```


Violin plot



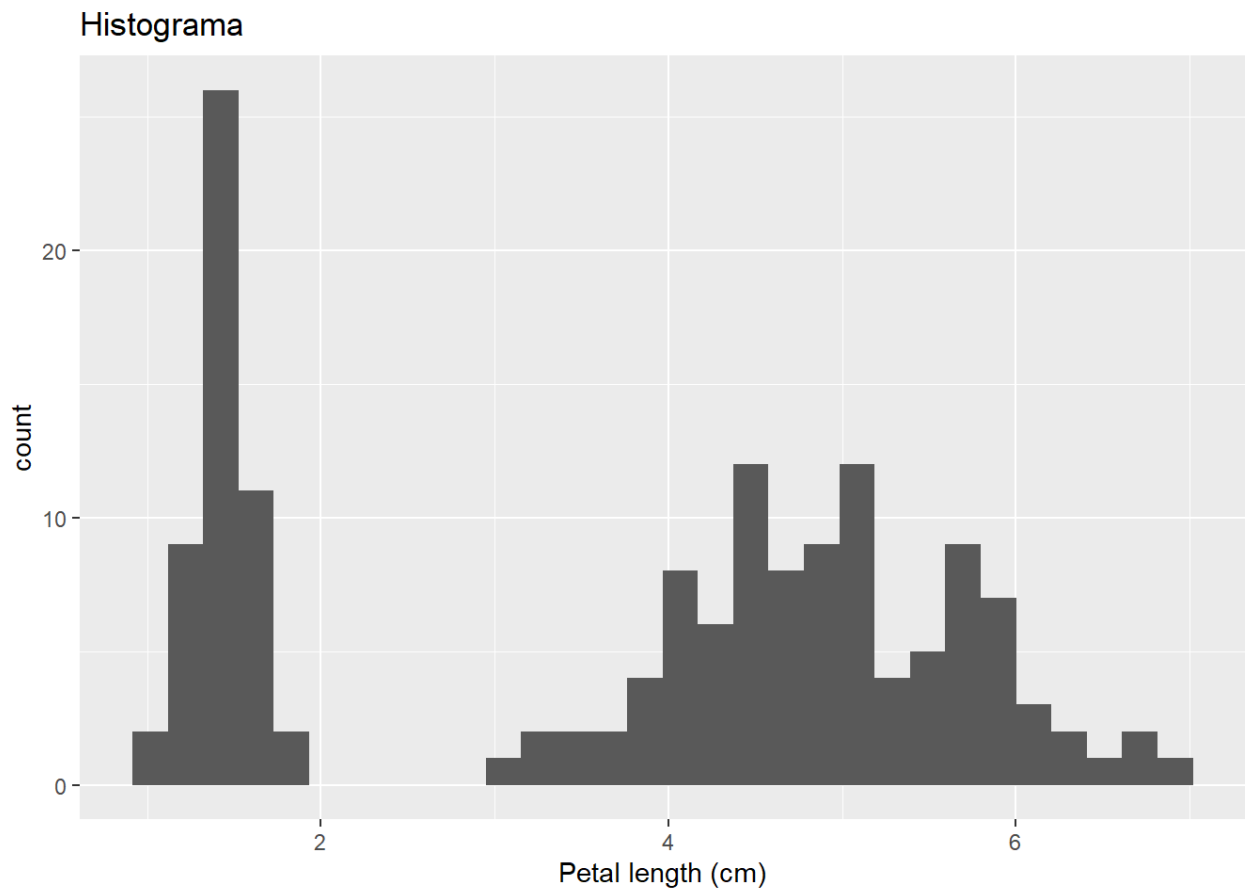
En este gráfico, al igual que en el boxplot, se ve claramente donde se concentran los datos.

3.4.4 Histogramas, polígonos de frecuencia y densidad

Una forma clásica de ver la distribución de los datos cuantitativos es el *histograma*. Esto se logra agrupando las observaciones en en clases o intervalos y luego contando la cantidad de observaciones dentro de cada intervalo. De esta manera en el eje X se tiene el rango de valores y en el eje Y se representan las frecuencias, que pueden ser absolutas o relativas (%).

Veamos como se distribuye la longitud de los pétalos primero ignorando la especie:

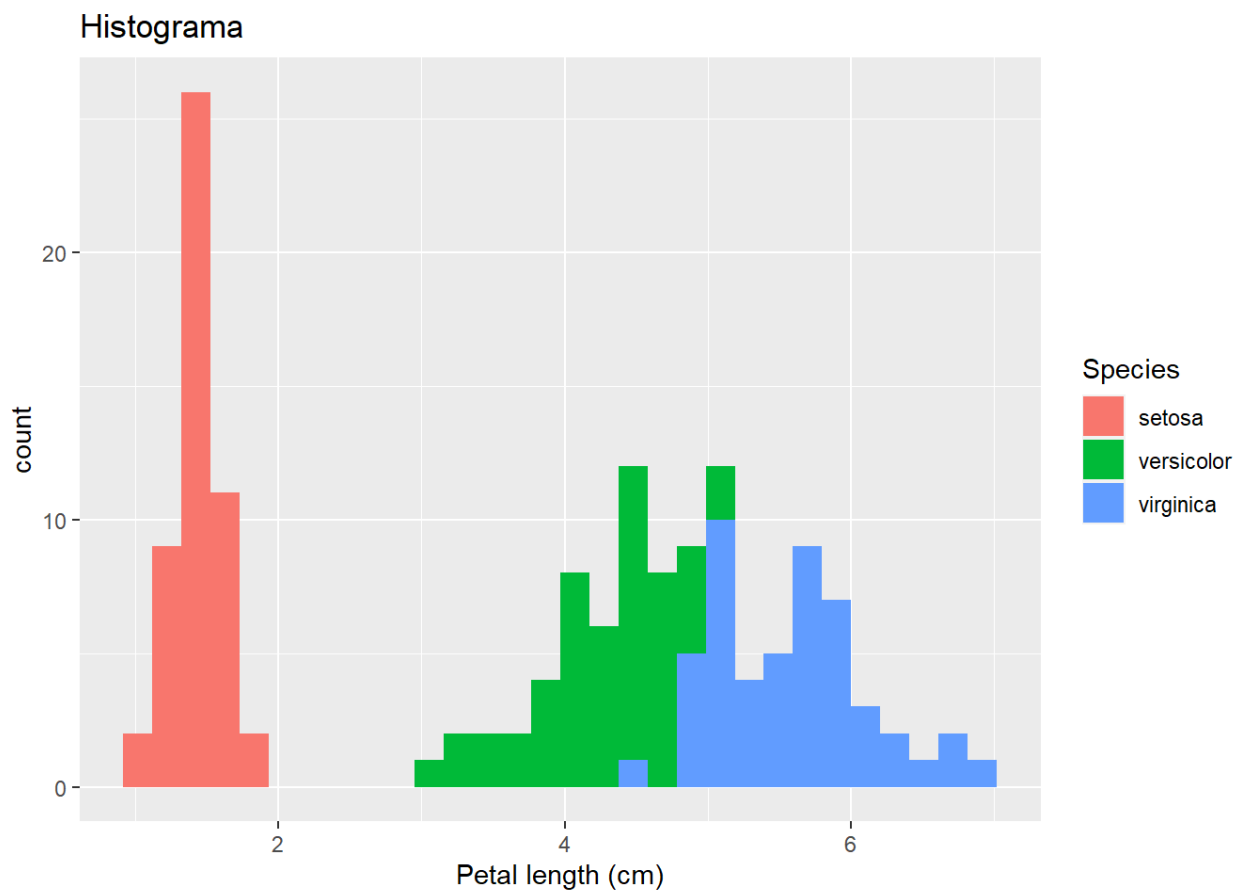
```
ggplot(data = iris) +  
  aes(x = Petal.Length) +  
  geom_histogram() +  
  labs(x = "Petal length (cm)",  
       title = "Histograma")
```



Aquí vemos que los datos tienen al menos 2 grupos bien definidos. Usando el argumento `binwidth` y `bins` podemos controlar el ancho y el número de clases.

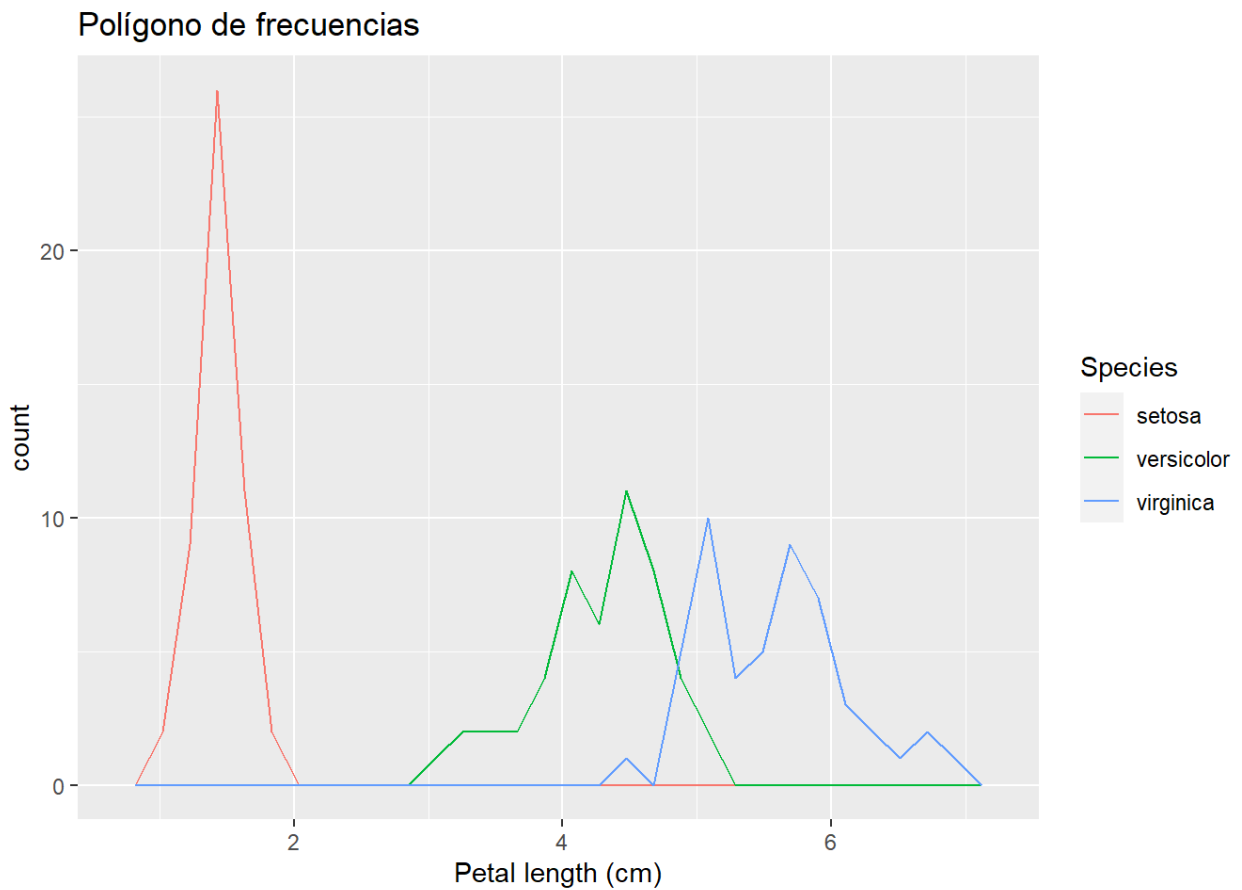
Si agregamos la especie usando `fill` o `color` podemos diferenciar entre *S. versicolor* y *S. virginica*.

```
ggplot(data = iris) +  
  aes(x = Petal.Length, fill = Species) +  
  geom_histogram() +  
  labs(x = "Petal length (cm)", fill = "Species",  
       title = "Histograma")
```



Cuando los grupos se superponen esta visualización no es muy útil. Una alternativa es usar `geom_freqpoly()` para obtener un contorno del histograma que permita ver grupos superpuestos.

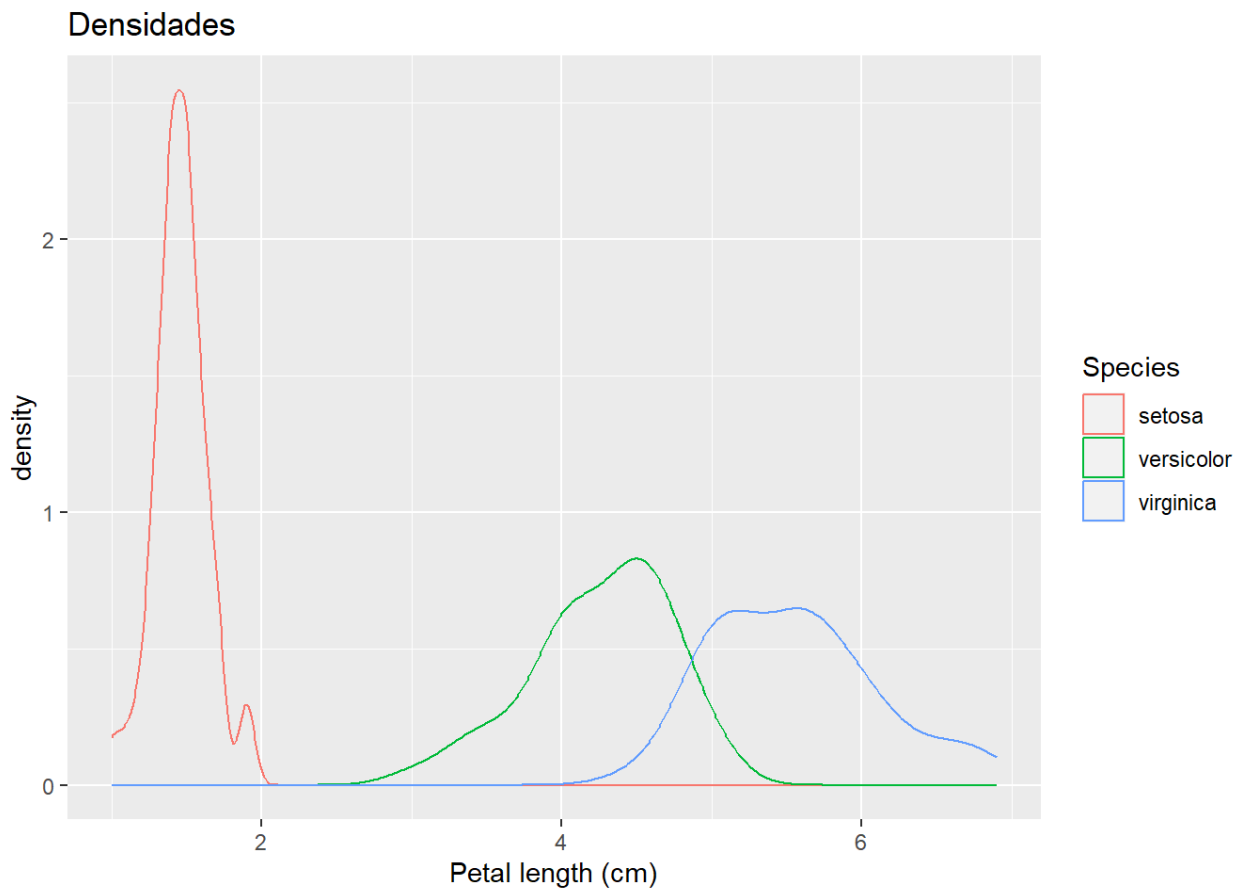
```
ggplot(data = iris) +  
  aes(x = Petal.Length, color = Species) +  
  geom_freqpoly() +  
  labs(x = "Petal length (cm)", fill = "Species",  
        title = "Polígono de frecuencias")
```



Ahi vemos bien el detalle donde se superponen estas dos especies. La forma de los polígonos dependen del grado de detalle que le demos eligiendo el numero y ancho de clases.

Una alternativa al polígono de frecuencia es la gráfica de densidad, `geom_density()`. Estas gráficas de densidad son más difíciles de interpretar ya que los cálculos subyacentes son más complejos y ademas hacen suposiciones que no son ciertas para todos los datos, por ejemplo la distribución subyacente es continua, ilimitada y fluida.

```
ggplot(data = iris) +  
  aes(x = Petal.Length, color = Species) +  
  geom_density() +  
  labs(x = "Petal length (cm)", fill = "Species",  
       title = "Densidades")
```

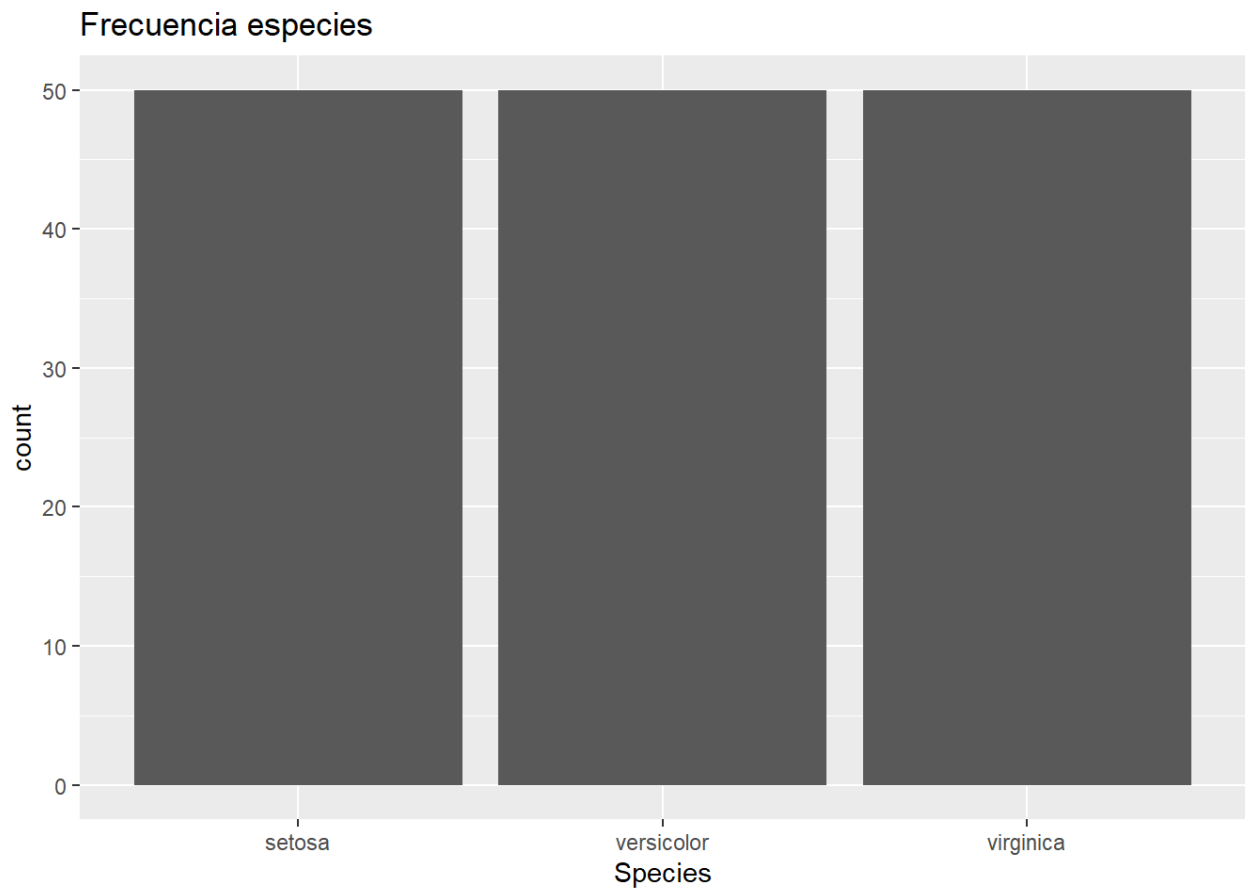


3.4.5 Graficos de barras o columnas

Estos tipos de gráficos se realizan con las funciones: `geom_bar()` y `geom_col()`. La diferencia radica en el tipo de `stat` que se calcula y representa en la altura de las barras.

`geom_bar()` hace que la altura de la barra sea proporcional al número de casos en cada grupo (o si se proporciona la estética del peso, la suma de los pesos) y por lo tanto usa `stat_count()`. Esto es de utilidad para cuando queremos hacer un gráfico de frecuencias de una variable discreta.

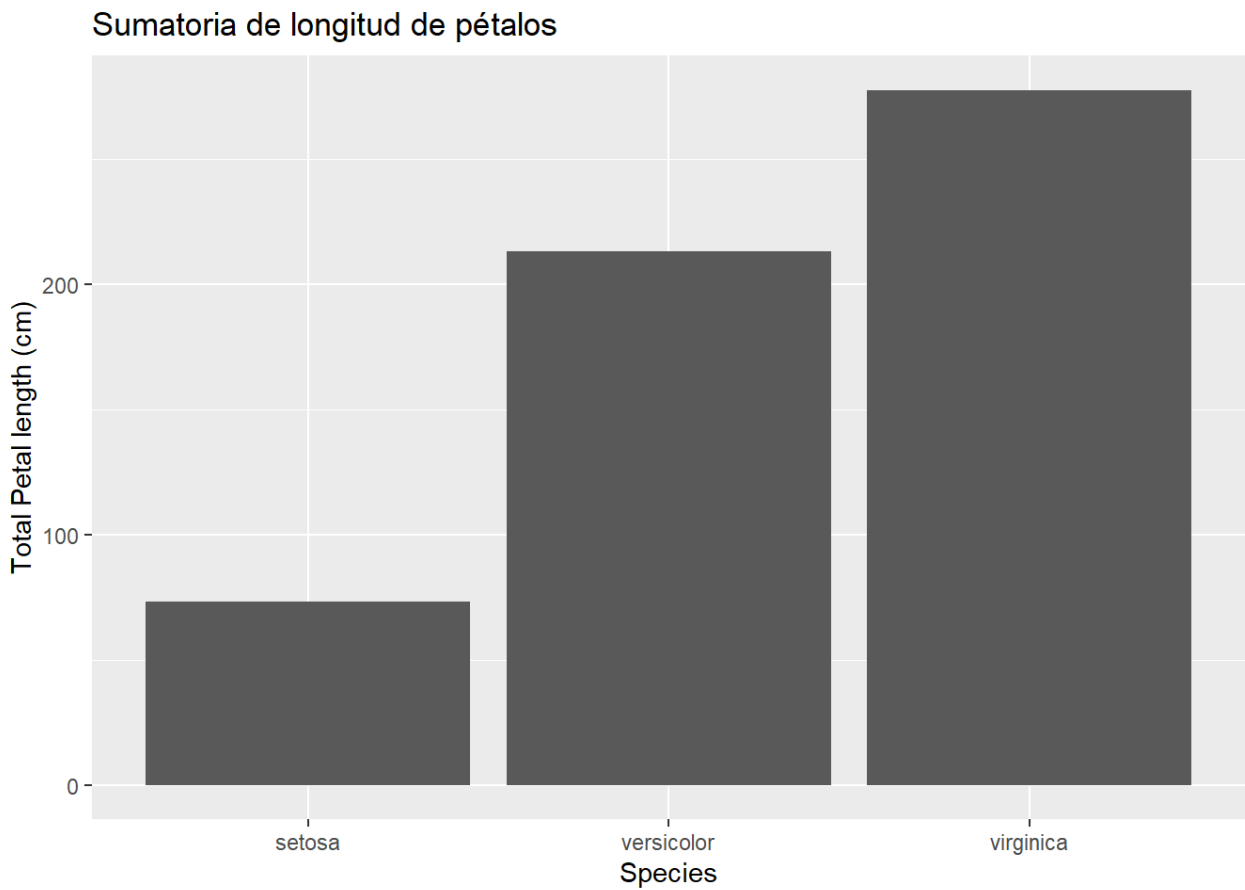
```
ggplot(data = iris) +  
  aes(x = Species) +  
  geom_bar() +  
  labs(title = "Frecuencia especies")
```



En este caso el gráfico es poco atractivo ya que el set de datos tiene 50 casos para cada especie.

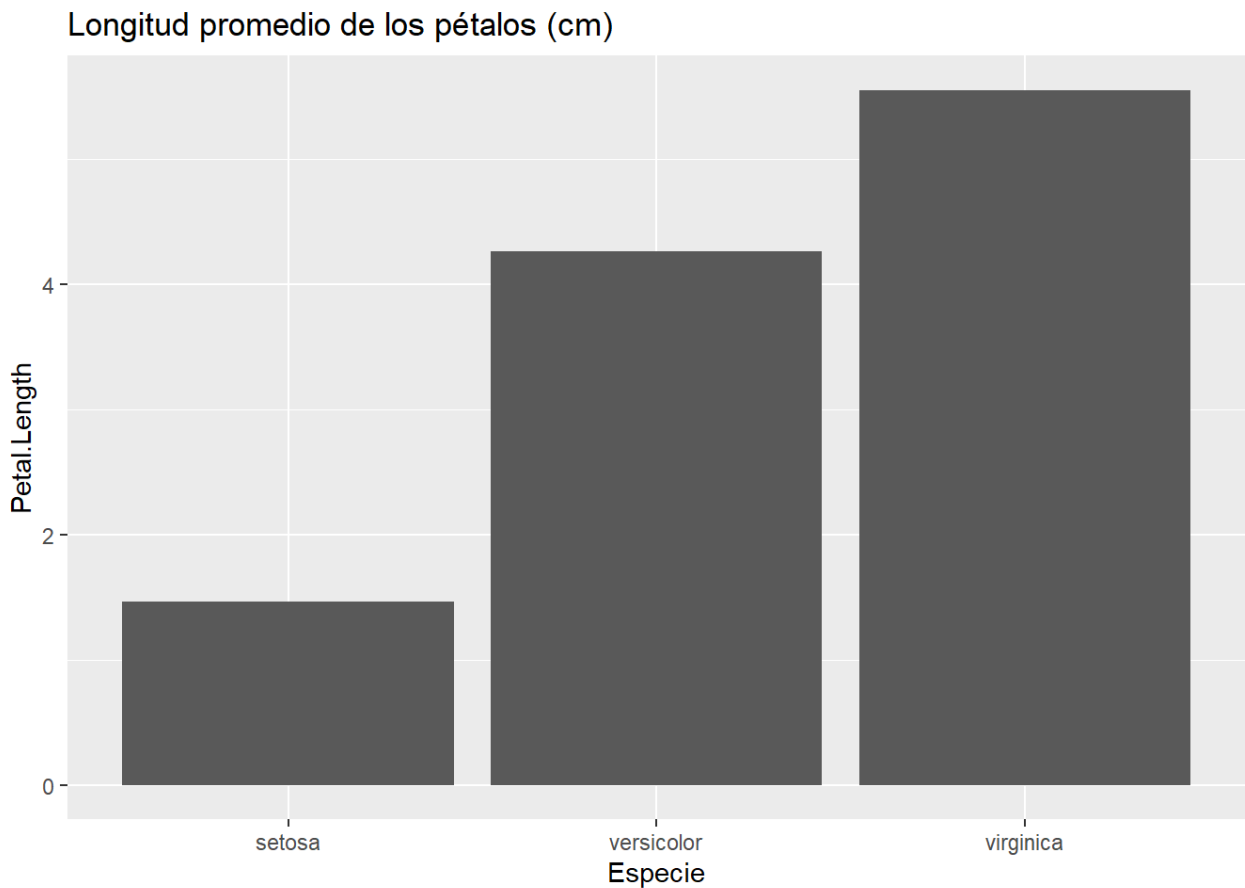
Por su parte, `geom_col()` sirve para representar datos únicos en la altura de las barras ya que usa `stat_identity()` y deja los datos como están. Si por cada valor de x tenemos varios datos éstos se van a agregar y se muestra el total.

```
ggplot(data = iris) +  
  aes(x = Species, y = Petal.Length) +  
  geom_col() +  
  labs(title = "Sumatoria de longitud de pétalos",  
        y = "Total Petal length (cm)")
```



Este gráfico a priori no tiene sentido ya que el eje y muestra la sumatoria de la longitud de pétalos de las 50 observaciones de cada especie. Para mostrar una medida de resumen se debe combinar `geom_bar()` con `stat_summary()`

```
ggplot(data = iris) +  
  aes(x = Species, y = Petal.Length) +  
  geom_bar(stat = "summary", fun = mean) +  
  labs(title = "Longitud promedio de los pétalos (cm)",  
        x = "Especie")
```



3.4.6 Graficos medias e intervalos

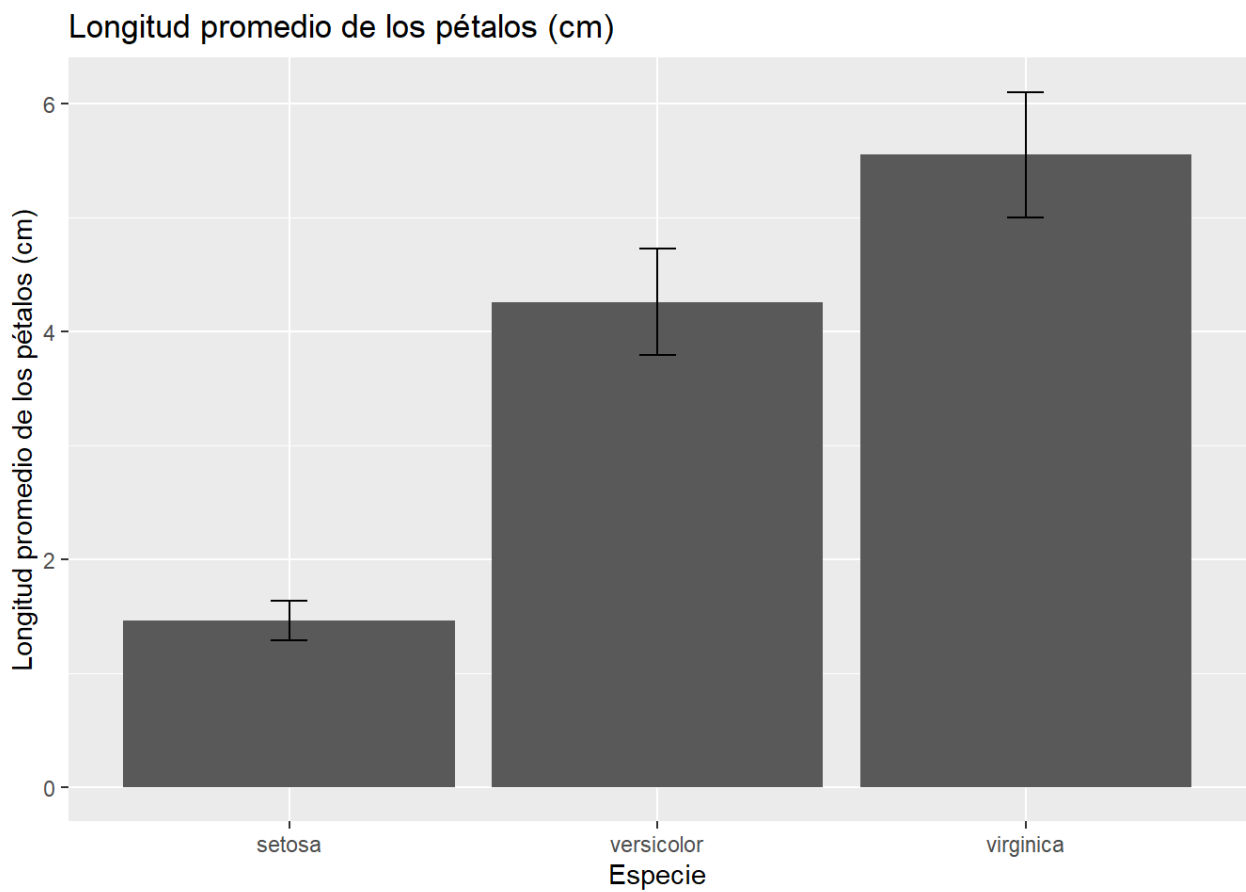
Una opción muy común es agregar una medida de variabilidad o incertidumbre cuando las barras o puntos representan medidas de tendencia central. Supongamos que queremos tener para cada especie el largo de pétalo medio y un intervalo que represente el desvío estándar respecto a esa medida de resumen. Primero necesitamos obtener las estadísticas descriptivas. Para esto podemos usar `dplyr`

```
library(dplyr)
meds_sd <- group_by(iris, Species) %>%
  summarise(ybar = mean(Petal.Length), s = sd(Petal.Length))
meds_sd
```

```
## # A tibble: 3 × 3
##   Species    ybar    s
##   <fct>    <dbl> <dbl>
## 1 setosa    1.46 0.174
## 2 versicolor 4.26 0.470
## 3 virginica 5.55 0.552
```

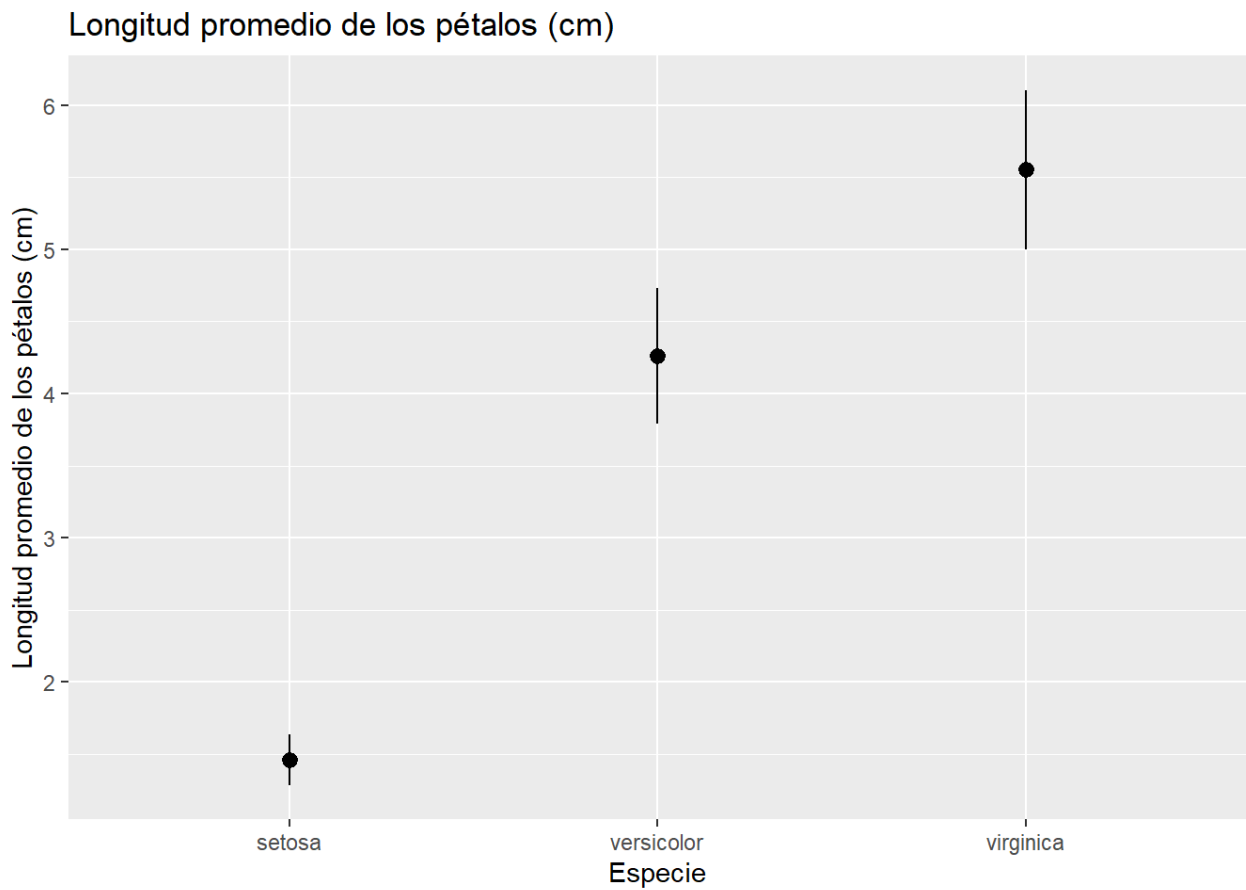
Luego usamos ese nuevo data frame para construir la visualización con `geom_col()` mas un layer `geom_errorbar()`. Este nuevo layer agrega más atributos estéticos para mapear.

```
ggplot(data = meds_sd) +
  aes(x = Species, y = ybar, ymin = ybar - s, ymax = ybar + s) +
  geom_col() +
  geom_errorbar(width = 0.1) +
  labs(title = "Longitud promedio de los pétalos (cm)",
       x = "Especie", y = "Longitud promedio de los pétalos (cm)")
```

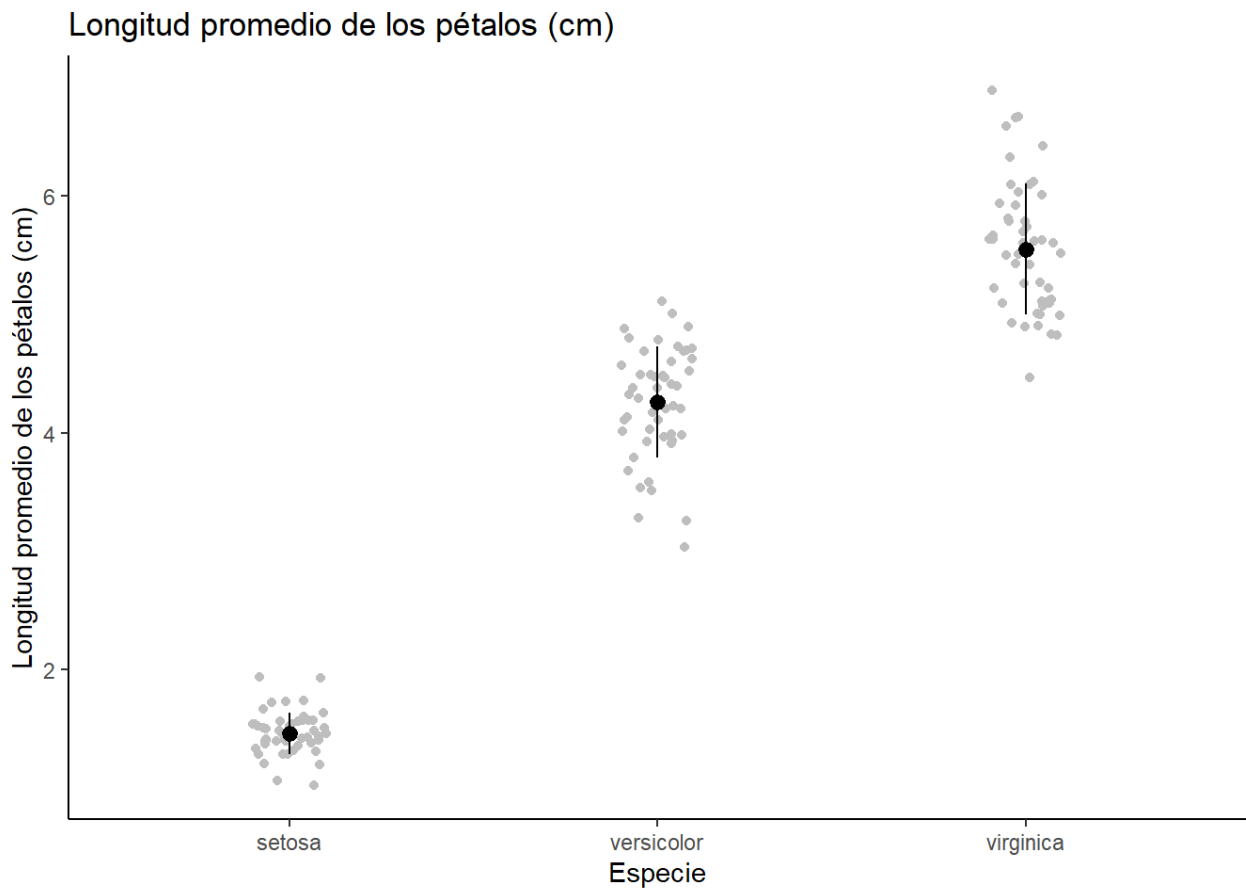
Otra opción es usar la geometría `geom_pointrange()` :

```
ggplot(data = meds_sd) +  
  aes(x = Species, y = ybar, ymin = ybar - s, ymax = ybar + s) +  
  geom_pointrange() +  
  labs(title = "Longitud promedio de los pétalos (cm)",  
        x = "Especie", y = "Longitud promedio de los pétalos (cm)")
```



Finalmente podríamos agregar sobre este gráfico los puntos originales con un color distinto. Como combinamos dos set de datos distintos conviene definir `data` y `aes` en cada layer. De paso probemos el tema `theme_classic()`.

```
ggplot() +
  geom_jitter(
    data = iris,
    aes(x = Species, y = Petal.Length),
    color = "gray", width = 0.1
  ) +
  geom_pointrange(
    data = meds_sd,
    aes(x = Species, y = ybar, ymin = ybar - s, ymax = ybar + s)
  ) +
  labs(title = "Longitud promedio de los pétalos (cm)",
        x = "Especie", y = "Longitud promedio de los pétalos (cm)") +
  theme_classic()
```



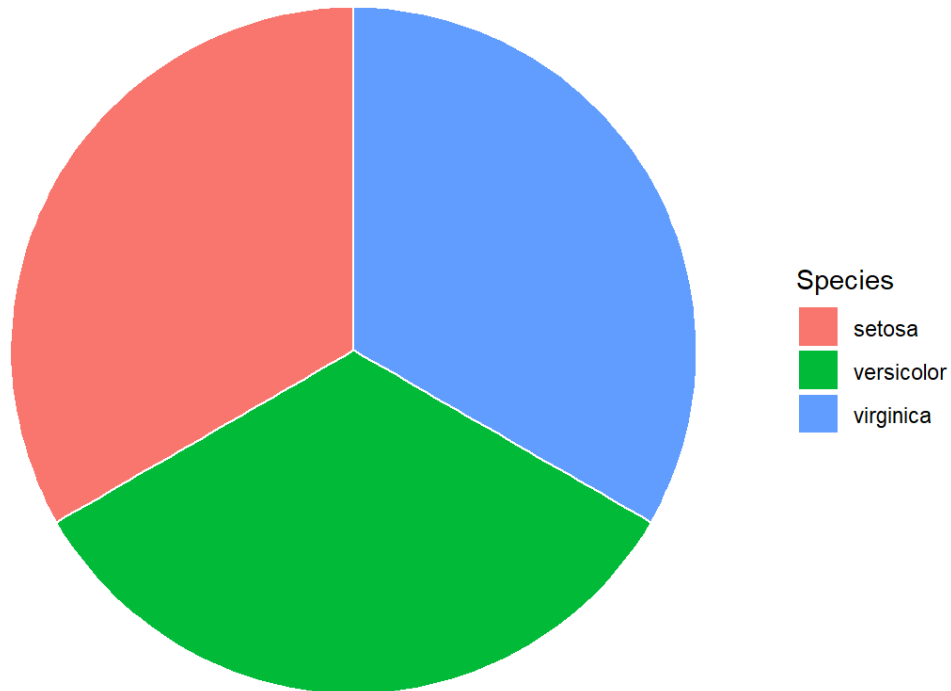
3.4.7 Gráficos de torta

Finalmente, una opción popular para representar datos categóricos es el famoso gráfico de torta o sectores o *pie chart*. Aunque este tipo de gráficos es muy conocido, existe un amplio criticismo (<https://www.bernardmarr.com/default.asp?contentID=1779>) y se aconseja solo cuando las categorías son pocas.

Desde el punto de vista de la *gramática de gráficos*, este gráfico es un gráfico de barras o columnas representado usando coordenadas polares. Veamos como transformar el gráfico de barras representando frecuencias de especies en un gráfico de torta usando `coord_polar()` y algunos extras:

```
ggplot(data = iris) +
  aes(x = "", fill = Species) +
  geom_bar(width = 1, color = "white") +
  coord_polar("y") +
  labs(title = "Frecuencia especies") +
  theme_void()
```

Frecuencia especies

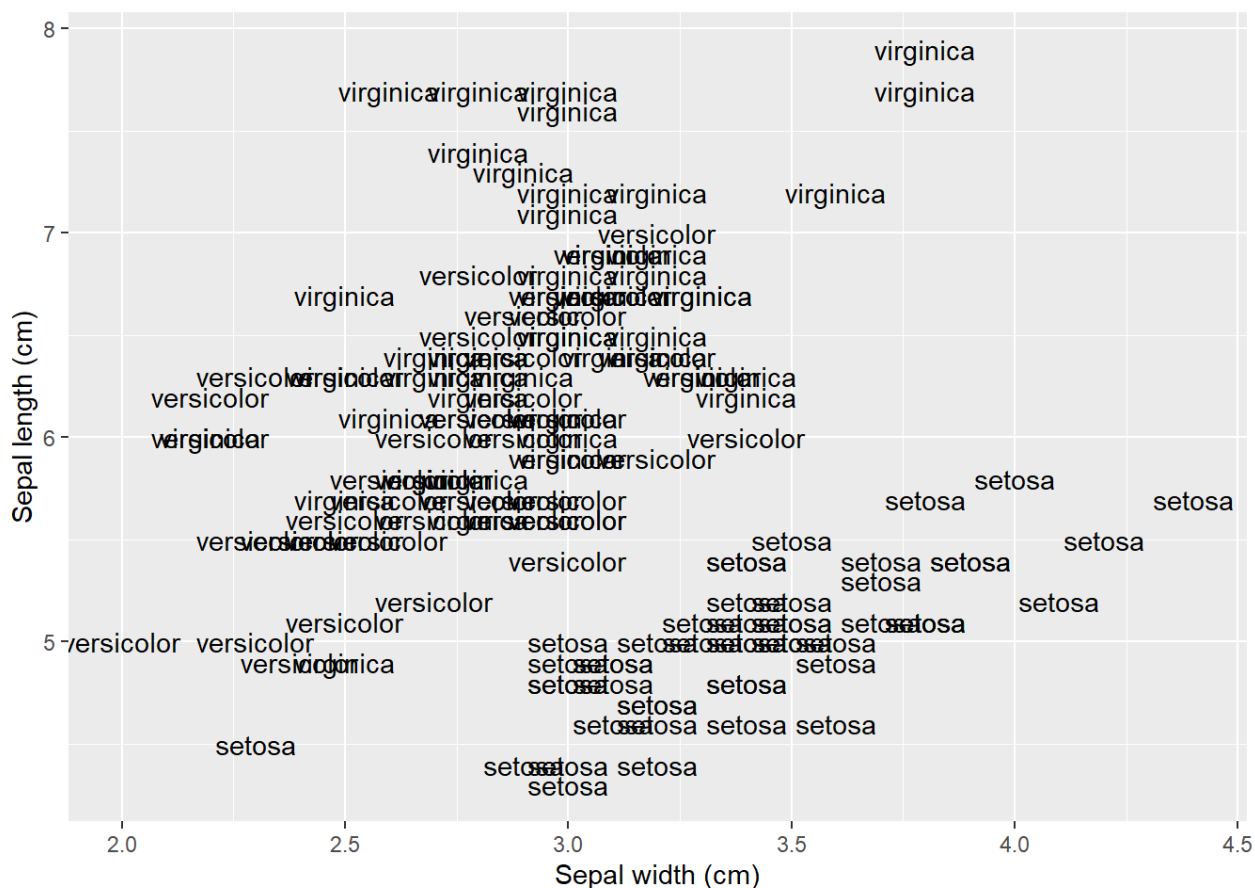


En este caso, `theme_void()` nos permite eliminar algunos componentes del gráfico (fondo, grilla, etiquetas numéricas, ejes) que generalmente no se usan en este tipo de visualización.

3.4.8 Gráficos de texto

Así como se usaron puntos para representar los datos en un gráfico de dispersión y otras variables que definen grupos se incluyeron como colores, rellenos, etc., es posible directamente usar el identificador del caso o el grupo al que pertenece la observación como una geometría usando `geom_text()`.

```
ggplot(data = iris) +  
  aes(x = Sepal.Width, y = Sepal.Length) +  
  geom_text(aes(label = Species)) +  
  labs(x = "Sepal width (cm)", y = "Sepal length (cm)")
```



En casos como éste donde hay una superposición importante de los rótulos, y además los rótulos se repiten mucho (hay solo 3 especies en 150 observaciones), es más eficiente manejar una escala de colores como vimos antes.

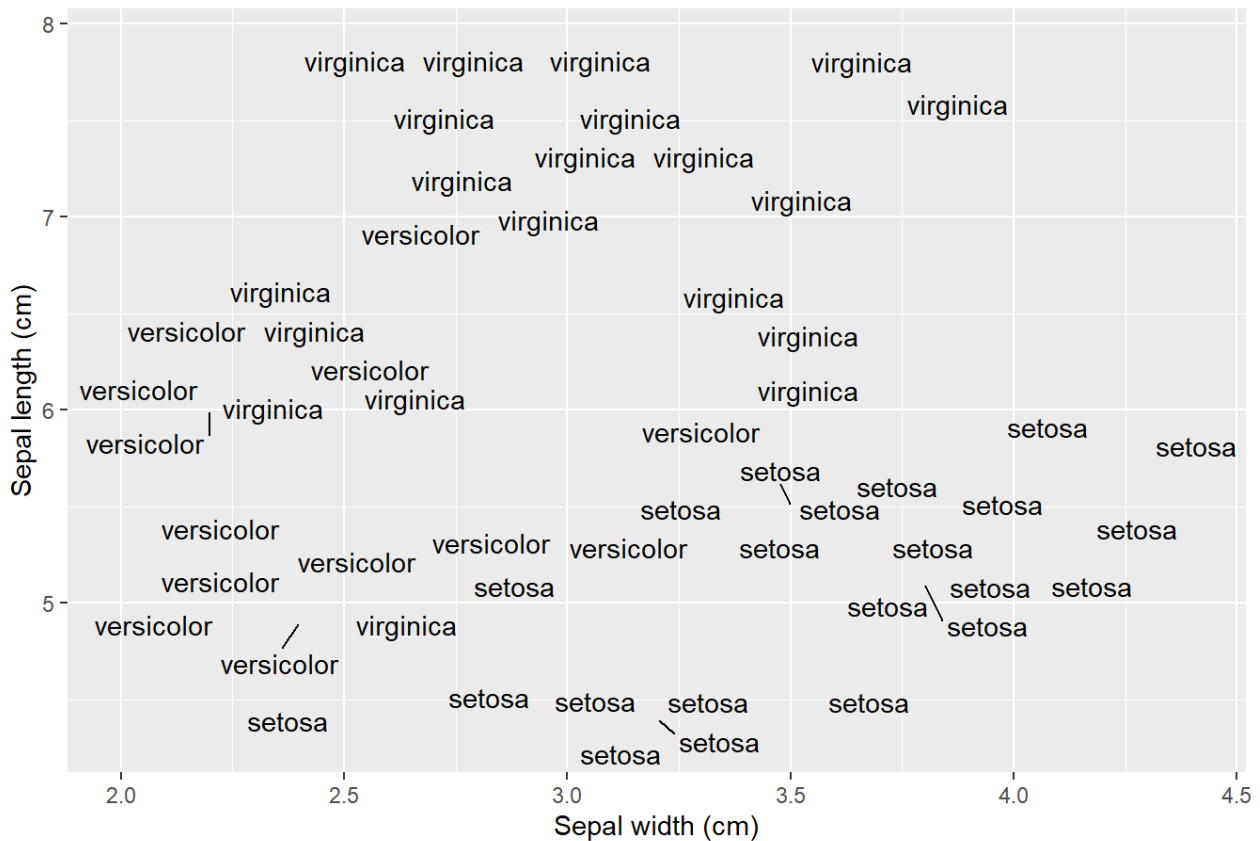
No obstante, si de todos modos quisiéramos mejorar esta distribución de rótulos evitando superposición, podríamos usar el paquete `ggrepel` que tiene funciones para evitar superposición.

```
# Instalar ggrepel si no está instalado
# install.packages('ggrepel')

# Cargar ggrepel
library(ggrepel)

# Gráfico con ggrepel::geom_text_repel()
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length) +
  geom_text_repel(aes(label = Species)) +
  labs(x = "Sepal width (cm)", y = "Sepal length (cm)",
       title = "Gráfico de dispersión")
```

Gráfico de dispersión



Como comentamos antes, en este caso se esta forzando el algoritmo y hay muchos puntos que al superponerse demasiado son omitidos. Esto se controla con `max.overlaps` .

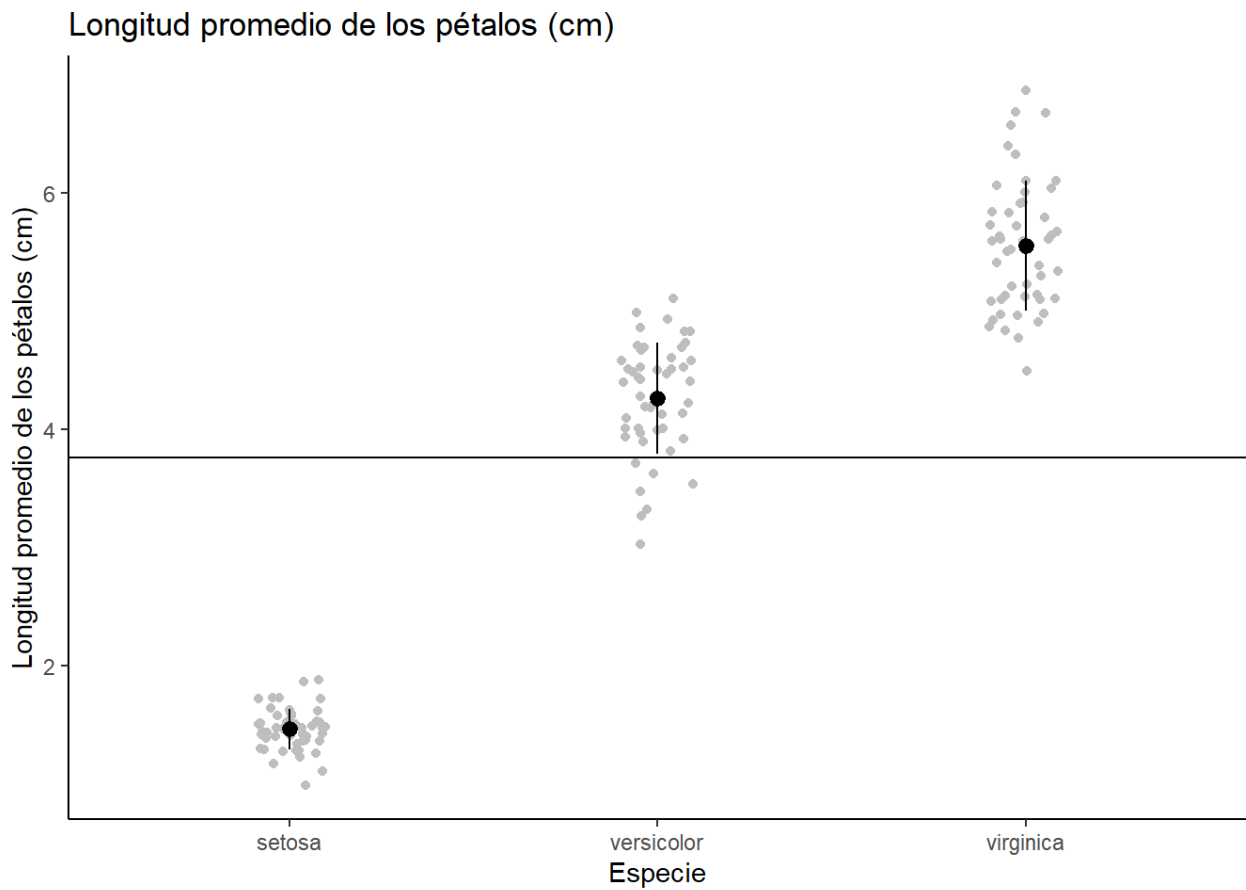
3.4.9 Algunos extras

Las geometrías anteriormente vistas se pueden combinar en distintas formas. Algunas constituyen en si mismas algunas visualizaciones estándar (e.g. histograma, boxplot). Otras al combinarse crean visualizaciones únicas.

A continuación se muestran algunas geometrías extra comúnmente usadas:

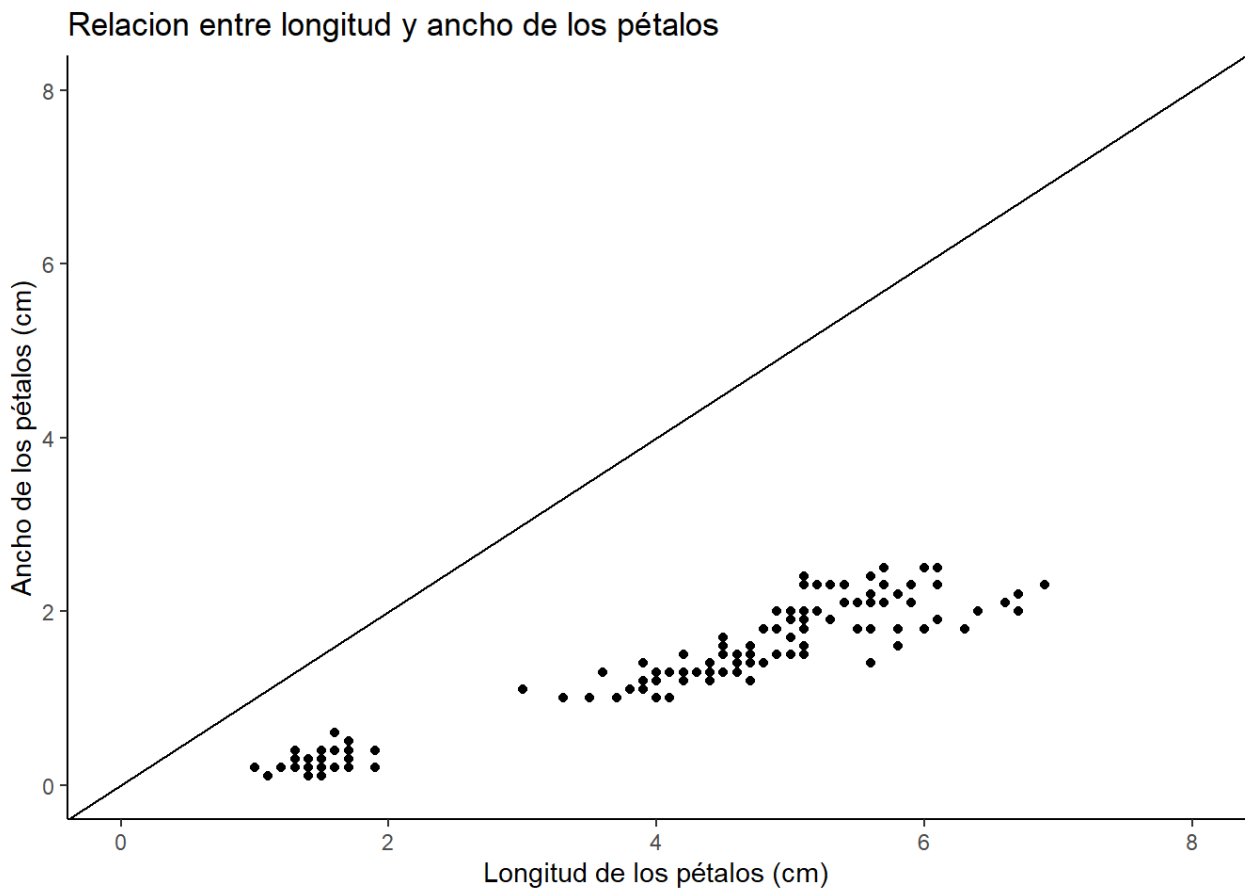
Por ejemplo lineas verticales, horizontales o con una determinada orientación. Tomando el ejemplo de las medias de longitud de pétalos por especie se podría pensar en agregar la media general de la longitud de los pétalos combinando la información de las tres especies. Esto se hace con `geom_hline()` .

```
ggplot() +
  geom_jitter(
    data = iris,
    aes(x = Species, y = Petal.Length),
    color = "gray", width = 0.1
  ) +
  geom_pointrange(
    data = meds_sd,
    aes(x = Species, y = ybar, ymin = ybar - s, ymax = ybar + s)
  ) +
  geom_hline(data = iris, aes(yintercept = mean(Petal.Length))) +
  labs(title = "Longitud promedio de los pétalos (cm)",
    x = "Especie", y = "Longitud promedio de los pétalos (cm)") +
  theme_classic()
```



Se pueden usar varias líneas de referencia, incluso por paneles, etc. Otro ejemplo sería agregar una línea 1:1 con `geom_abline()` a un gráfico de dispersión entre el largo y ancho de los pétalos. En este caso tendría una ordenada al origen 0 y pendiente 1

```
ggplot(iris) +
  aes(x = Petal.Length, y = Petal.Width) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0) +
  labs(title = "Relacion entre longitud y ancho de los pétalos", x = "Longitud de los
pétalos (cm)",
        y = "Ancho de los pétalos (cm)") +
  lims(x = c(0, 8), y = c(0,8)) +
  theme_classic()
```



De acá vemos que la relación es menor a 1.

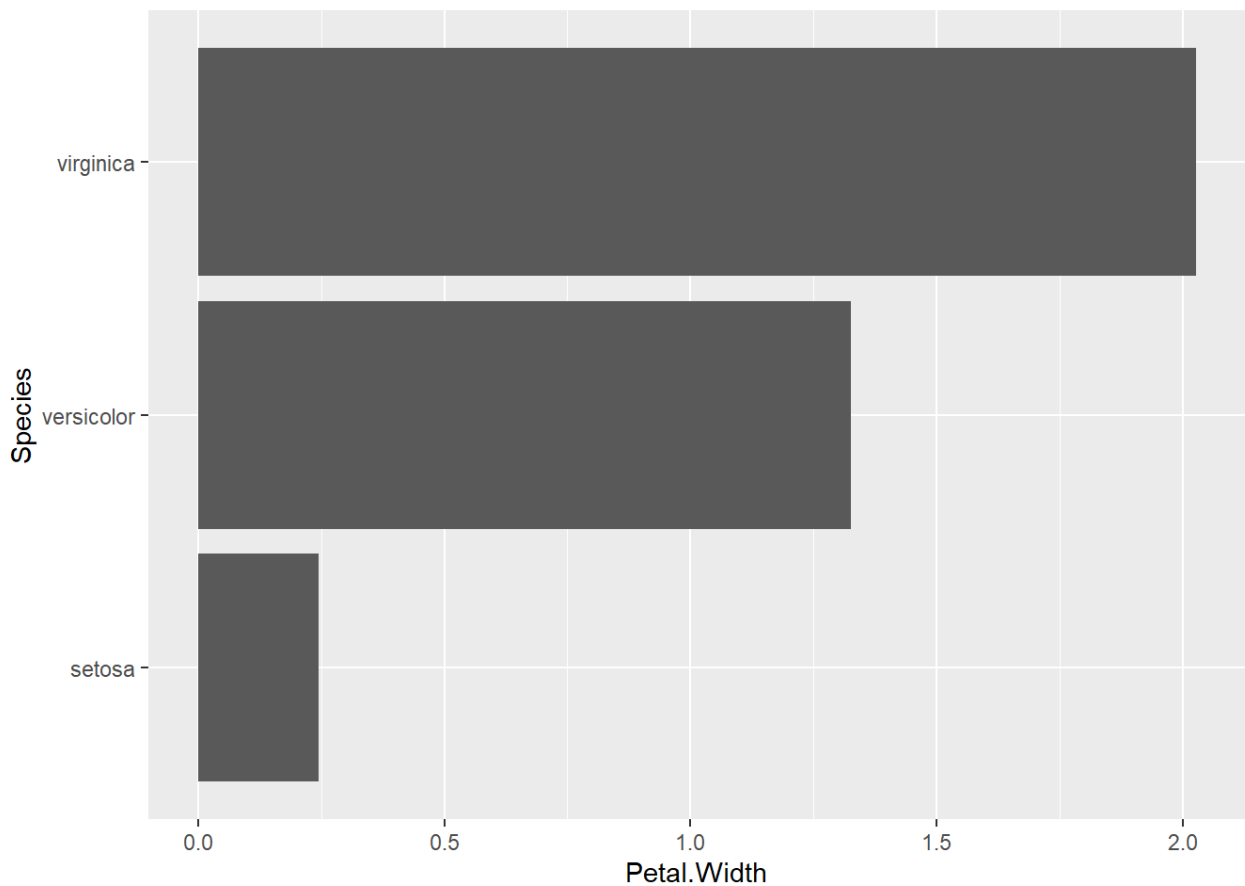
3.5 Sistemas de coordendas

Como vimos antes para el gráfico de torta, podemos usar el componente `coords` para definir cambios en el sistema de coordenadas de un gráfico. Por defecto siempre se asume que las coordenadas son cartesianas `coord_cartesian()` donde hay ejes X e Y.

La otra alternativa es sistema de coordenadas polares (`coord_polar()`) como vimos para gráfico de torta y que también se aplica los gráfico tipo tela de araña o radar.

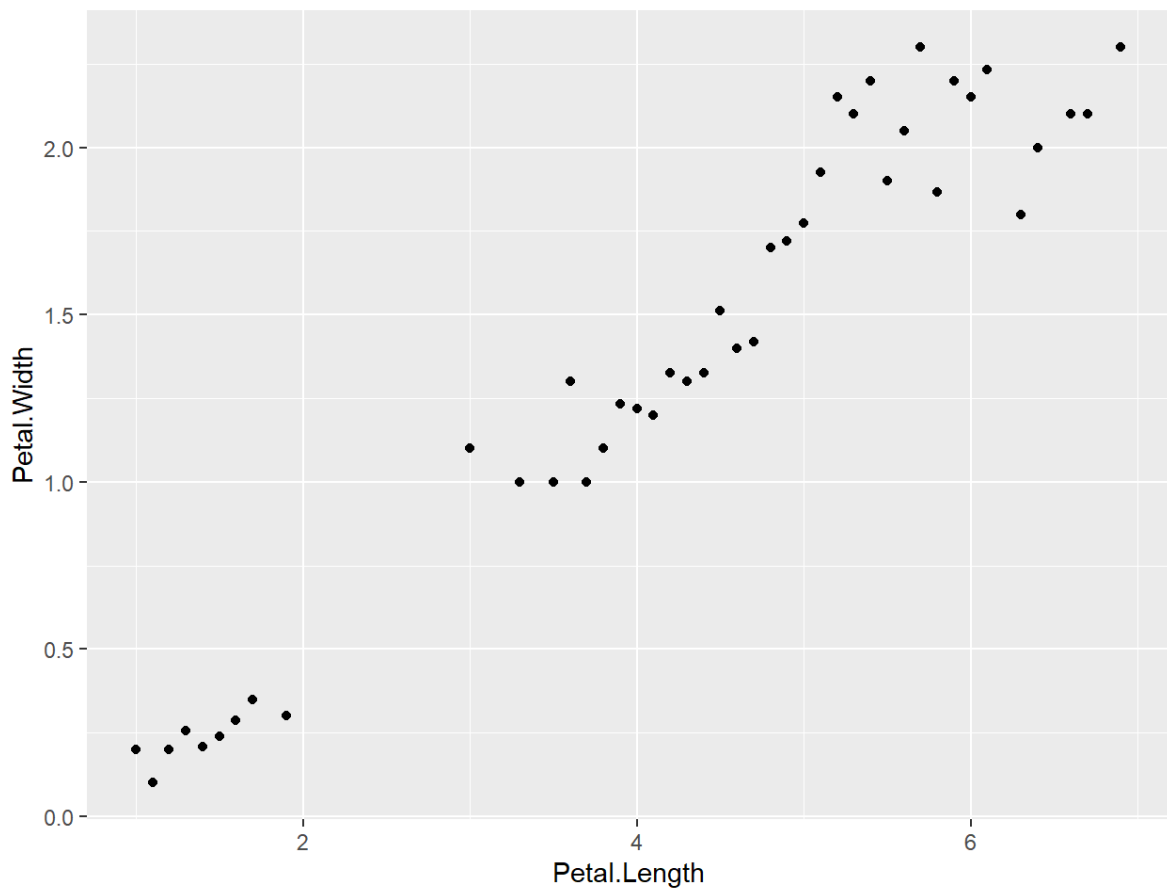
Existen algunos atajos útiles para alterar el sistema de coordenadas como por ejemplo `coord_flip()` que rota el gráfico 90°.

```
ggplot(data = iris) +
  aes(x = Species, y = Petal.Width) +
  geom_bar(stat = "summary", fun = mean) +
  coord_flip()
```

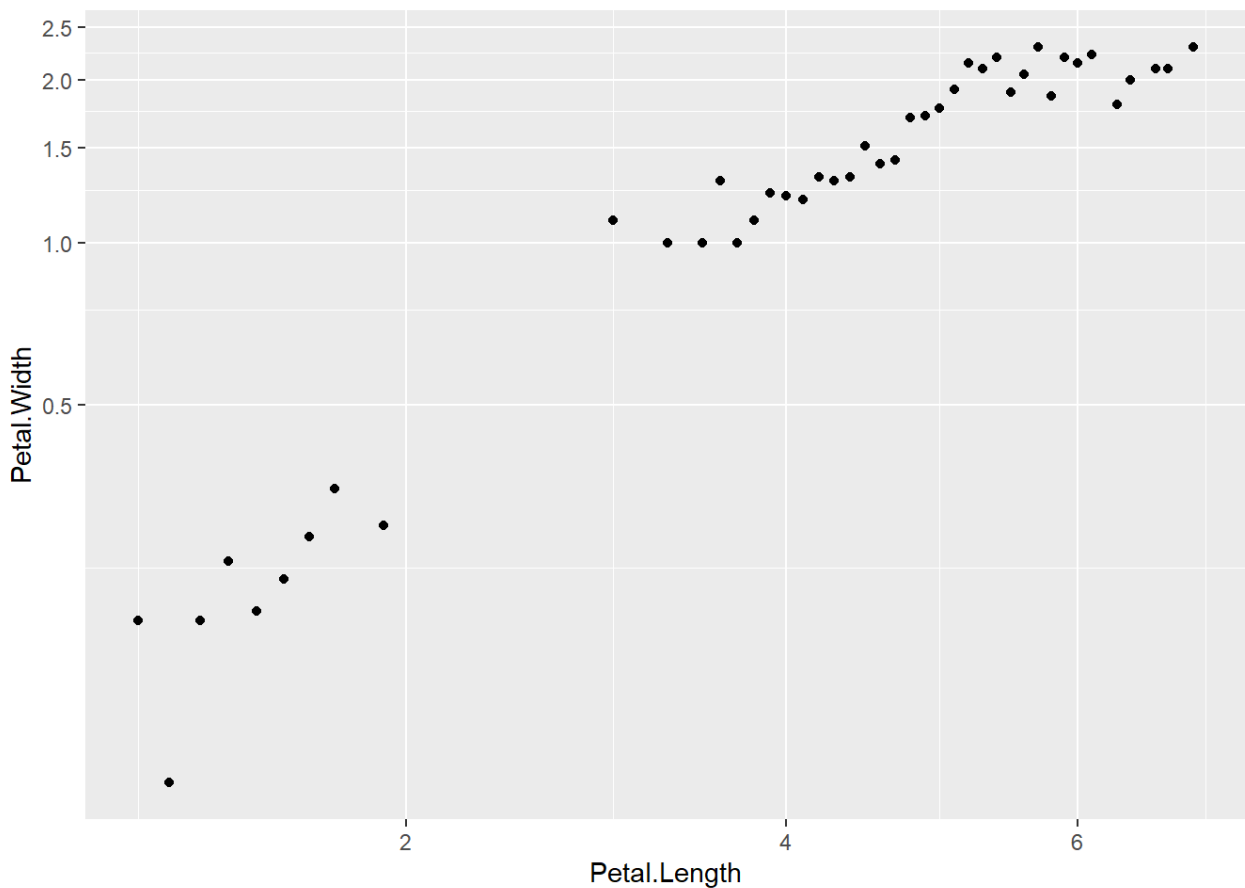
O `coord_fixed()` que permite ajustar una ambos ejes para mantener un determinado aspecto controlado por `ratio` . Por ejemplo, para hacer un gráfico que muestre la relación entre las dimensiones de los pétalos en una escala 2 a 1, es decir, 2 unidades en el eje `y` representan 1 unidad en `x` . Si la relación deseada es 1:1 entonces `coord_equal()` es un atajo útil.

```
ggplot(data = iris) +  
  aes(x = Petal.Length, y = Petal.Width) +  
  geom_point(stat = "summary", fun = mean) +  
  coord_fixed(ratio = 2)
```



También se pueden aplicar transformaciones específicas a cada eje con `coord_trans()`. Veamos en el caso anterior como podríamos expresar `Petal.Width` en escala logarítmica y `Petal.Length` en escala de raíz cuadrada.

```
ggplot(data = iris) +  
  aes(x = Petal.Length, y = Petal.Width) +  
  geom_point(stat = "summary", fun = mean) +  
  coord_trans(y = "log10", x = "sqrt")
```



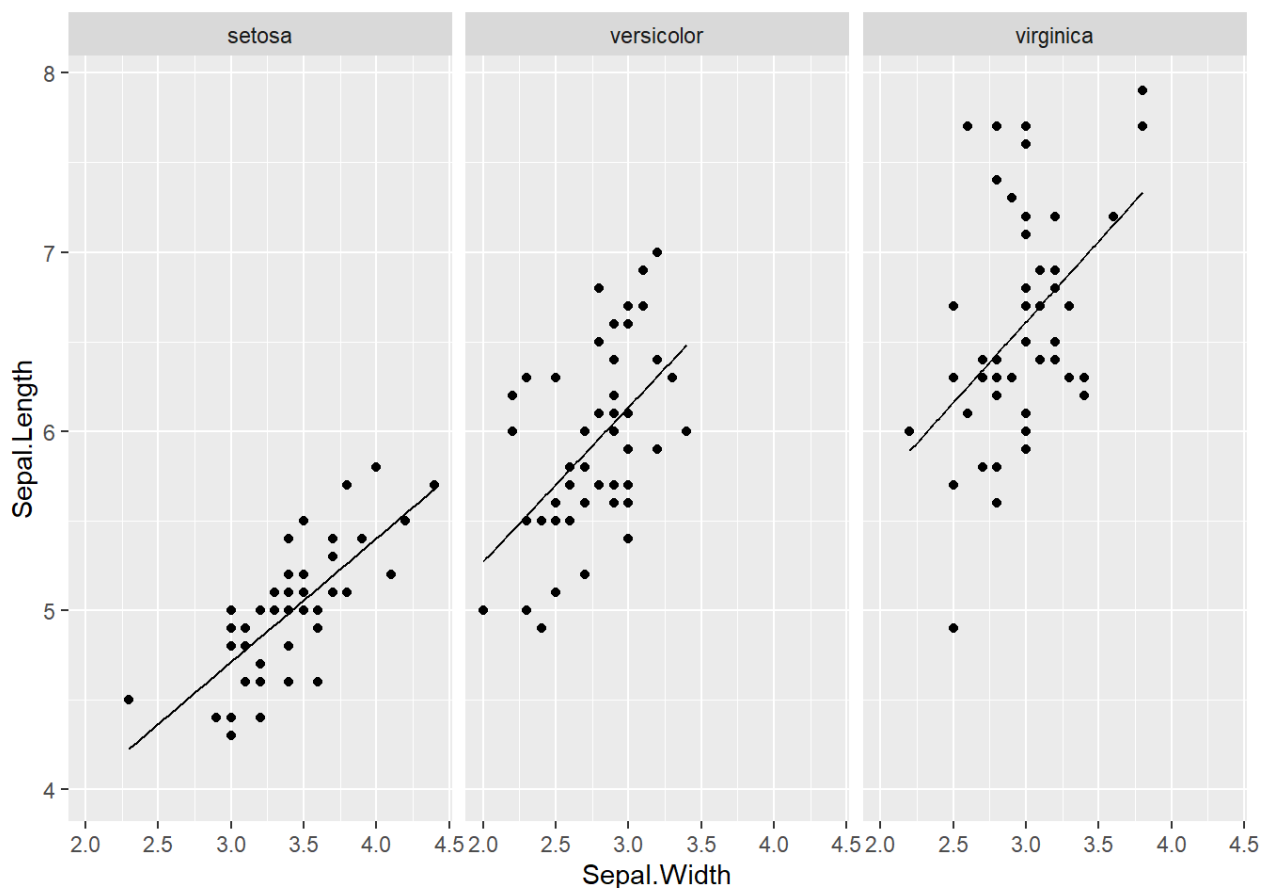
En este caso no tiene mucho sentido pero para variables con distribuciones muy asimétricas, es pueden mostrar una relación fácil para modelar linealmente.

3.6 Gráficos condicionales o por paneles: facets

Todo lo anteriormente visto puede se aplicado tanto para un gráfico único o bien dividido en subgraficos o paneles. Esto último se hace facilmente usando una o más variables condicionales con `facets`. Este tipo de gráficos también se denomina gráficos condicionales ya que muestran la relación de al menos dos variables de interés a través de los niveles de una tercera variable: $y \sim x \mid z$.

Por ejemplo, en el primer gráfico vimos como mostrar la relación de las dimensiones de los sépalos usando la especie como color. Eventualmente, ese gráfico podría dividirse en 3 paneles (uno por especie) y mostrar en cada uno el subconjunto de puntos.

```
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  facet_wrap(~ Species)
```

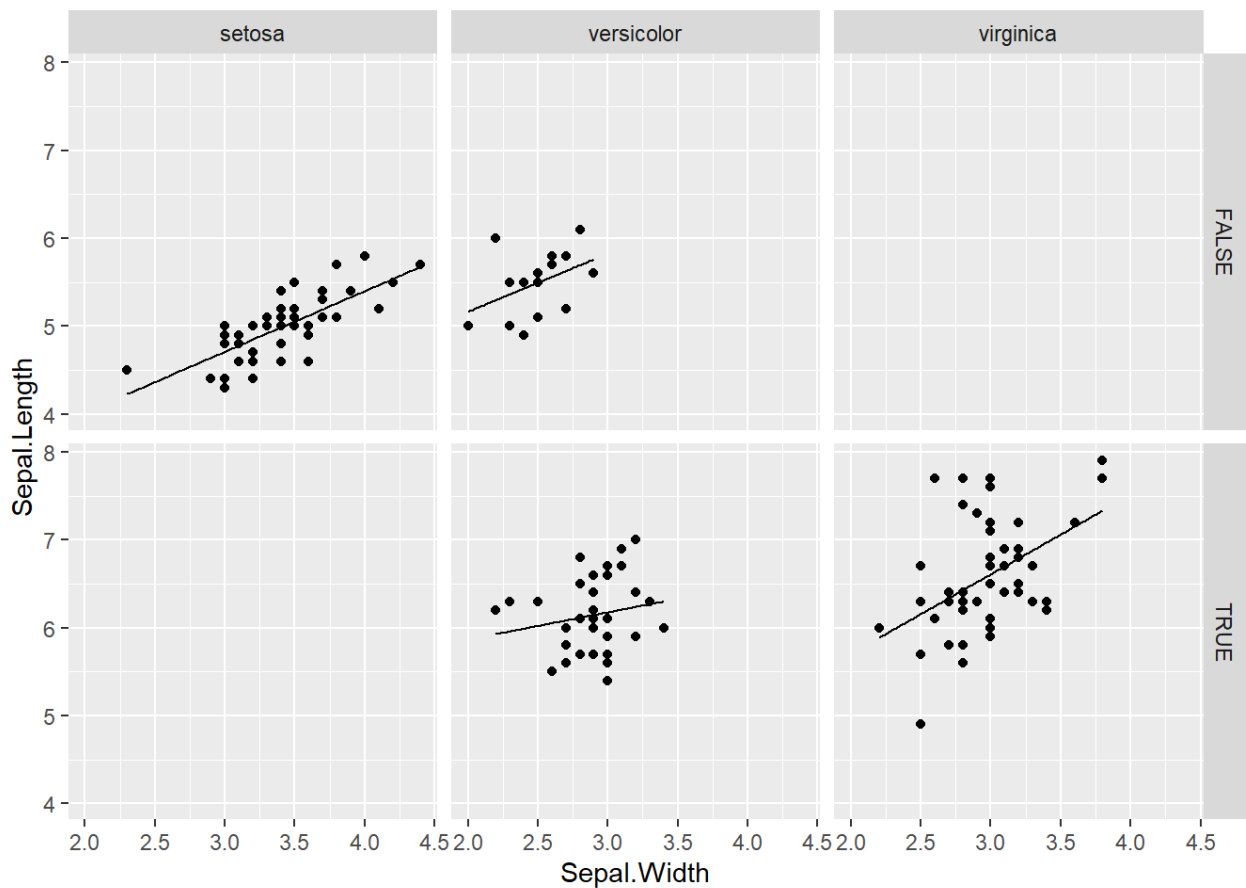


Hay dos tipos de facetado: `facet_wrap()` y `facet_grid()`. El primero permite agregar una o mas variables condicionales pero cada subpanel se muestra secuencialmente. Funciona bien cuando tenemos una sola variable para dividir los subplots o pocos niveles en la combinación. La forma de indicar la variable es `~ variable`.

En cambio, `facet_grid()` permite organizar los subplots en filas y columnas. Las variables se indican en este orden `fila ~ columna`. Para este ejemplo vamos a crear una variable que divide las observaciones segun un largo de pétalo arbitrario y luego usar la especie y esa nueva variable para mostrar la relación de las dimensiones de los sépalos:

```
# Creacion de una segunda variable para ilustrar el ejemplo
iris$Petal4 <- iris$Petal.Length > 4
```

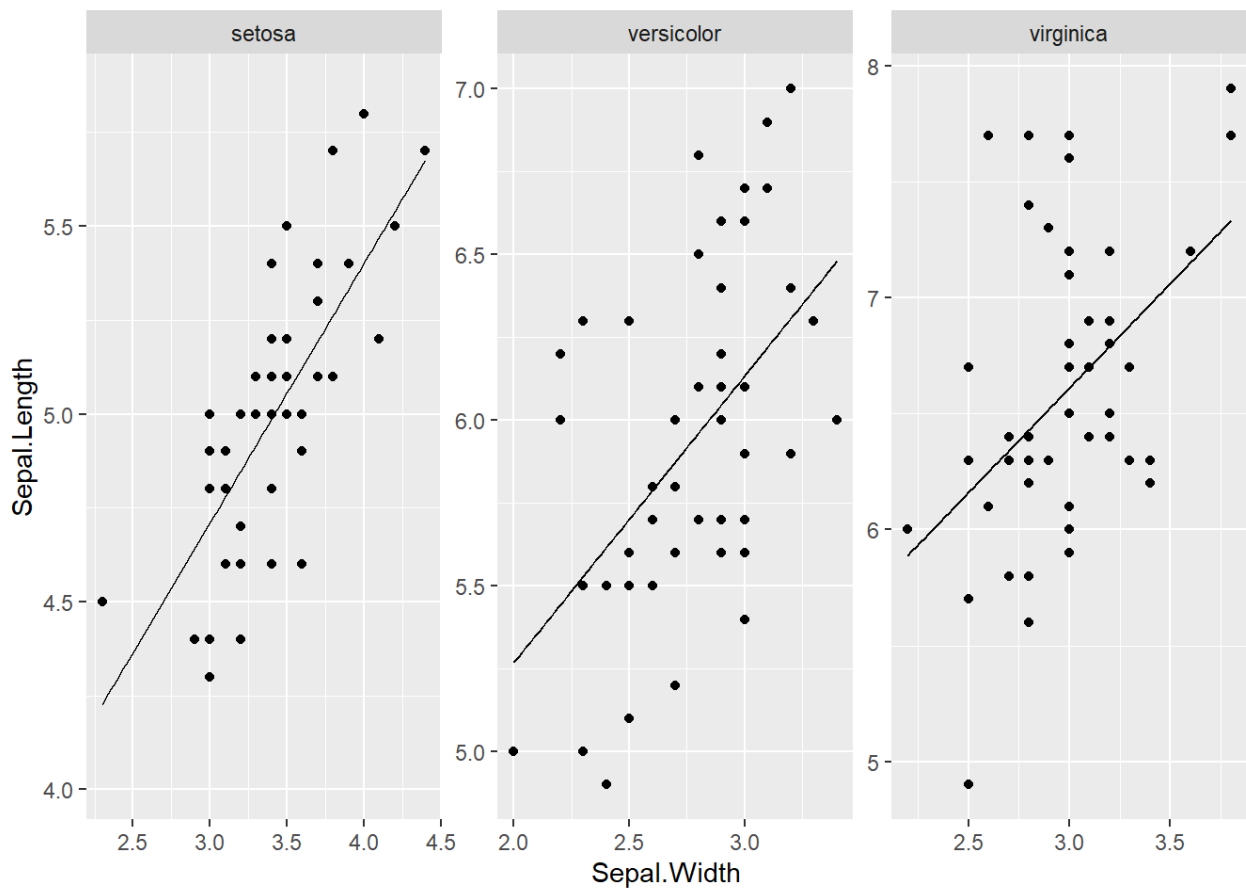
```
# Gráfico usando facet_grid
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  facet_grid(Petal4 ~ Species)
```



Aquí vemos como las especies quedaron en columnas y se generaron dos filas de acuerdo a si las observaciones tenían una longitud de pétalo mayor a 4 cm.

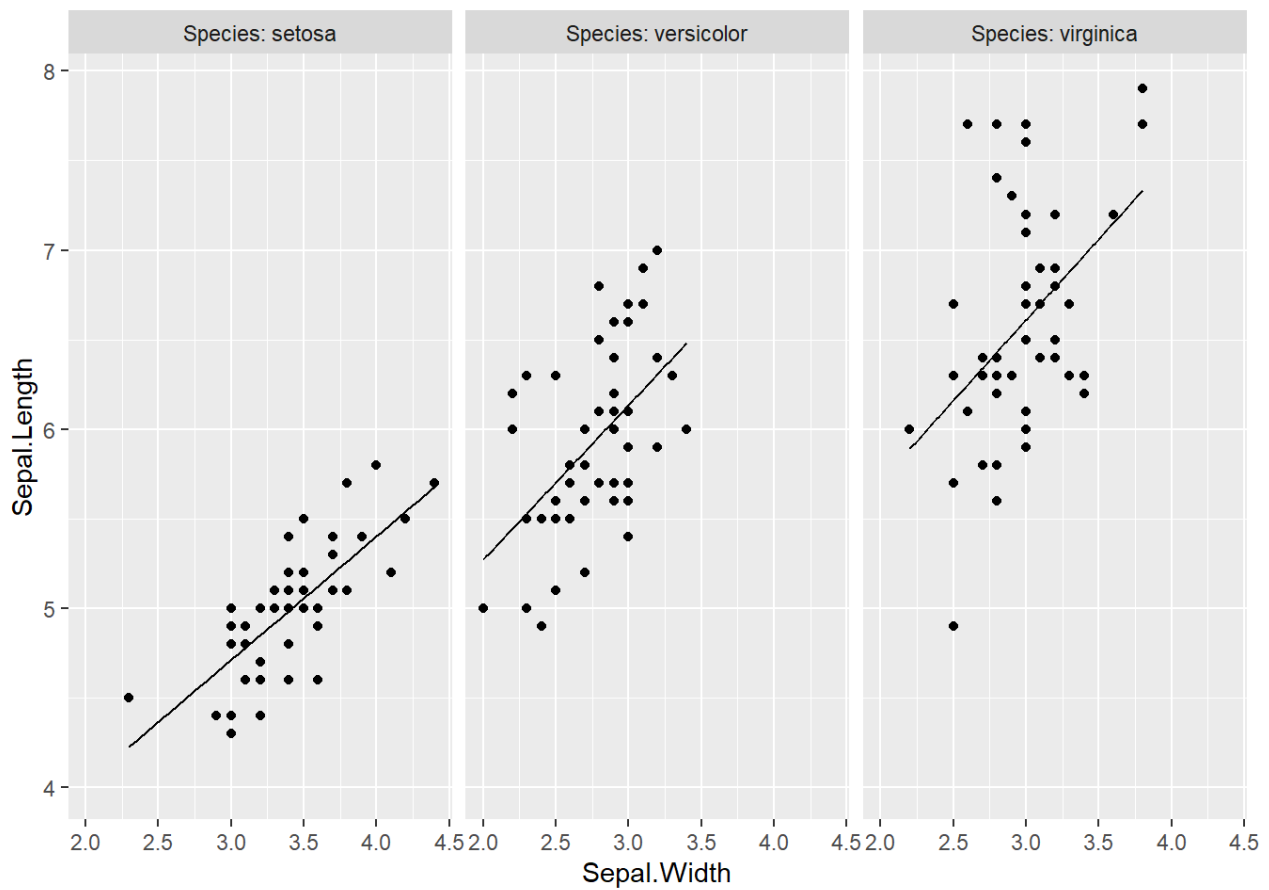
Por defecto los subplots o `facets` tienen escalas iguales en ambos ejes. A veces conviene dejar una o las dos escalas variar libremente, esto se hace con el argumento `scales` y las palabras clave `'free_y'`, `'free_x'` o `'free'` (ambas a la vez).

```
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  facet_wrap(~ Species, scales = "free")
```



Otra aspecto importante en la visualización usando facets es el texto que identifica cada panales. Esto depende de como están configurados los datos y se controla con el argumento `labeller`. Por defecto se toma el valor del factor que se usa para definir los grupos. En algunos casos conviene incluir el nombres de la variable.

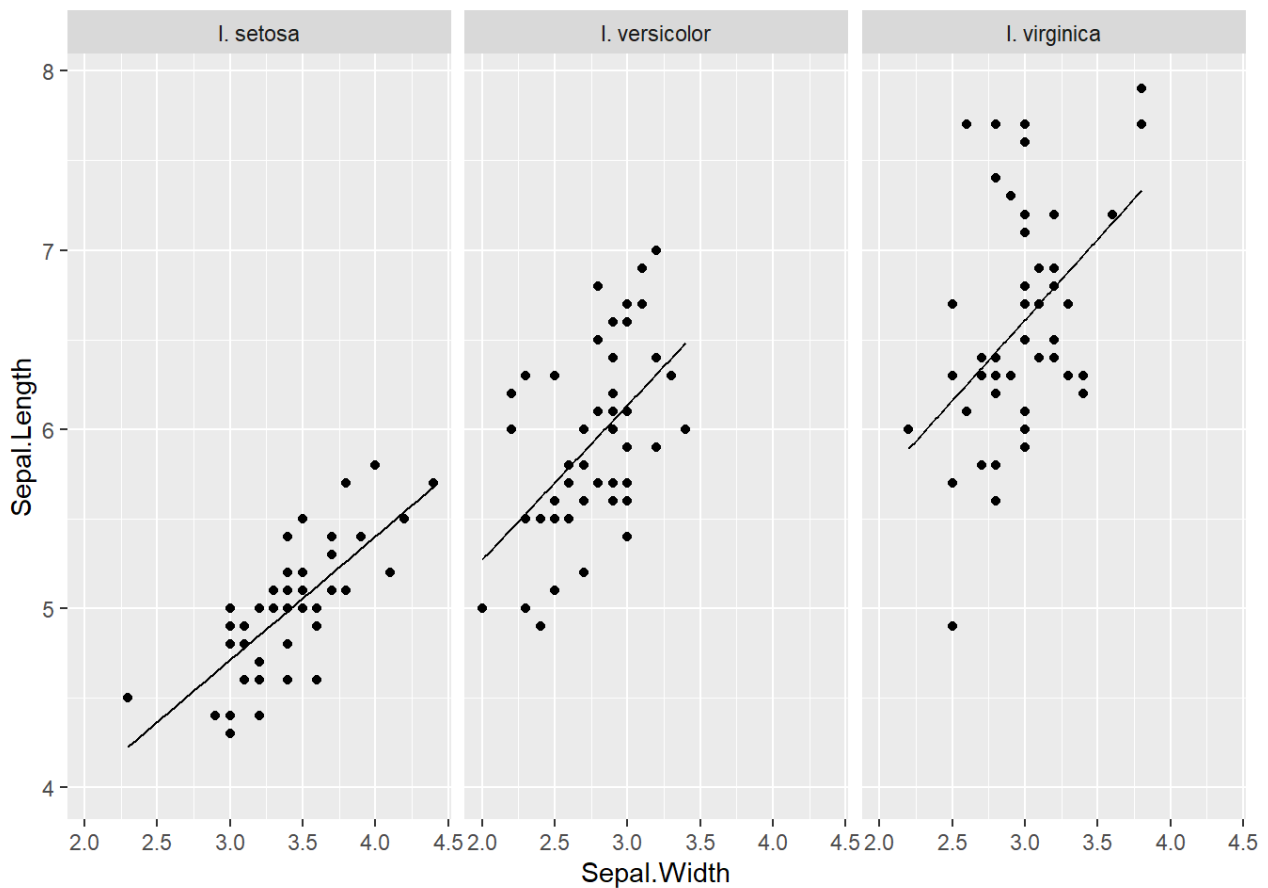
```
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  facet_wrap(~ Species, labeller = label_both)
```



O bien construir nuestros nombres ad-hoc:

```
# Vector con los nombres que queremos mostrar
lbls <- c(
  setosa = "I. setosa",
  virginica = "I. virginica",
  versicolor = "I. versicolor"
)

# Grafico con custom labels
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  facet_wrap(~ Species, labeller = labeller(Species = lbls))
```



3.7 Temas

Los temas en `ggplot` hacen referencia a la forma de controlar posición, el aspecto, y las formas de los distintos componentes del gráfico. El listado de componentes que se pueden modificar en un tema se incluyen en `?theme()`. Como vemos la lista es larga ya que cada aspecto del gráfico puede controlarse permitiendo crear nuestros propios temas.

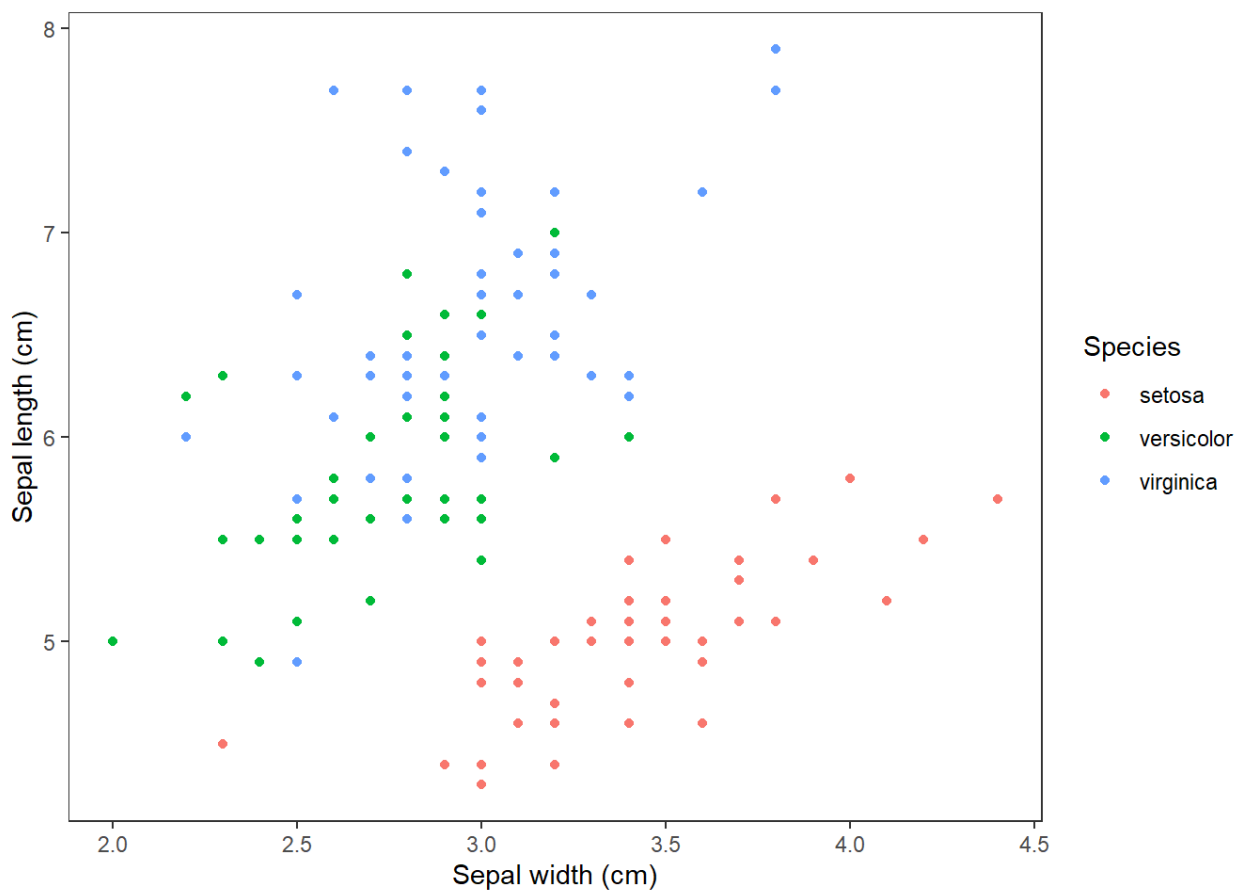
Por defecto los gráficos utilizan un tema llamado `theme_gray()` que tiene una selección de parámetros elegante y que sirve para la mayoría de los casos. No obstante, existen otros temas específicos que pueden ser de interés.

En algunos ejemplos anteriores vimos como agregar el tema blanco y negro `theme_bw()` el cual básicamente remueve el fondo gris. También vimos `theme_classic()` que introduce más modificaciones. El listado de temas se puede ver en la página de ayuda de cualquier tema, e.g. `?theme_gray`

Los temas pueden considerarse como un punto de partida sobre el cual podemos hacer modificaciones extra para definir nuestro propio tema. Por ejemplo, el tema `them_bw()` remueve el fondo gris pero si queremos quitar la grilla podemos hacer:

```
# Modificar el tema
mi_tema <- theme_bw() + theme(panel.grid = element_blank())

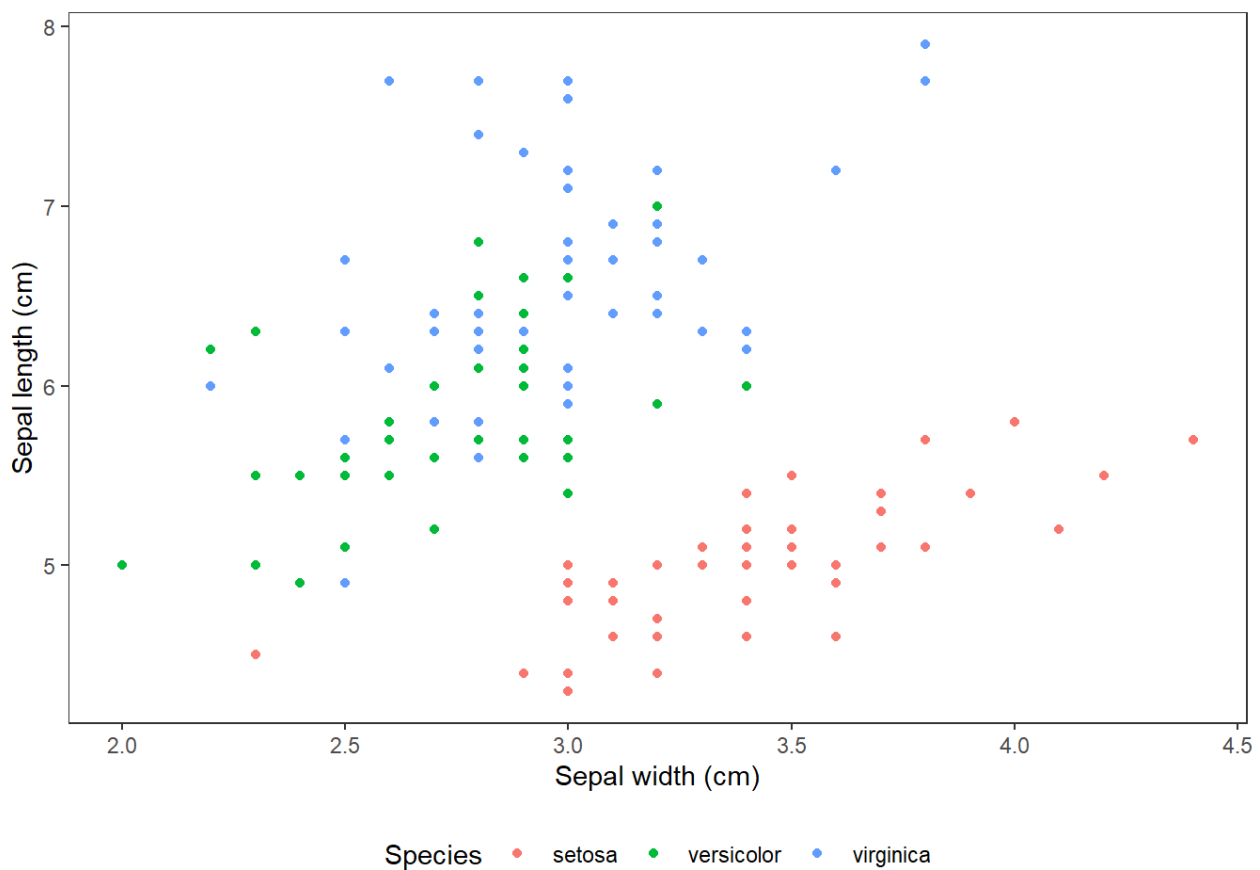
# Aplicar nuestro nuevo tema.
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length, color = Species) +
  geom_point() +
  labs(x = "Sepal width (cm)", y = "Sepal length (cm)",
       color = "Species") +
  guides(position = "bottom") +
  mi_tema
```

Otro aspecto muy útil es controlar la posición de la leyenda. Esto se puede hacer con los argumentos `legend.*` dentro del tema.

```
# Modificar el tema
mi_tema <- theme_bw() + theme(panel.grid = element_blank(),
                             legend.position = "bottom")

# Aplicar nuestro nuevo tema.
ggplot(data = iris) +
  aes(x = Sepal.Width, y = Sepal.Length, color = Species) +
  geom_point() +
  labs(x = "Sepal width (cm)", y = "Sepal length (cm)",
       color = "Species") +
  guides(position = "bottom") +
  mi_tema
```



3.8 Librería `plotly`: gráficos interactivos y más llamativos

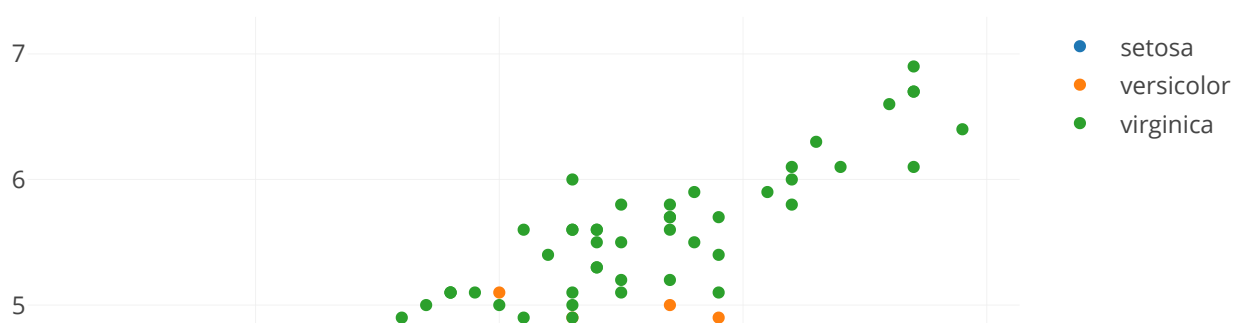
Como siempre ocurre con R, aparecen cada vez más librerías o paquetes que mejoran algunas características de los anteriores, o bien, proponen nuevas. En este caso vamos a presentar un ejemplo usando el paquete `plotly` (<https://plotly.com/r/reference>). Para instalarlo:

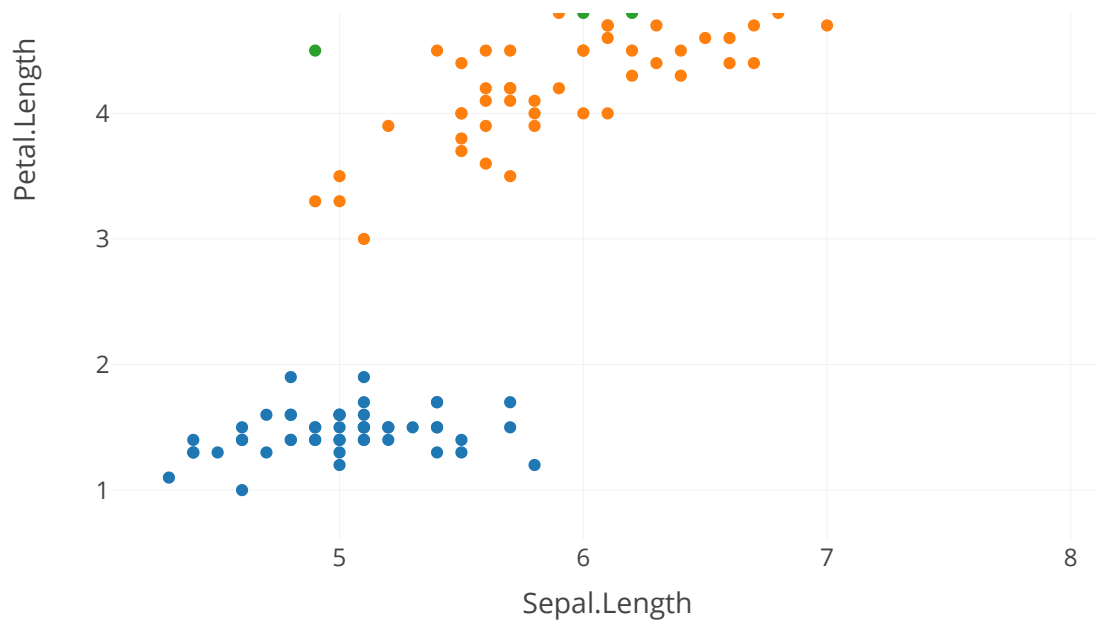
```
# Instalación del paquete plotly
install.packages("plotly")
```

Todos los gráficos hechos con el paquete `plotly` tienen la capacidad de ser interactivos dentro del panel de gráficos de **R** o **RStudio** o bien si se incluyen dentro de un documento HTML. Si se copia el gráfico y se pega en un documento de Word, o un PDF, perderá la capacidad de ser interactivo.

Por interactivo nos referimos a la capacidad de, mediante el mouse, obtener información del gráfico o bien interactuar haciendo zoom o paneo para explorar partes específicas. Veamos un ejemplo con un gráfico de puntos.

```
library(plotly)
plot_ly(data = iris, x = ~Sepal.Length, y = ~Petal.Length,
        type = "scatter", name = ~Species)
```

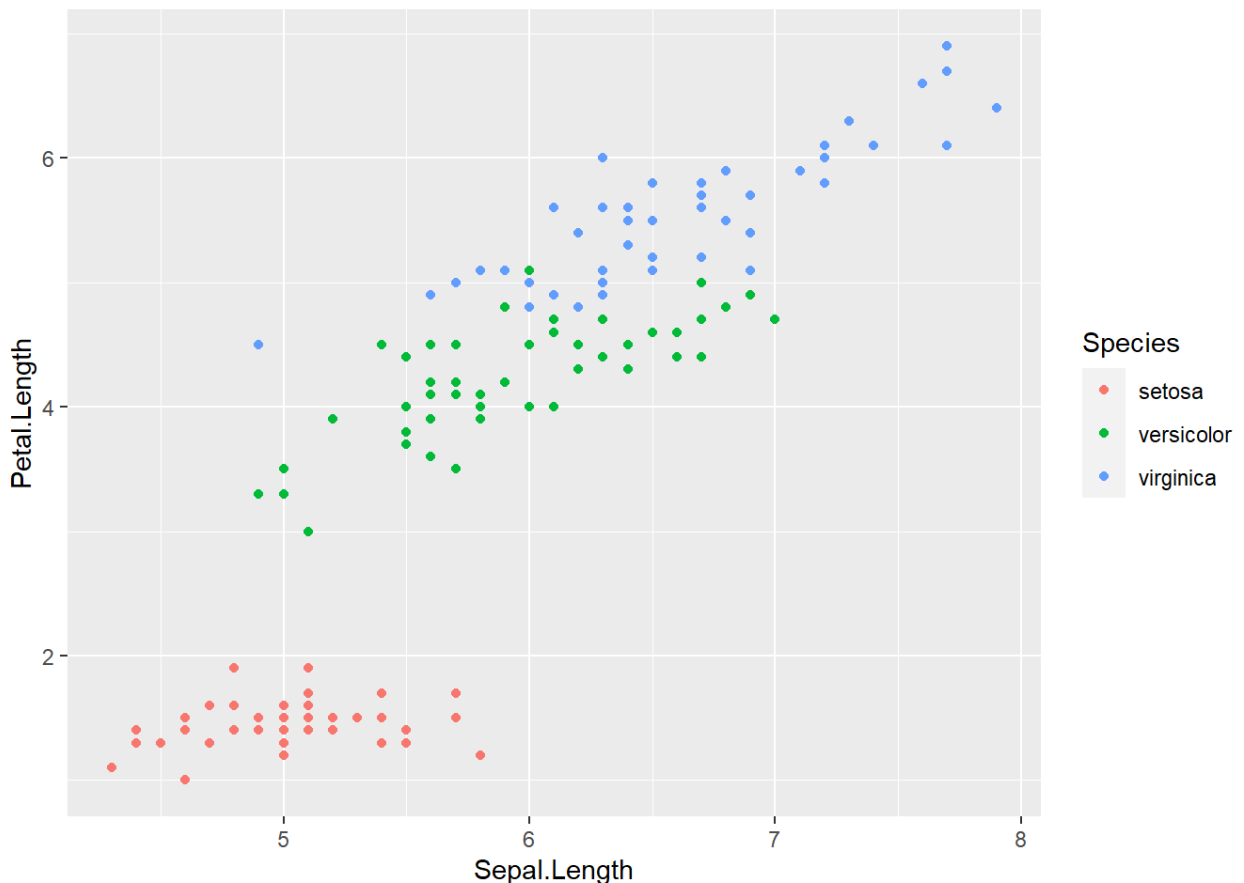




Las sintaxis es un poco diferente a lo que veníamos viendo para `ggplot2` pero conserva alguna lógica. Como vemos necesitamos definir el dataframe con los datos, que variable se mapean en cada eje y en `type` el tipo de gráfico o geometría que queremos utilizar y con `name` los subgrupos.

Una opción para convertir nuestro gráfico hecho en `ggplot2` en una visualización interactiva es usando la función de `ggplotly()`. Funciona bien para visualizaciones simples y con pocos layers. Hagamos el mismo gráfico anterior primero en `ggplot2` y luego convertimos en `plotly`:

```
# gráfico ggplot
plt <- ggplot(data = iris) +
  aes(x = Sepal.Length, y = Petal.Length, color = Species) +
  geom_point()
plt
```



Conversión a plotly

```
ggplotly(plt)
```

