# Introduction to R and the tidyverse

– advanced plotting –

Paolo Crosetto

# Tweaking `ggplots`

# Quick recap of ggplot

> *A general ggplot template*

```
ggplot(data = ...) +
aes(x = ..., y = ..., ...)) +
```

`geom_*()` `+` add geometrical objects (points, lines, ...)

`facet_*()` `+` split into subplots

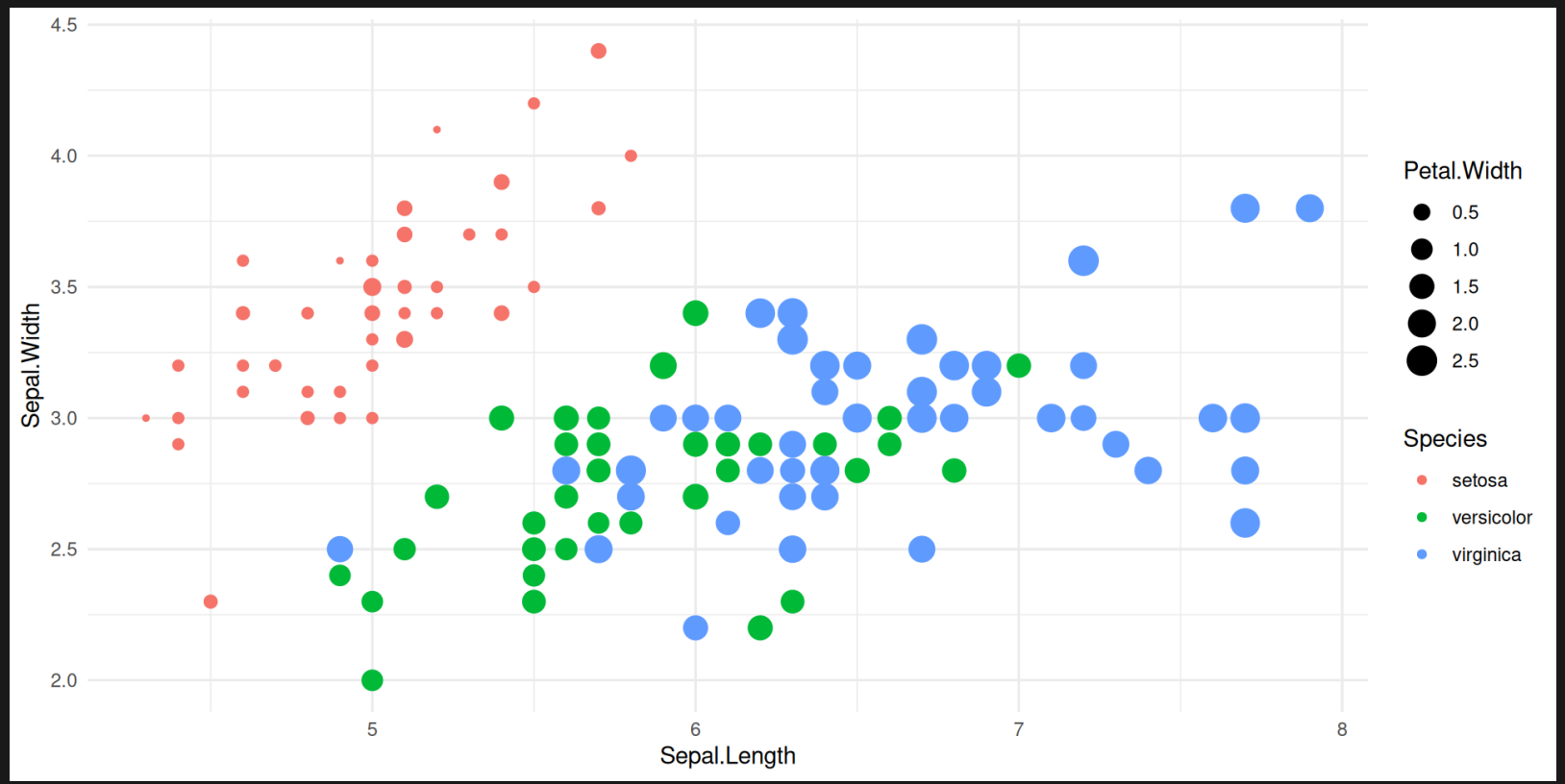
`coord_*()` `+` changes the coordinate system

`scale_*_*()` `+` changes the way `geoms` look

`theme_*()` changes the look & feel of the plot

# A base plot to work with

```
1 base <- iris %>%
2   ggplot(aes(x = Sepal.Length, y = Sepal.Width,
3              color = Species, size = Petal.Width))+
4   geom_point()
5 base
```



# 1 · Coordinates

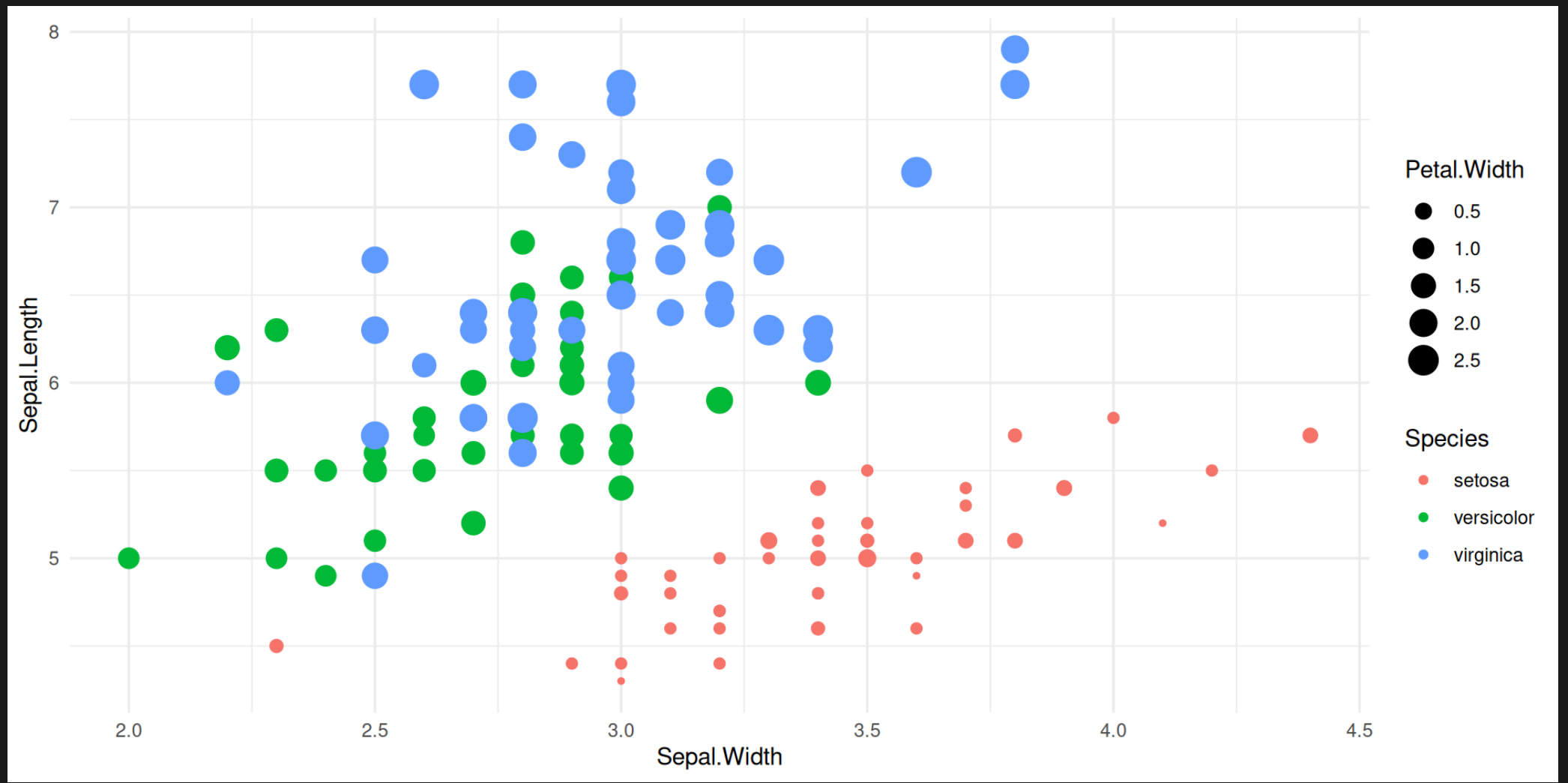
# Coord_functions

> `coord_*` functions change the behaviour of the **axes**

- invert coordinates
- boundaries
- scale (linear, log)
- type (cartesian, polar)

# Invert axes

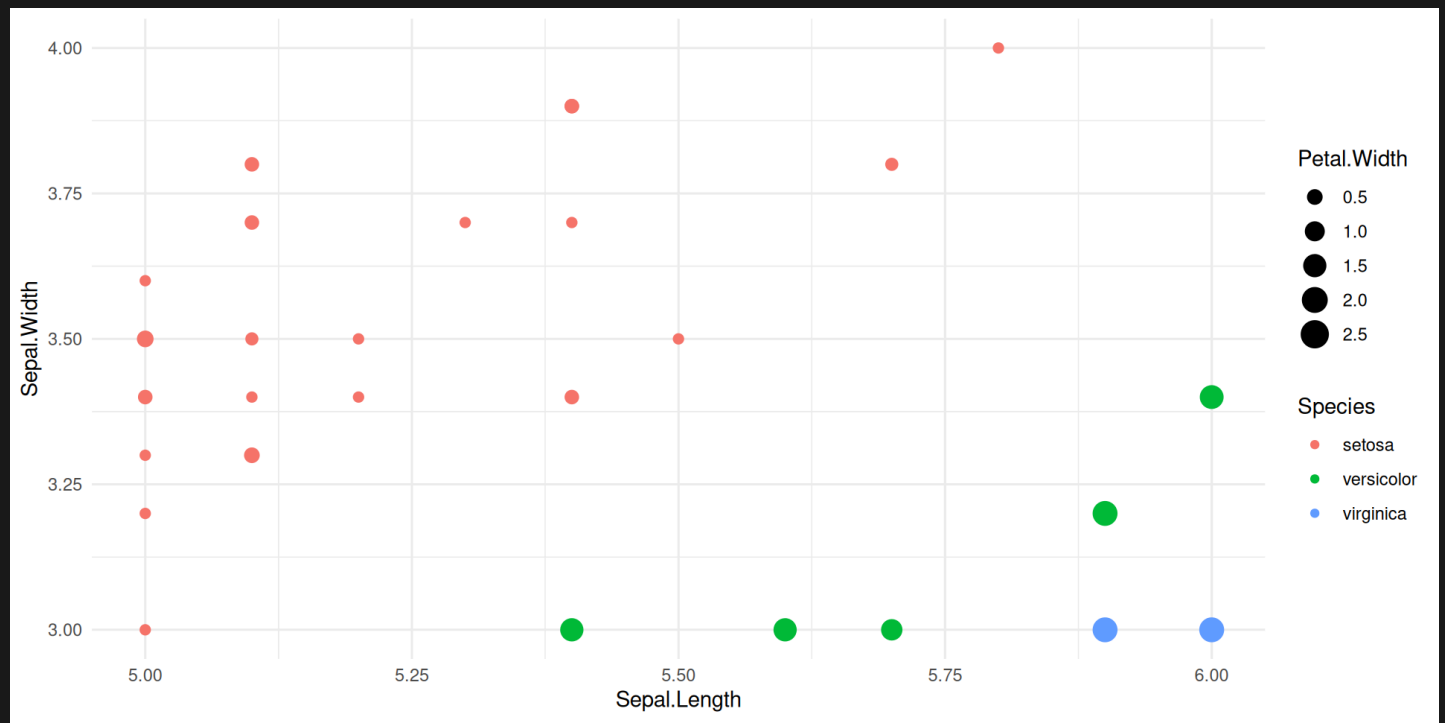
```
1  base + coord_flip()
```



# Cartesian linear coordinates: limits

> change the limits of the plot to ‘zoom’ on a part of it (!!handle with care!!)

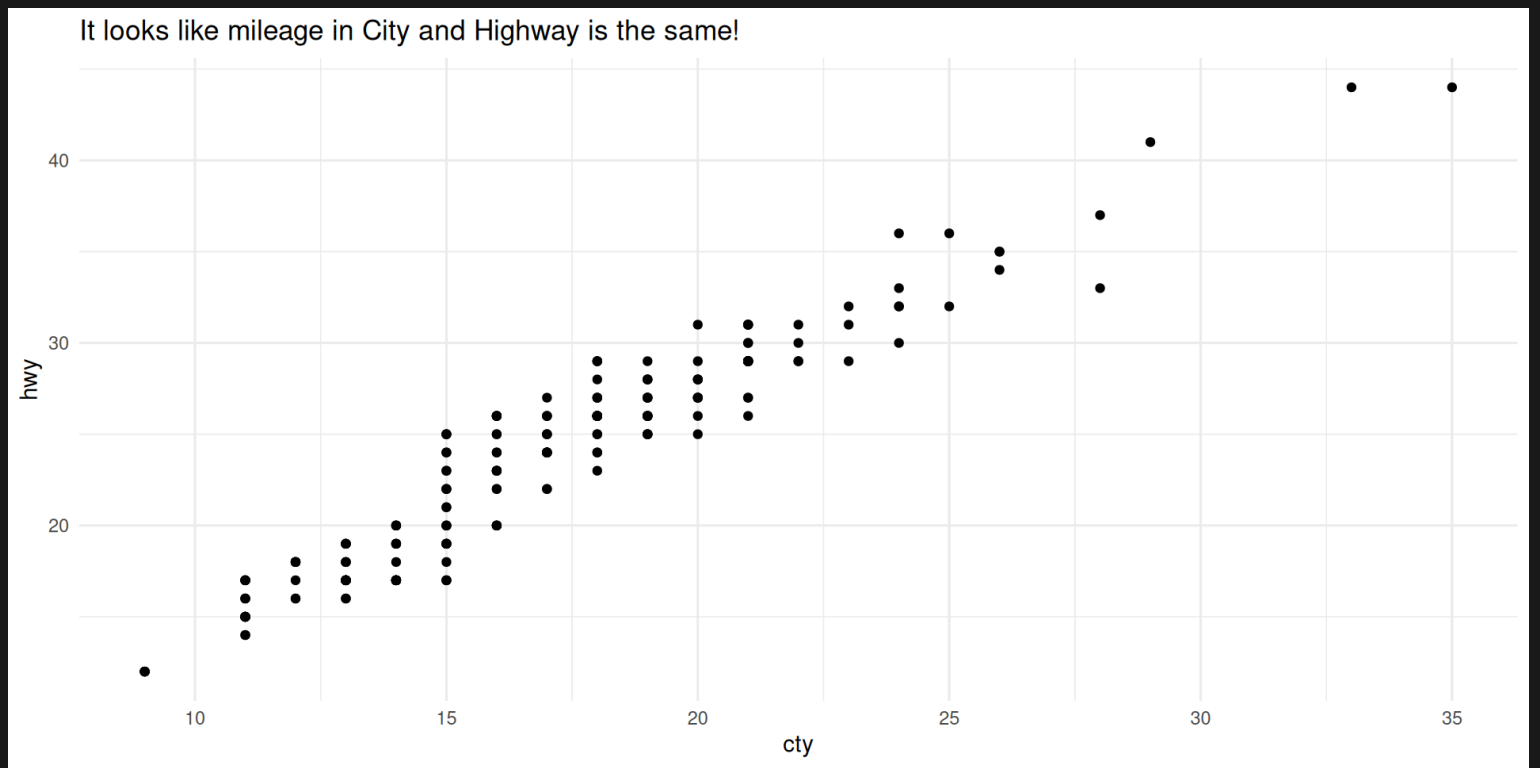
```
1  base + coord_cartesian(xlim = c(5, 6), ylim = c(3,4))
```



# Cartesian linear coordinates: fixed

> if you need the scales on the two axes to be comparable

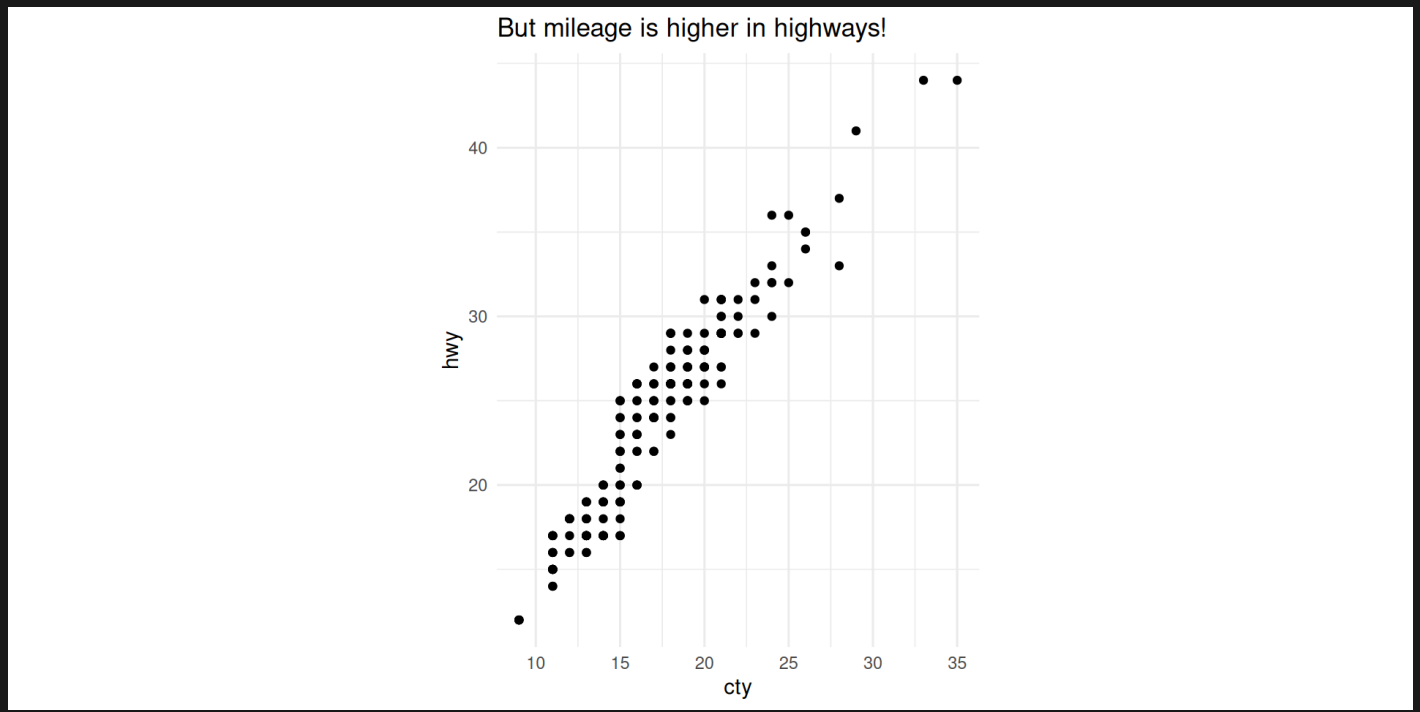
```
1 ggplot(mpg, aes(cty, hwy)) + geom_point() +
2   labs(title = "It looks like mileage in City and Highway is the same!")
```



# Cartesian linear coordinates: fixed

> if you need the scales on the two axes to be comparable

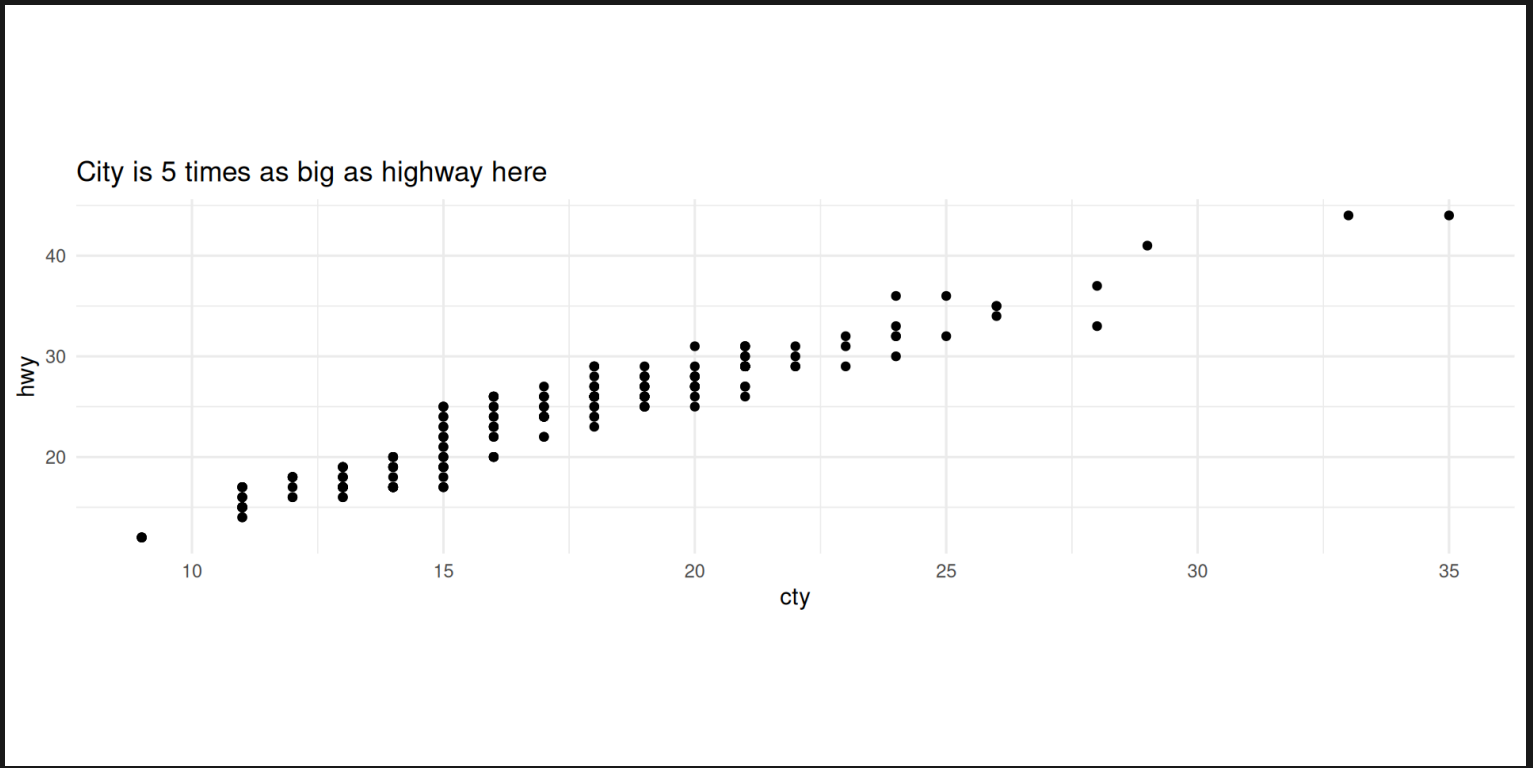
```
1 ggplot(mpg, aes(cty, hwy)) + geom_point() +
2   coord_fixed() +
3   labs(title = "But mileage is higher in highways!")
```



# Cartesian linear coordinates: fixed

> if you need axes to be in an exact proportion

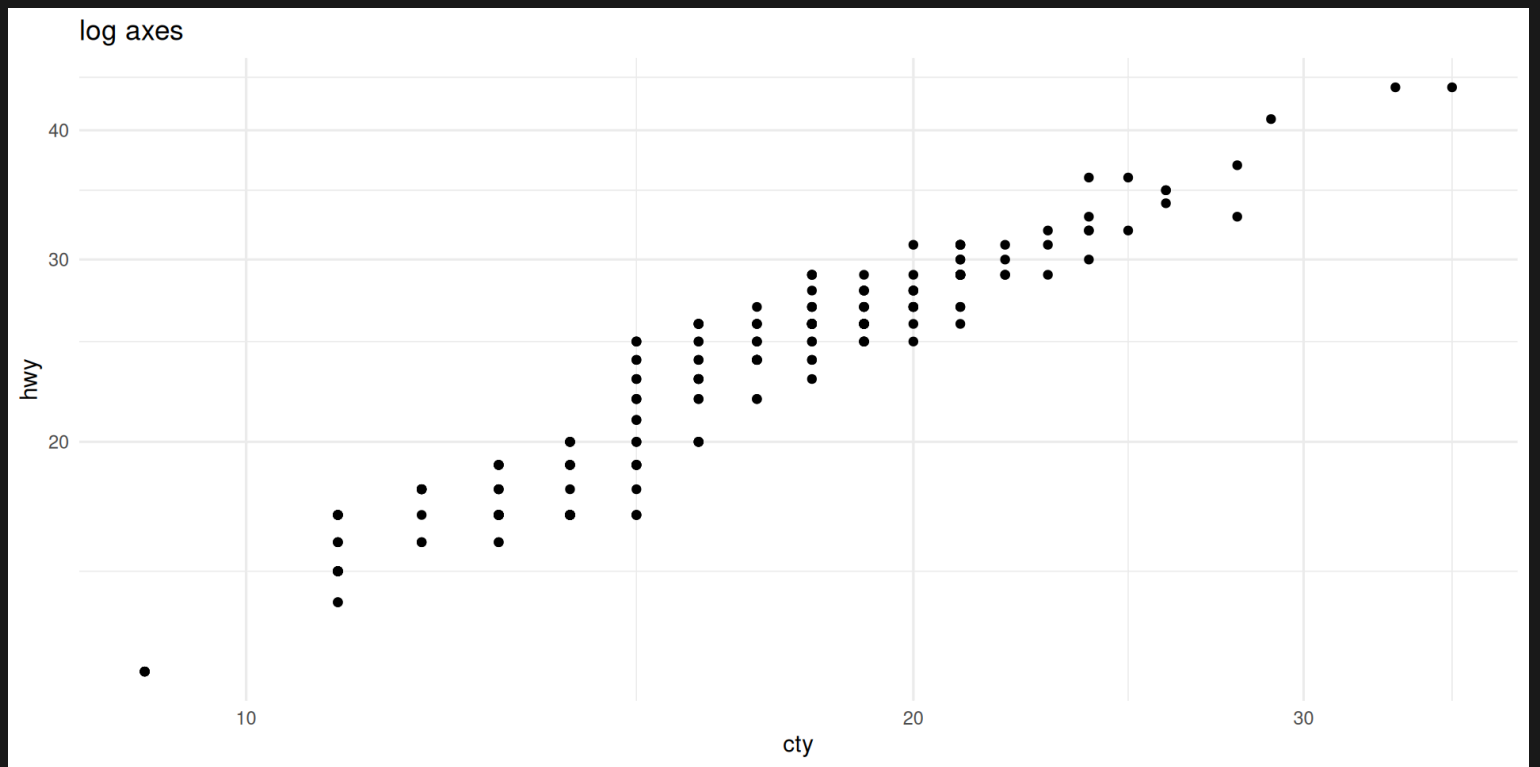
```
1 ggplot(mpg, aes(cty, hwy)) + geom_point() +
2   coord_fixed(ratio = 1/5) + labs(title = "City is 5 times as big as highwa
```



# coordinate transformations

## log scales

```
1  ggplot(mpg, aes(cty, hwy)) + geom_point() +
2    coord_trans(x = "log10", y = "log10")+ labs(title = "log axes")
```



# polar coordinates

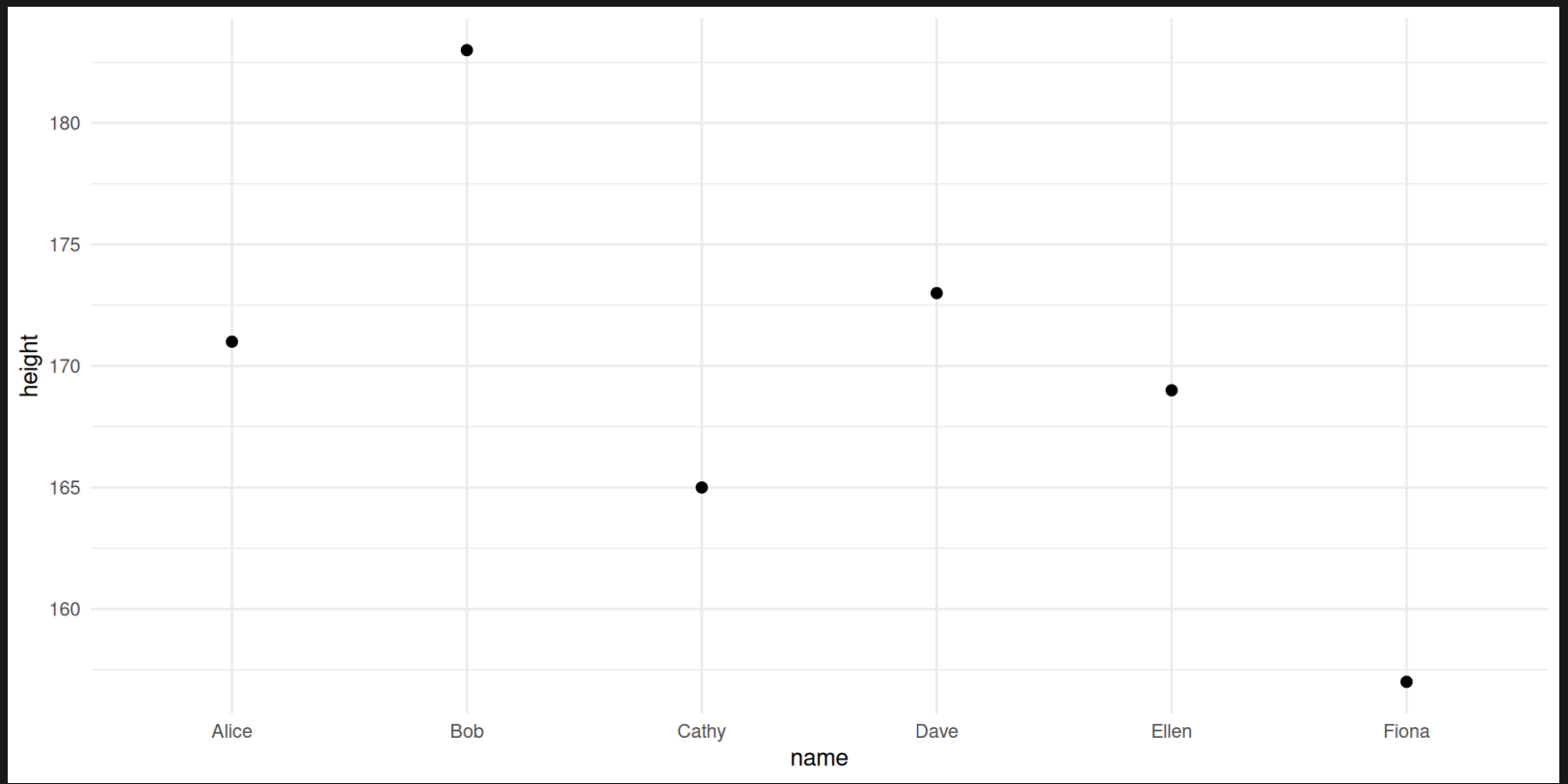
> we can also transform a (x,y) coordinate into a (distance, angle)

```
1  base + coord_polar()
```

# What are polar coordinates? /1

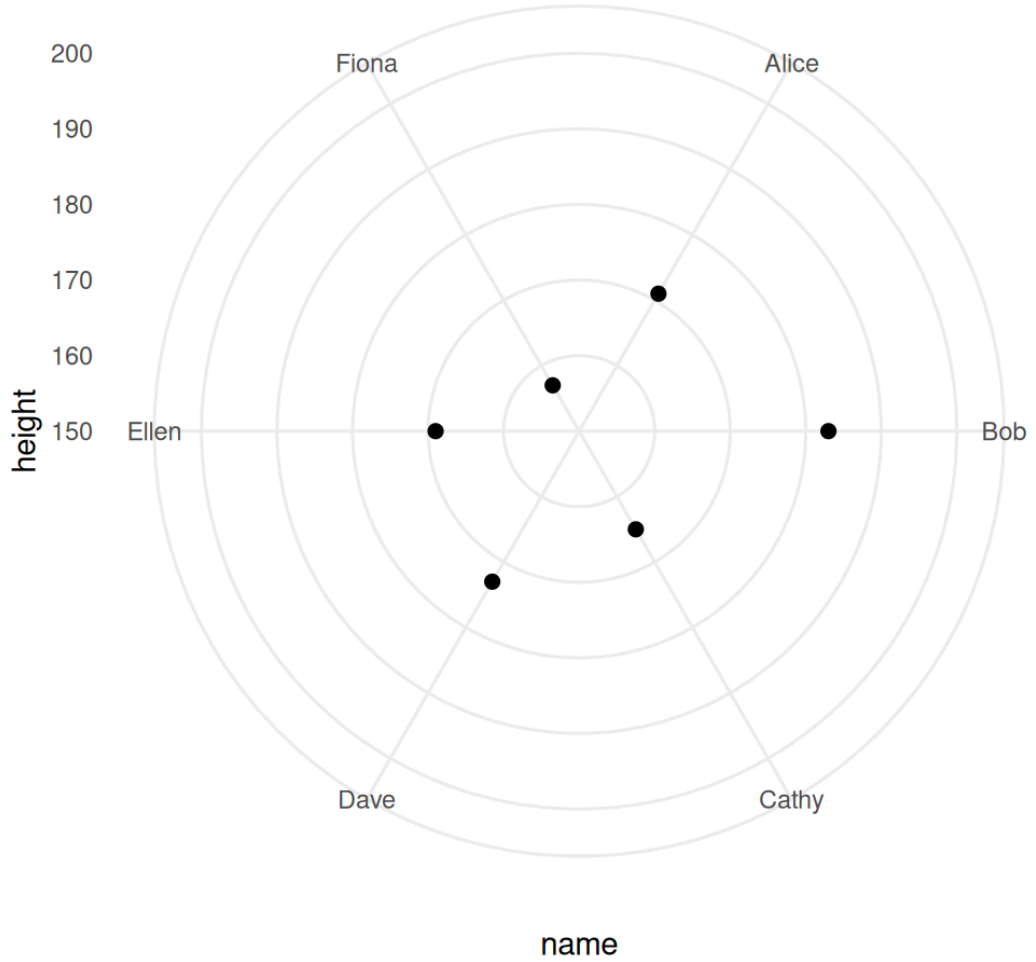
```r
df1 <- tibble(name = c("Alice", "Bob", "Cathy", "Dave", "Ellen", "Fiona"),
              height = c(171, 183, 165, 173, 169, 157))

plot <- df1 %>% ggplot() + aes(name, height) + geom_point(size = 2)
plot
```



# What are polar coordinates? /2

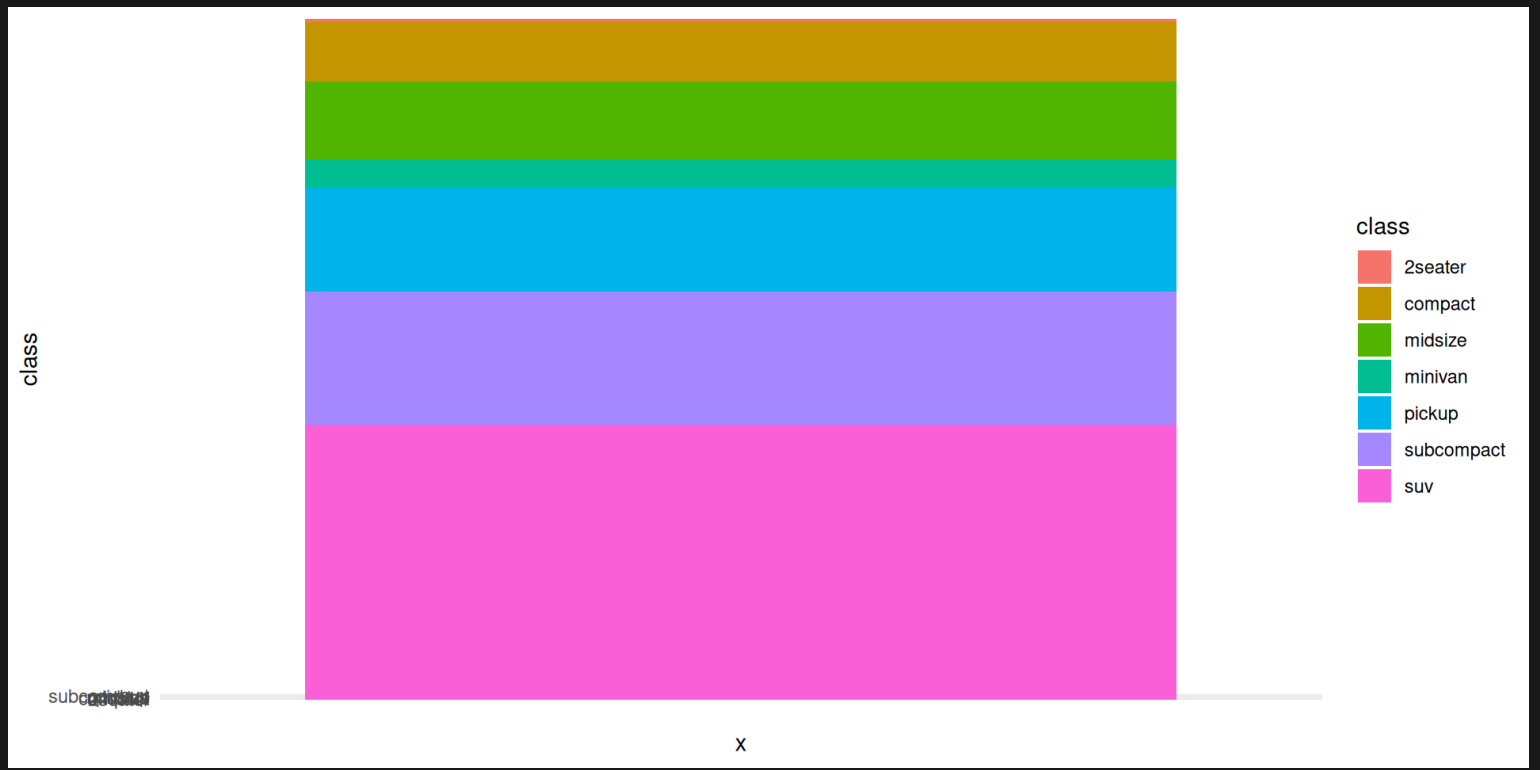
```
1  plot + coord_polar(clip = "off") + scale_y_continuous(limits = c(150,200))
```



# Polar coordinates: pie charts

> a pie is a barplot in polar coordinates

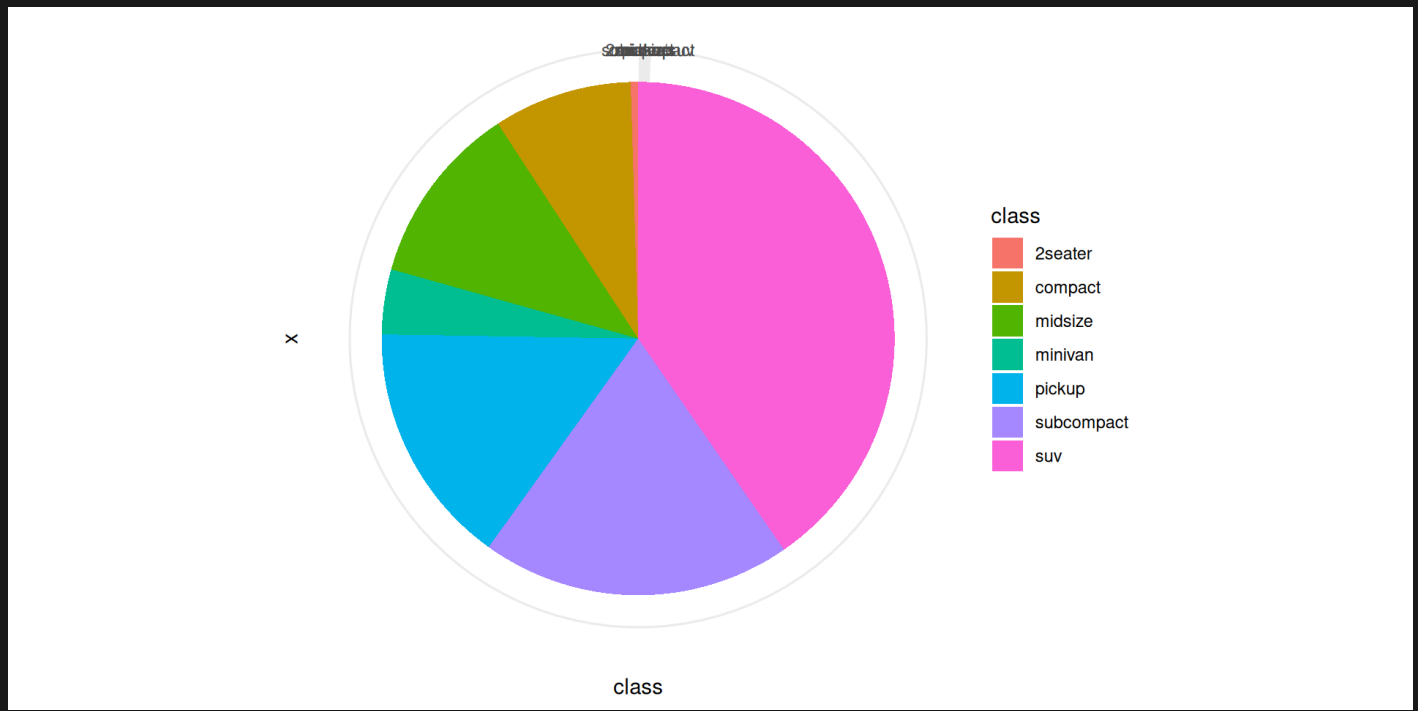
```
1 ggplot(mpg, aes(x = "", y = class, fill = class)) +
2   geom_col()
```



# Polar coordinates: pie charts

> a pie is a barplot in polar coordinates

```
1 ggplot(mpg, aes(x = "", y = class, fill = class)) +
2   geom_col() +
3   coord_polar("y", start = 0)
```



# 2 · Scales

# Scale functions

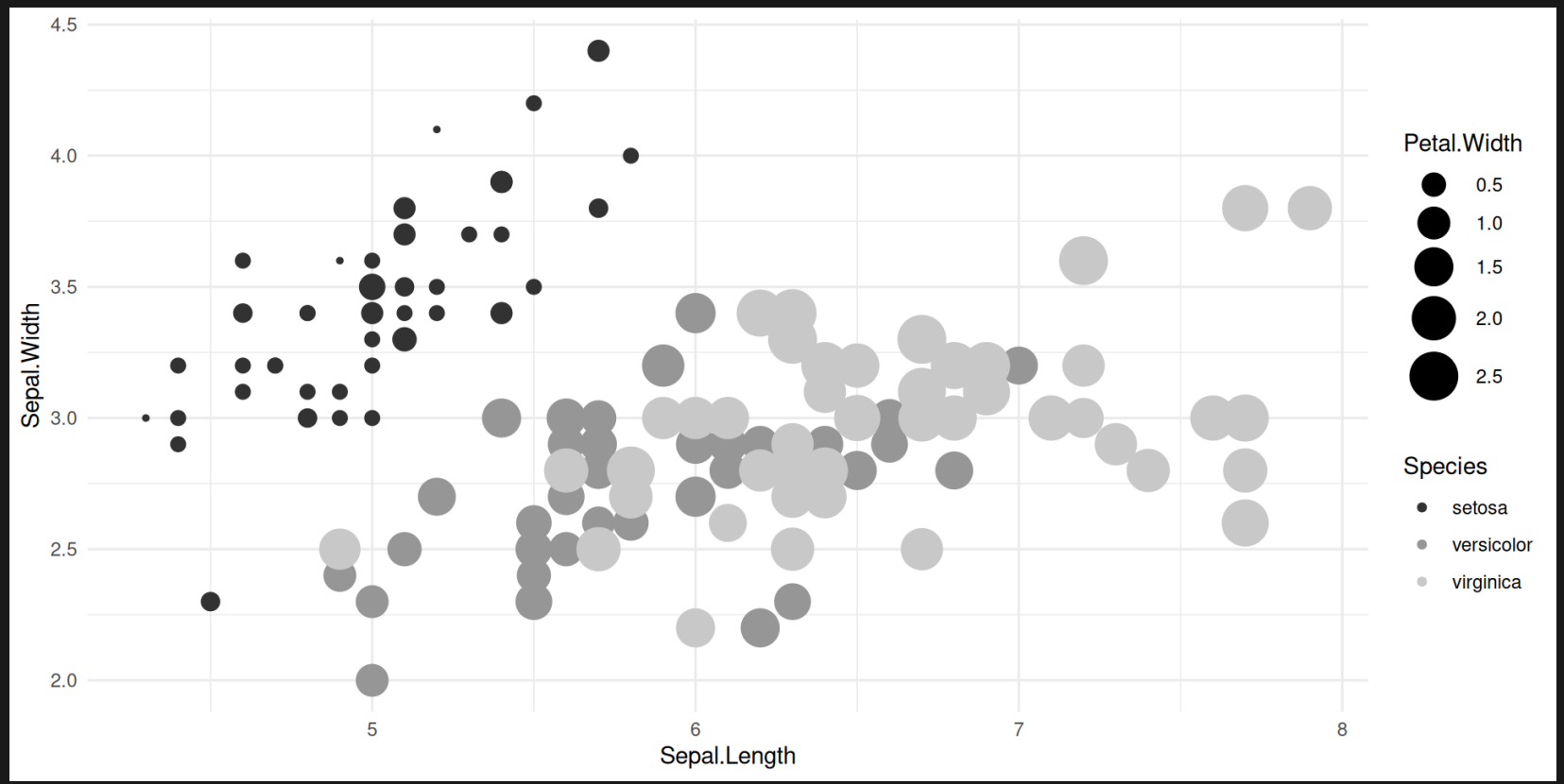
> `scale_*_*` change the **appearance** of the **mapping**

- general usage: `scale_NAME_TYPE`
- `NAME` can take: `alpha`, `color`, `fill`, `linetype`, `shape`, `size`, `x`, `y`
- `TYPE` can take different values according to the scale type

# A simple example

> make our base plot greyscale and change the size mapping

```
1  base + scale_color_grey() + scale_size_continuous(range = c(1, 10))
```



# Changing colors: `scale_color_*`

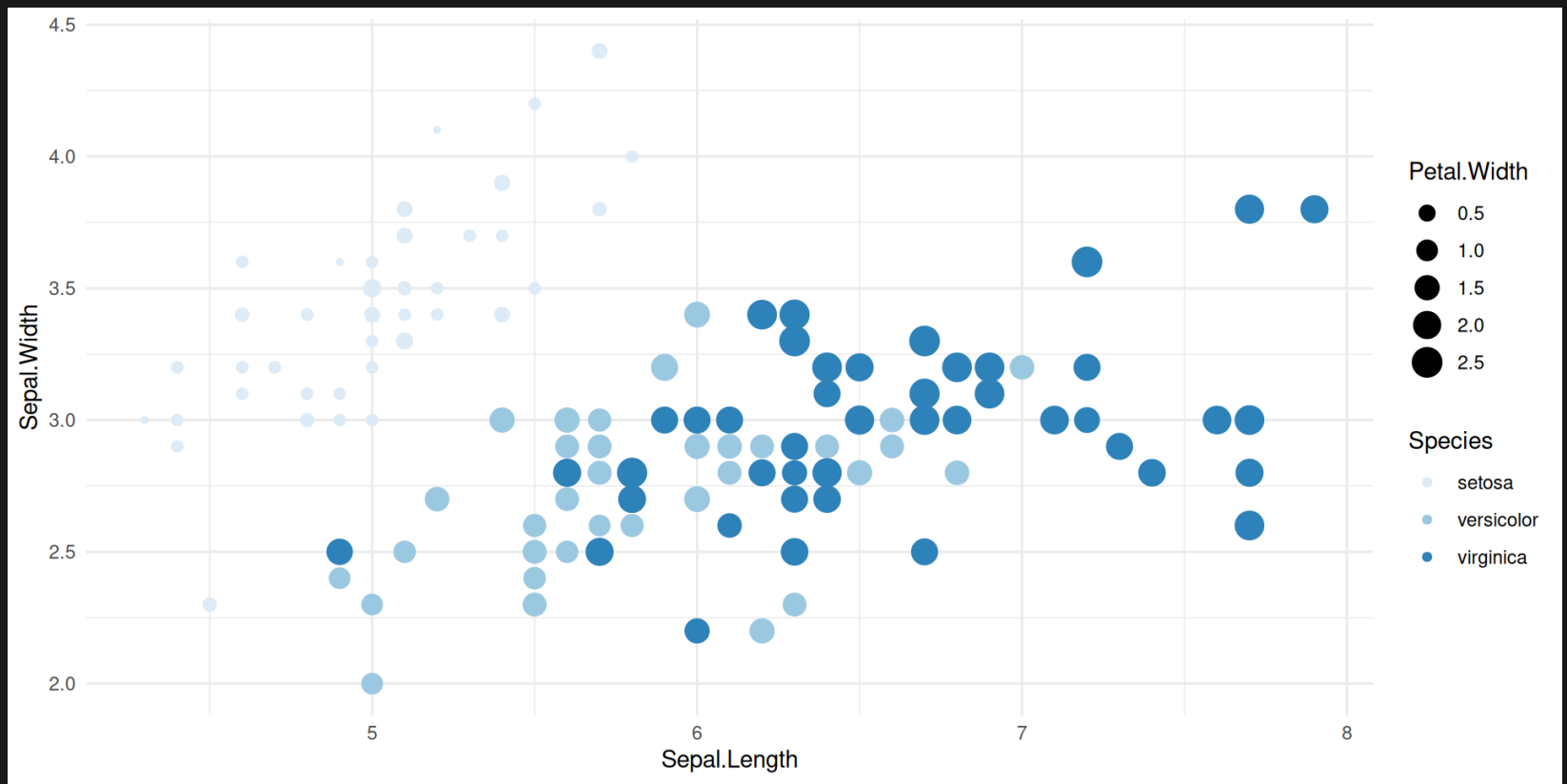
> Manipulating colors is fun! but beware:

- is your color variable **continuous** or **discrete**?
- do you want a **gradient**?
- can **colorblind** people use your plot?

# Discrete color scales

## `brewer` palettes

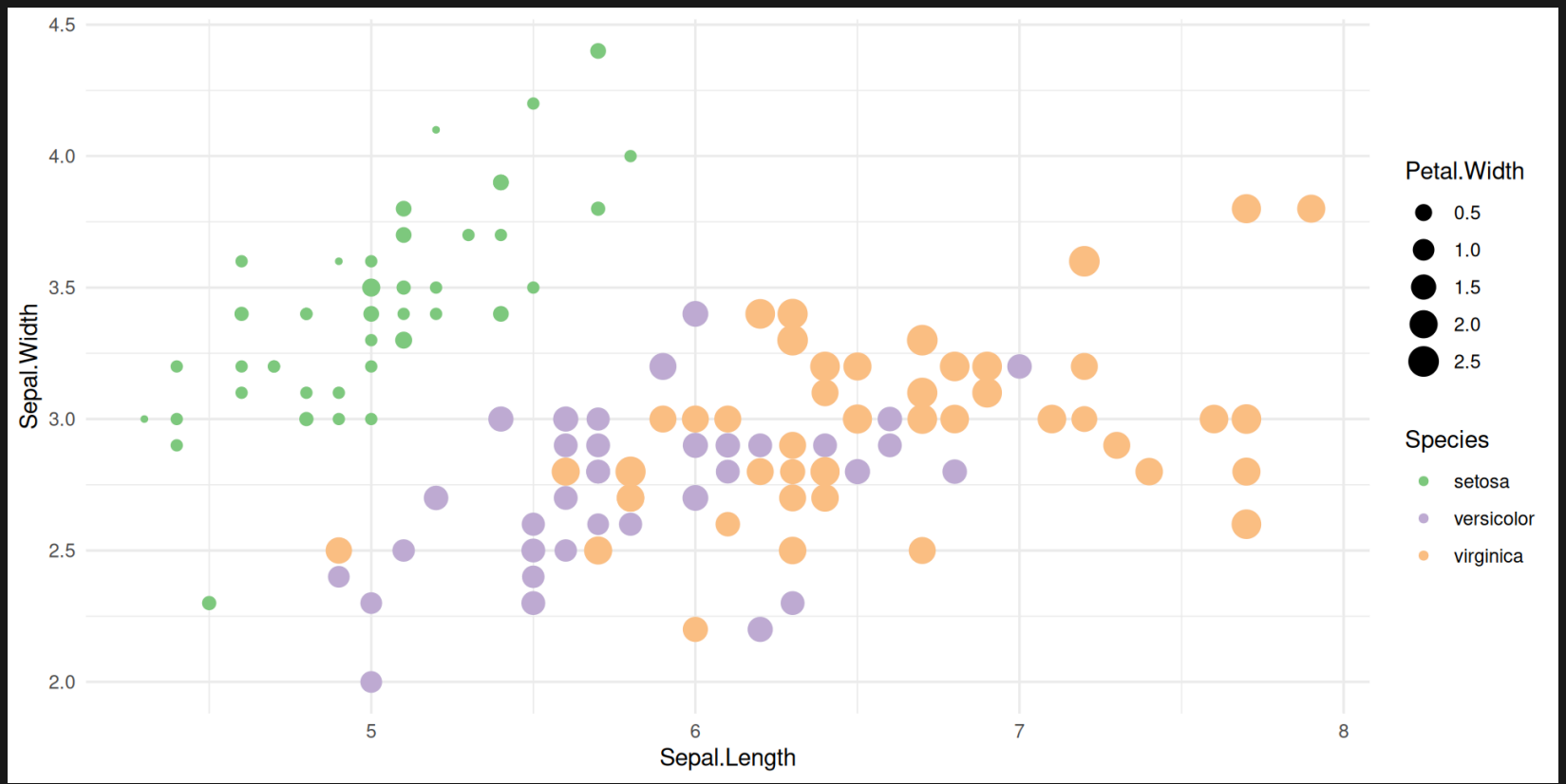
```
1  base + scale_color_brewer()
```



# Discrete color scales

## `brewer` palettes

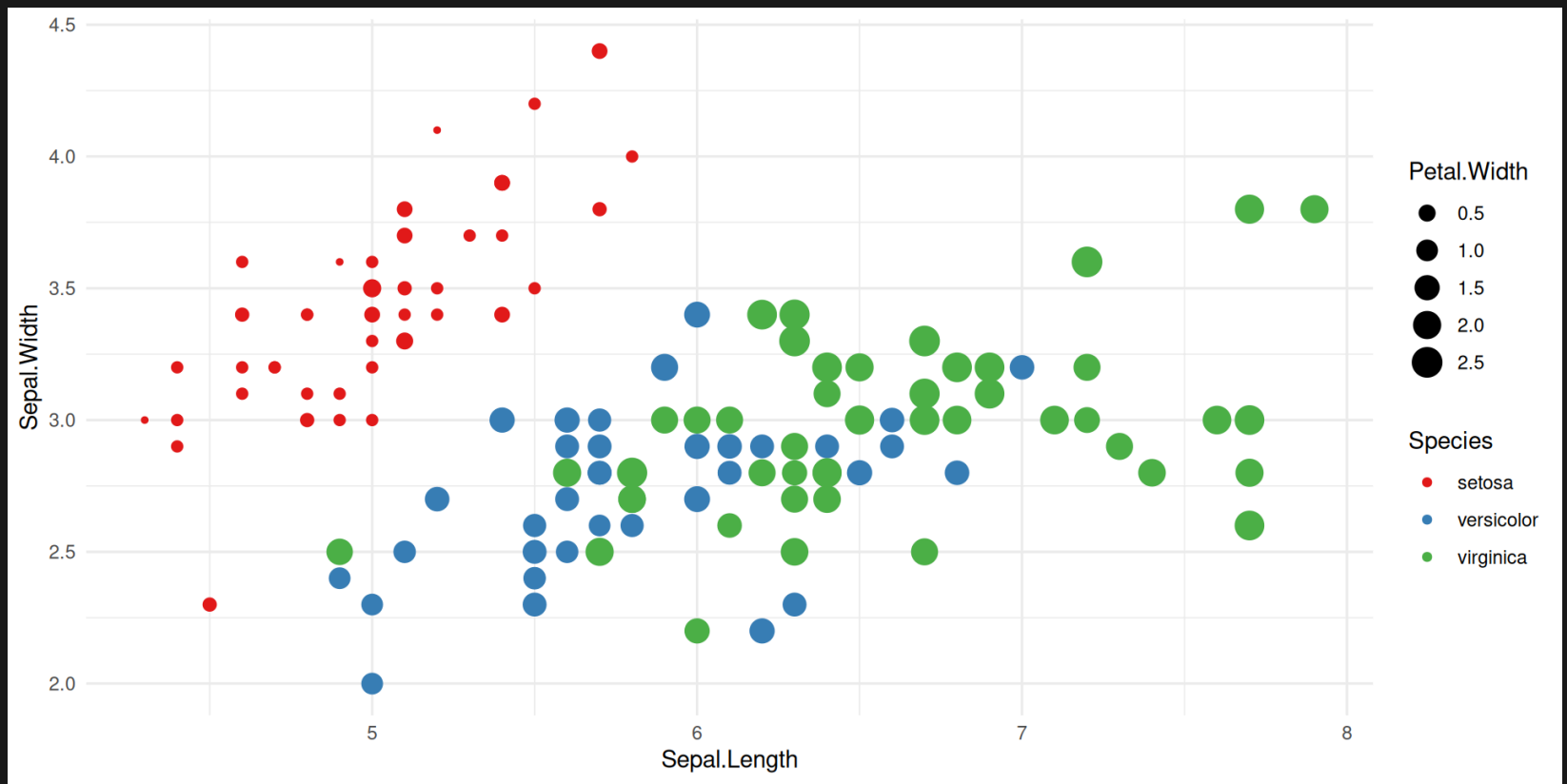
```
1  base + scale_color_brewer(type = "qual")
```



# Discrete color scales

> `brewer` has many palettes

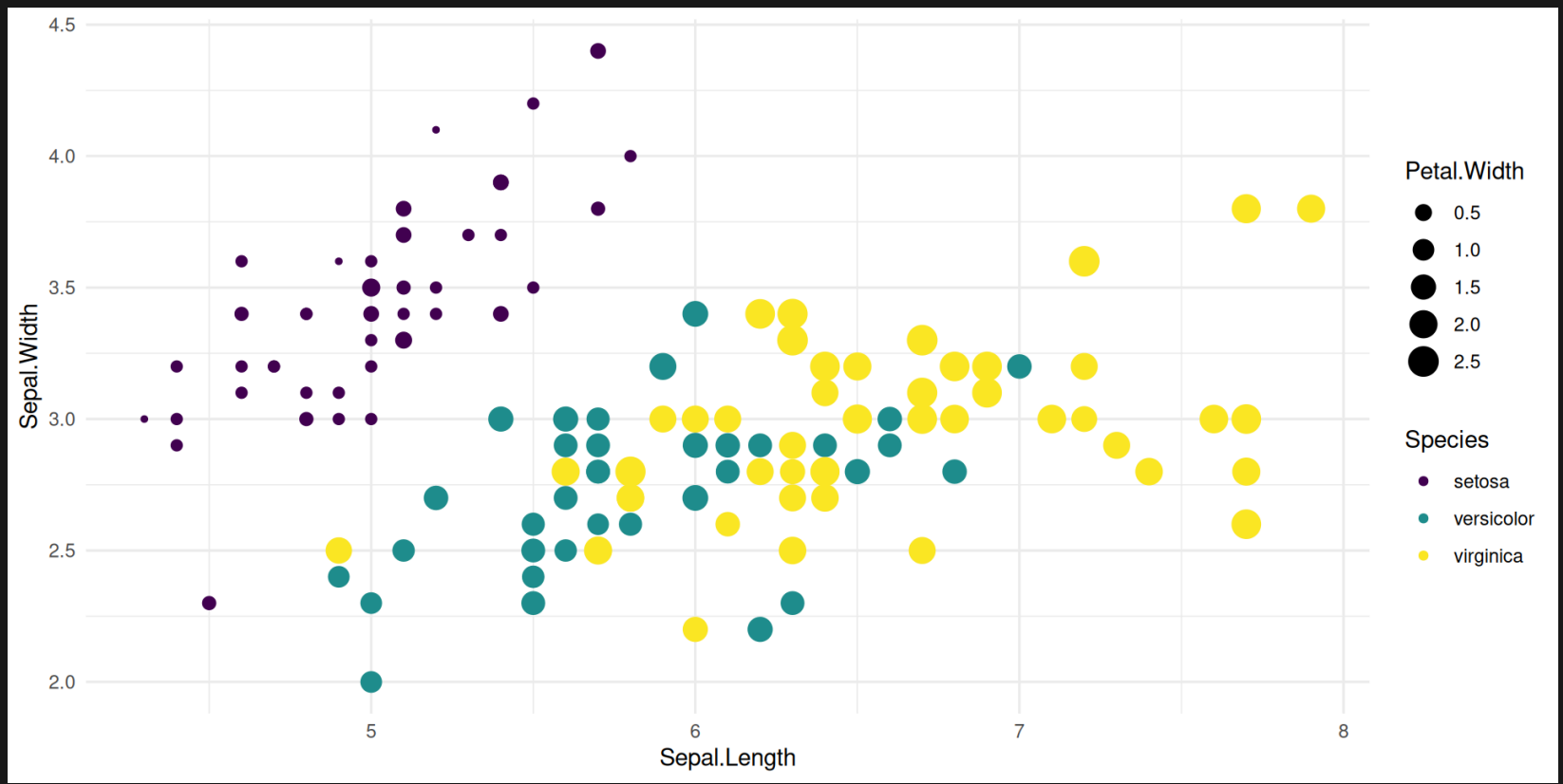
```
1  base + scale_color_brewer(palette = "Set1")
```



# Discrete color scales

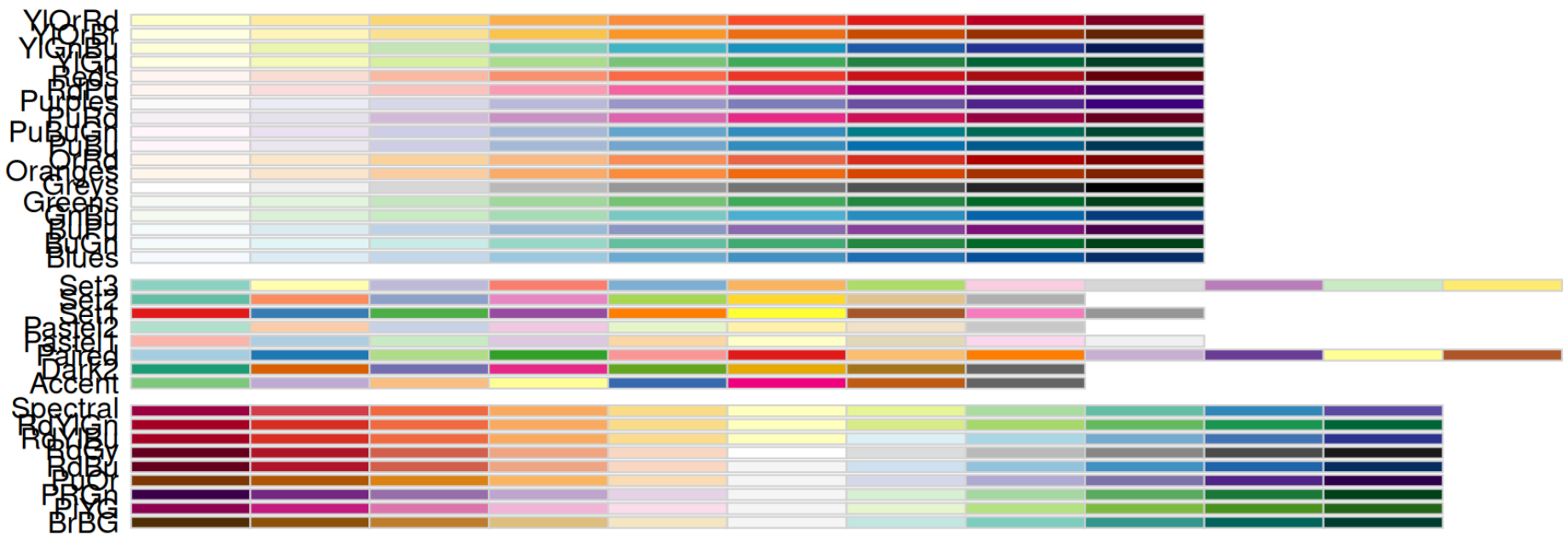
> `viridis` palette (for colorblind)

```
1  base + scale_color_viridis_d()
```

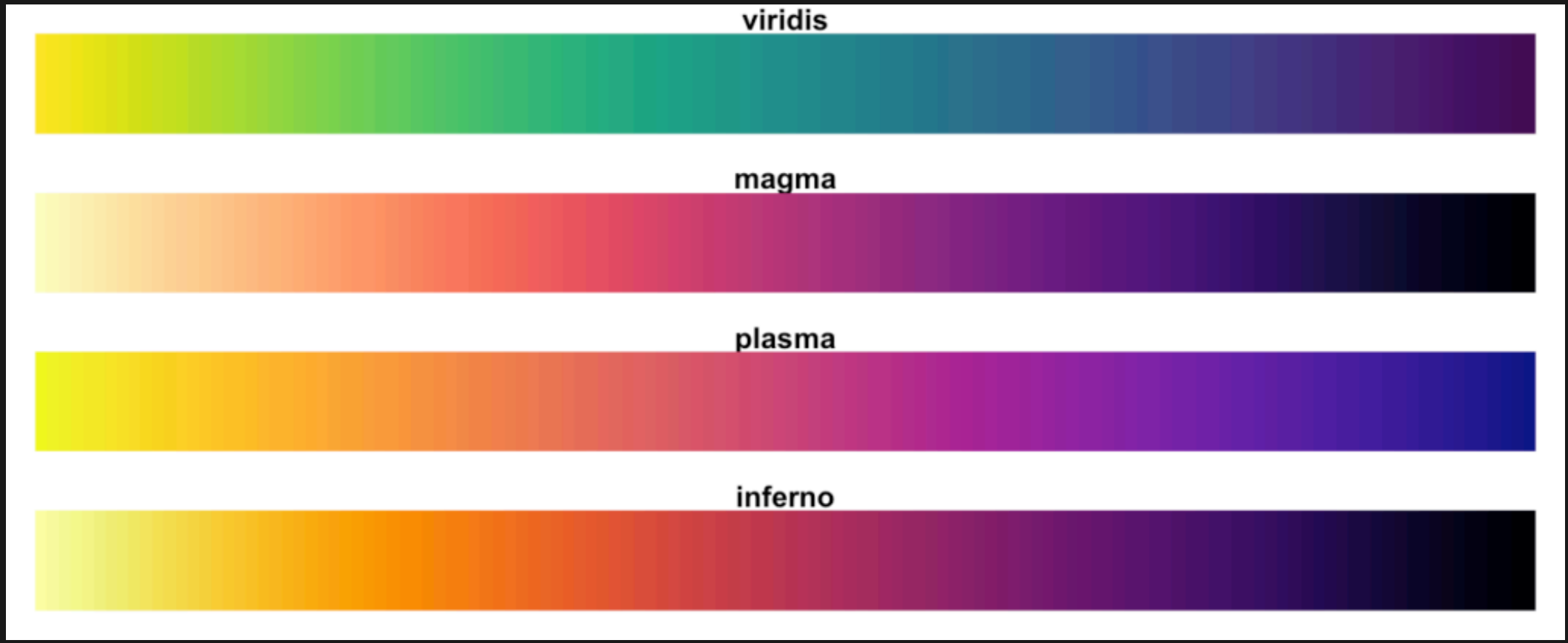


# Info on color palettes: brewer

```
1  RColorBrewer::display.brewer.all()
```

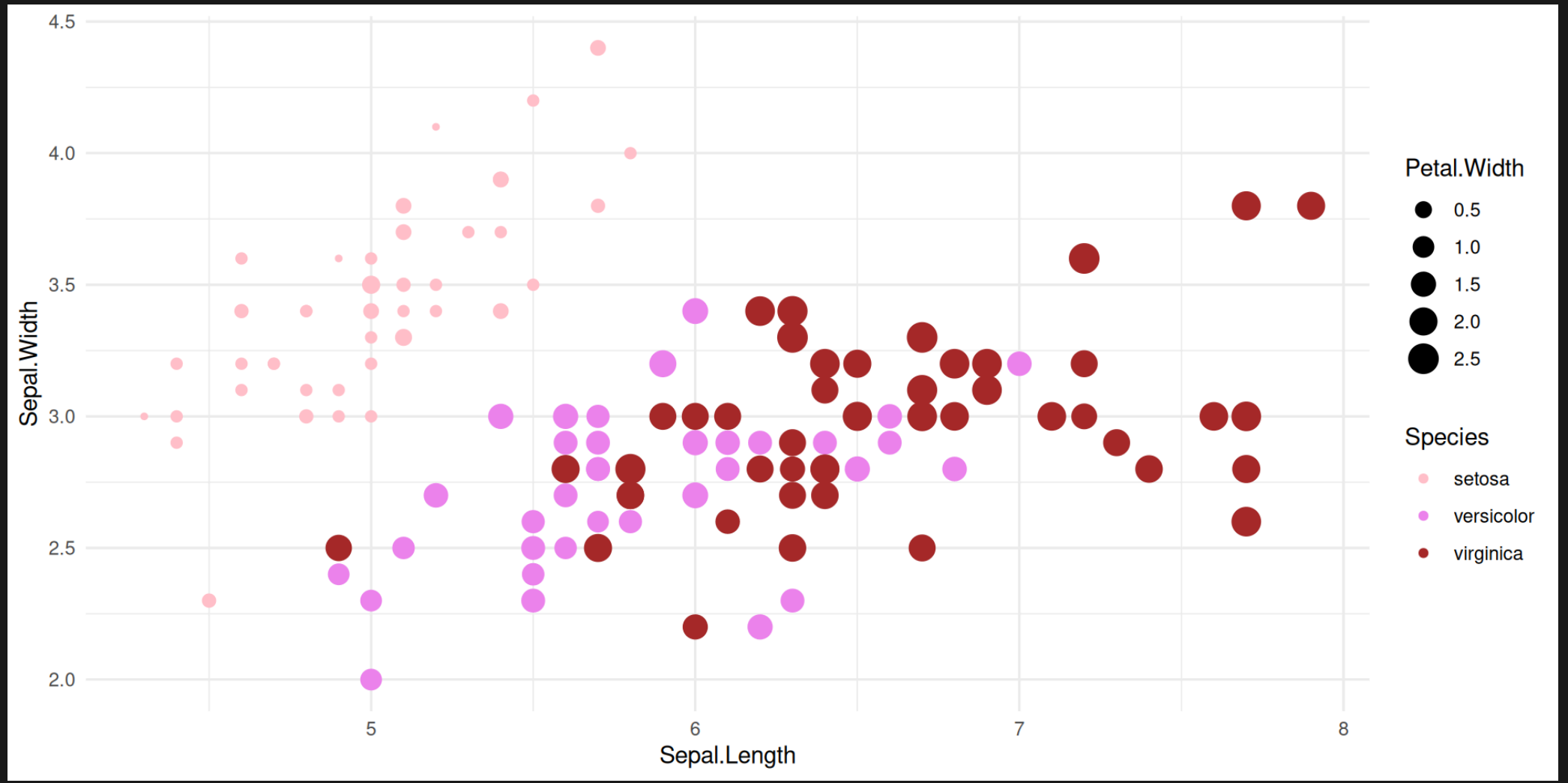


# Info on color palettes: `viridis`



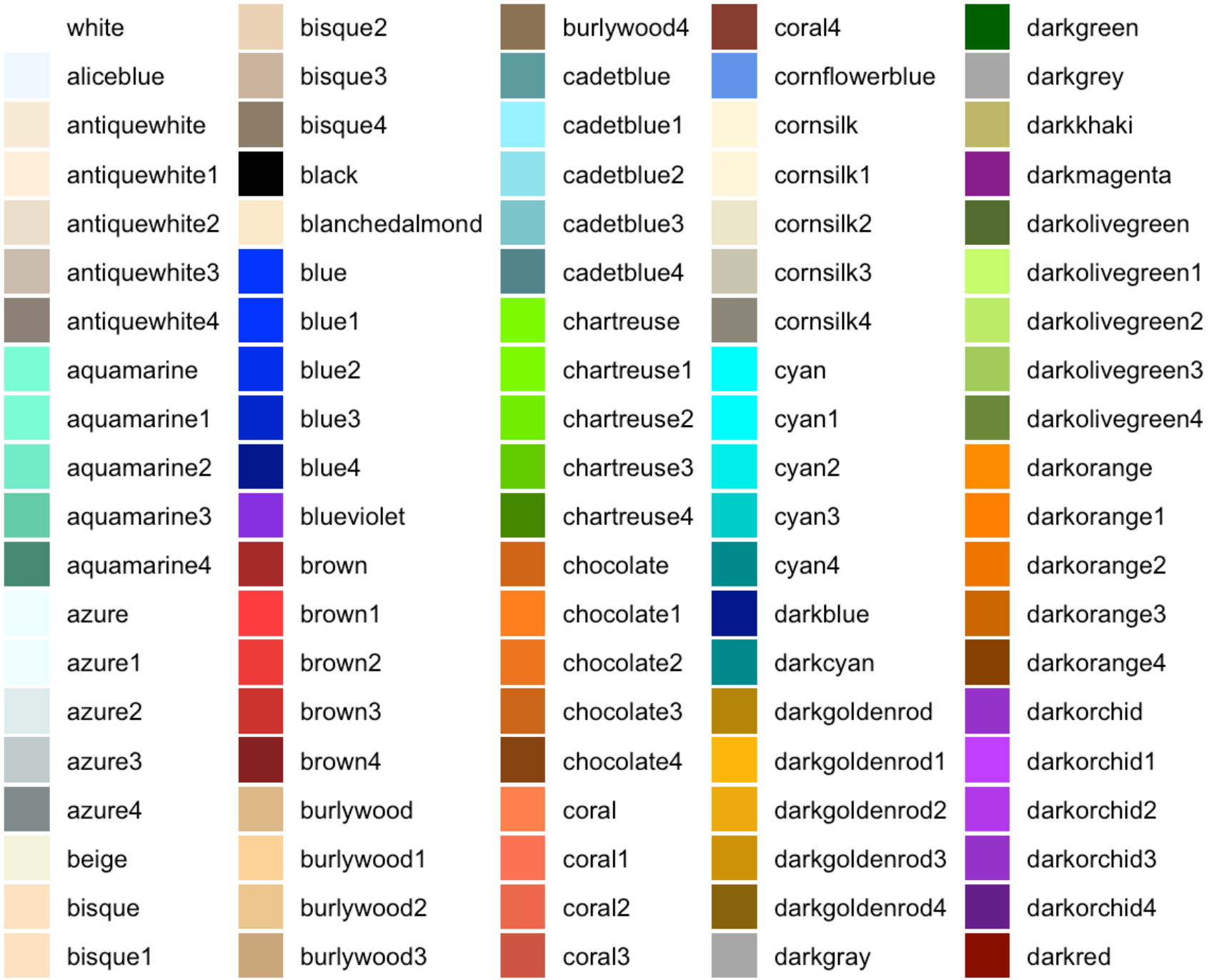
# Still not enough colors? manual scales

```
1  base + scale_color_manual(values = c("pink", "violet", "brown"))
```



# Colors in R

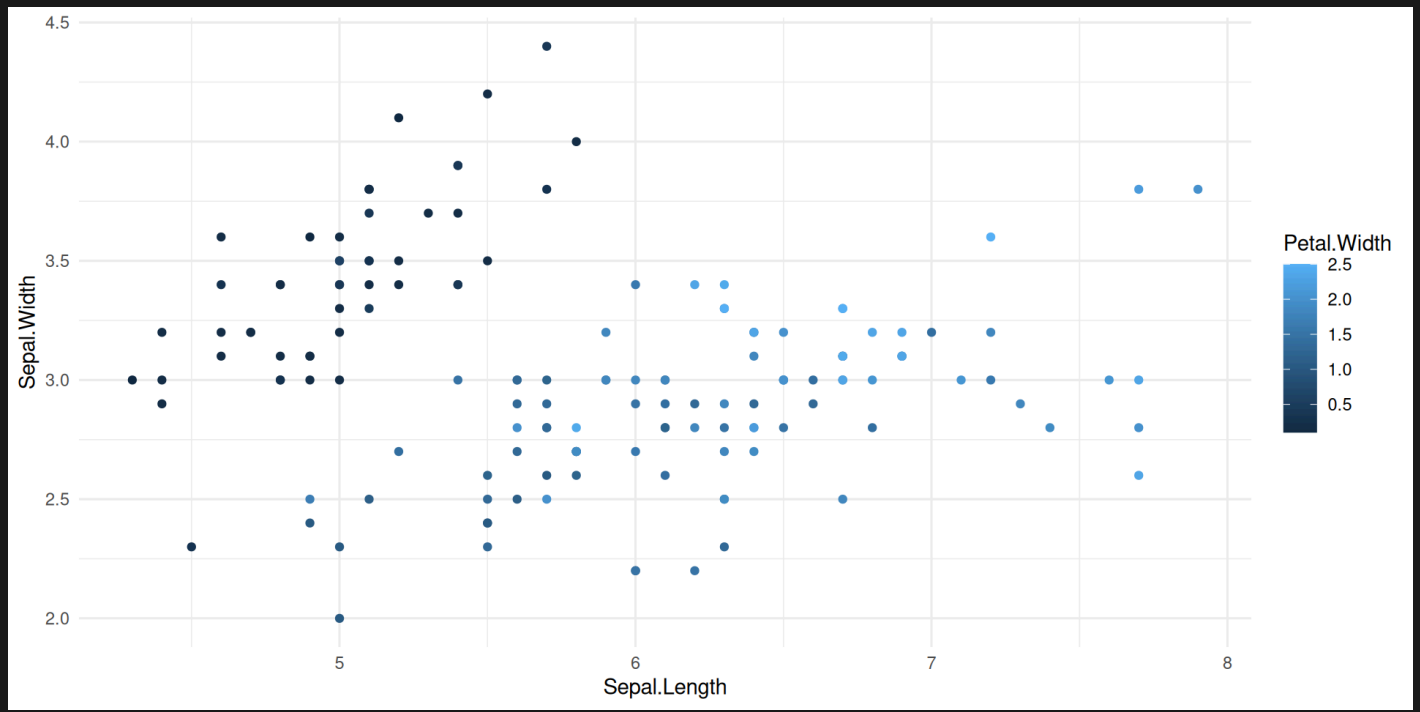
#1


white
aliceblue
antiquewhite
antiquewhite1
antiquewhite2
antiquewhite3
antiquewhite4
aquamarine
aquamarine1
aquamarine2
aquamarine3
aquamarine4
azure
azure1
azure2
azure3
azure4
beige
bisque
bisque1
bisque2
bisque3
bisque4
black
blanchedalmond
blue
blue1
blue2
blue3
blue4
blueviolet
brown
brown1
brown2
brown3
brown4
burlywood
burlywood1
burlywood2
burlywood3
burlywood4
coral4
cornflowerblue
cornsilk
cornsilk1
cornsilk2
cornsilk3
cornsilk4
cyan
cyan1
cyan2
cyan3
cyan4
darkblue
darkcyan
darkgoldenrod
darkgoldenrod1
darkgoldenrod2
darkgoldenrod3
darkgoldenrod4
darkgray
darkgreen
darkgrey
darkkhaki
darkmagenta
darkolivegreen
darkolivegreen1
darkolivegreen2
darkolivegreen3
darkolivegreen4
darkorange
darkorange1
darkorange2
darkorange3
darkorange4
darkorchid
darkorchid1
darkorchid2
darkorchid3
darkorchid4
darkred
cadetblue
cadetblue1
cadetblue2
cadetblue3
cadetblue4
chartreuse
chartreuse1
chartreuse2
chartreuse3
chartreuse4
chocolate
chocolate1
chocolate2
chocolate3
chocolate4
coral
coral1
coral2
coral3

# Continuous color

> a simple plot to showcase stuff

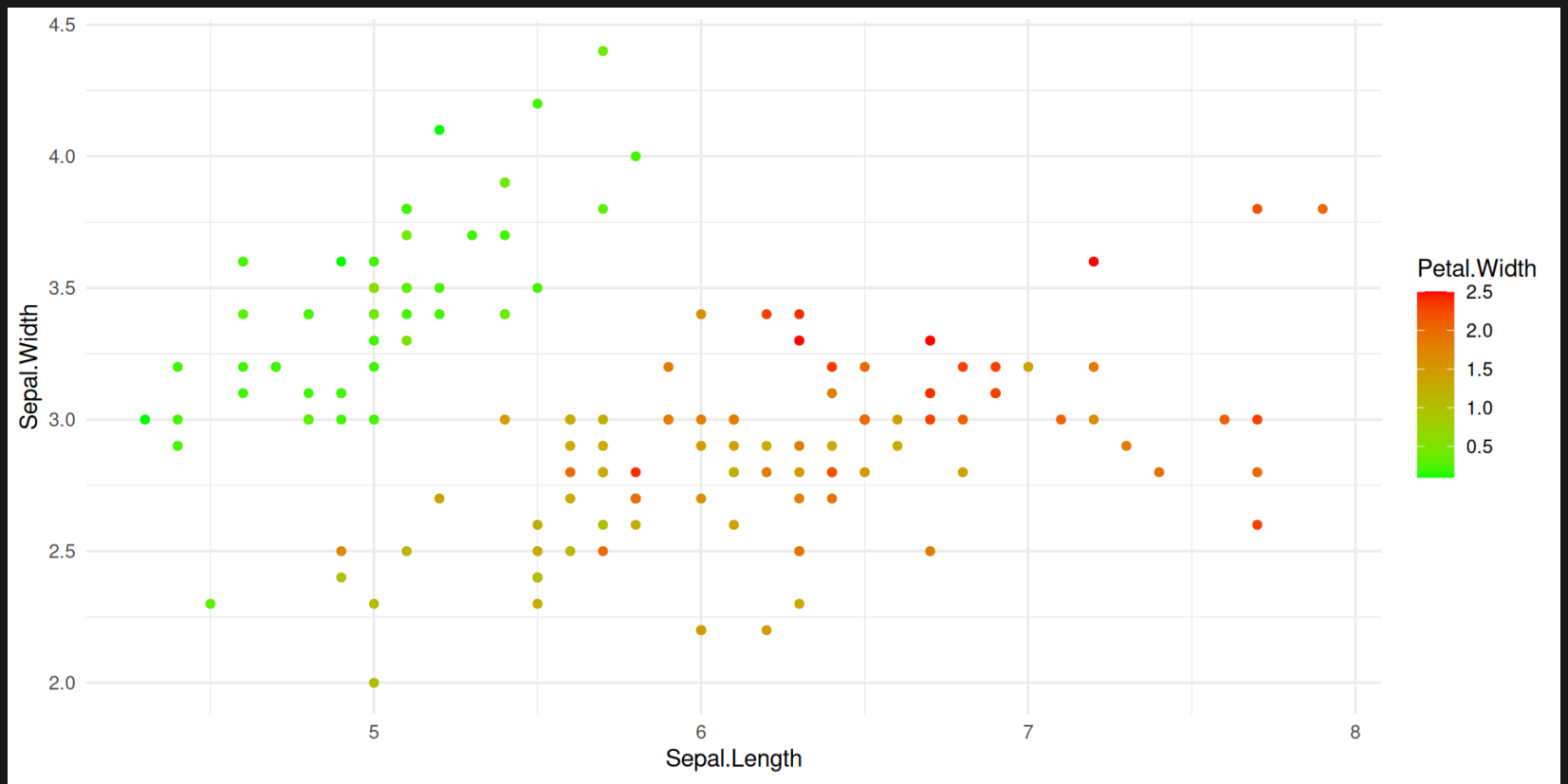
```
1 base2 <- ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Petal.
2   geom_point()
3 base2
```



# Continuous color: gradients

> change the gradient

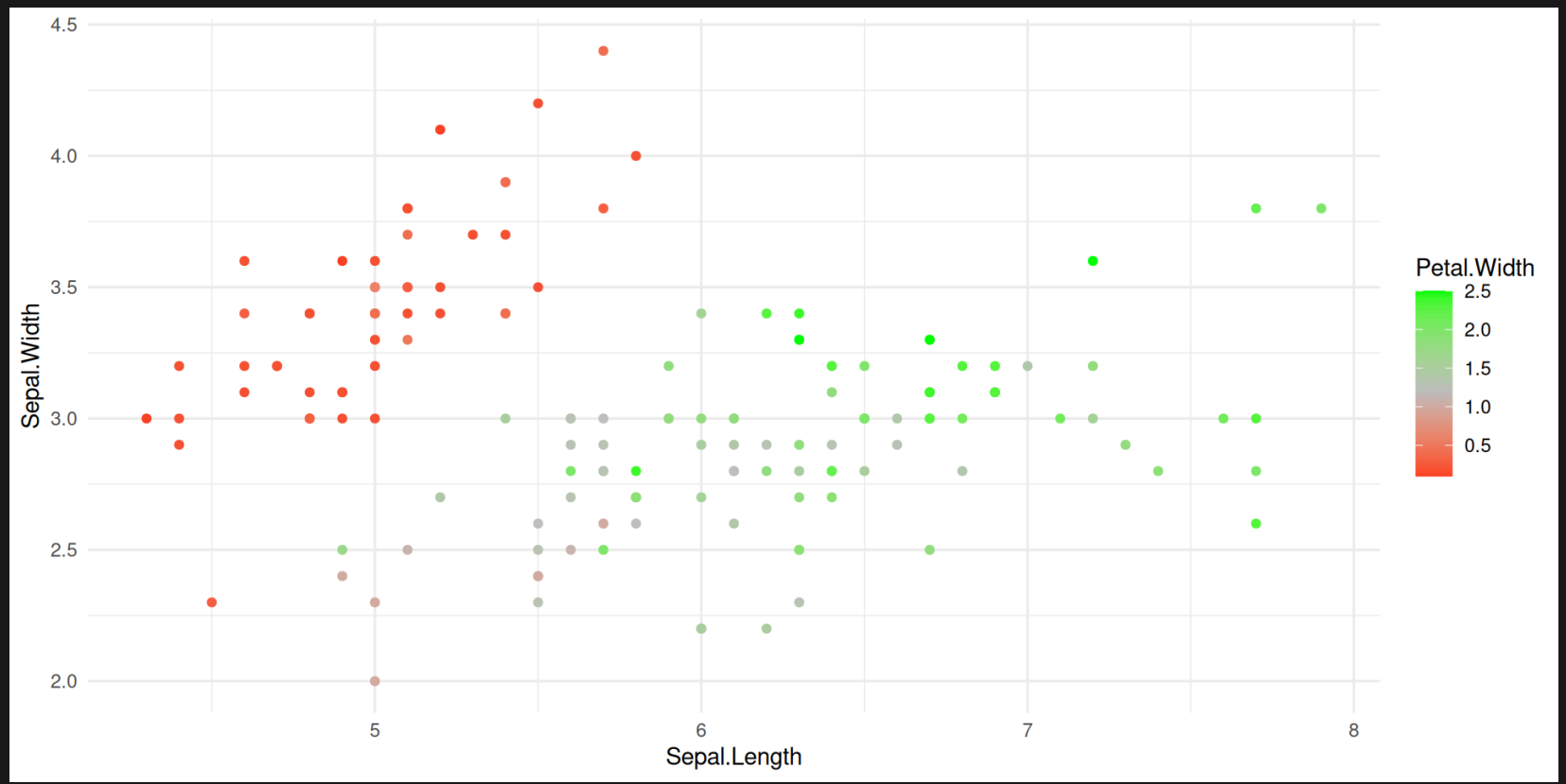
```
1  base2 + scale_color_gradient(low = "green", high = "red")
```



# Continuous color: gradients

> a gradient with an intermediate step

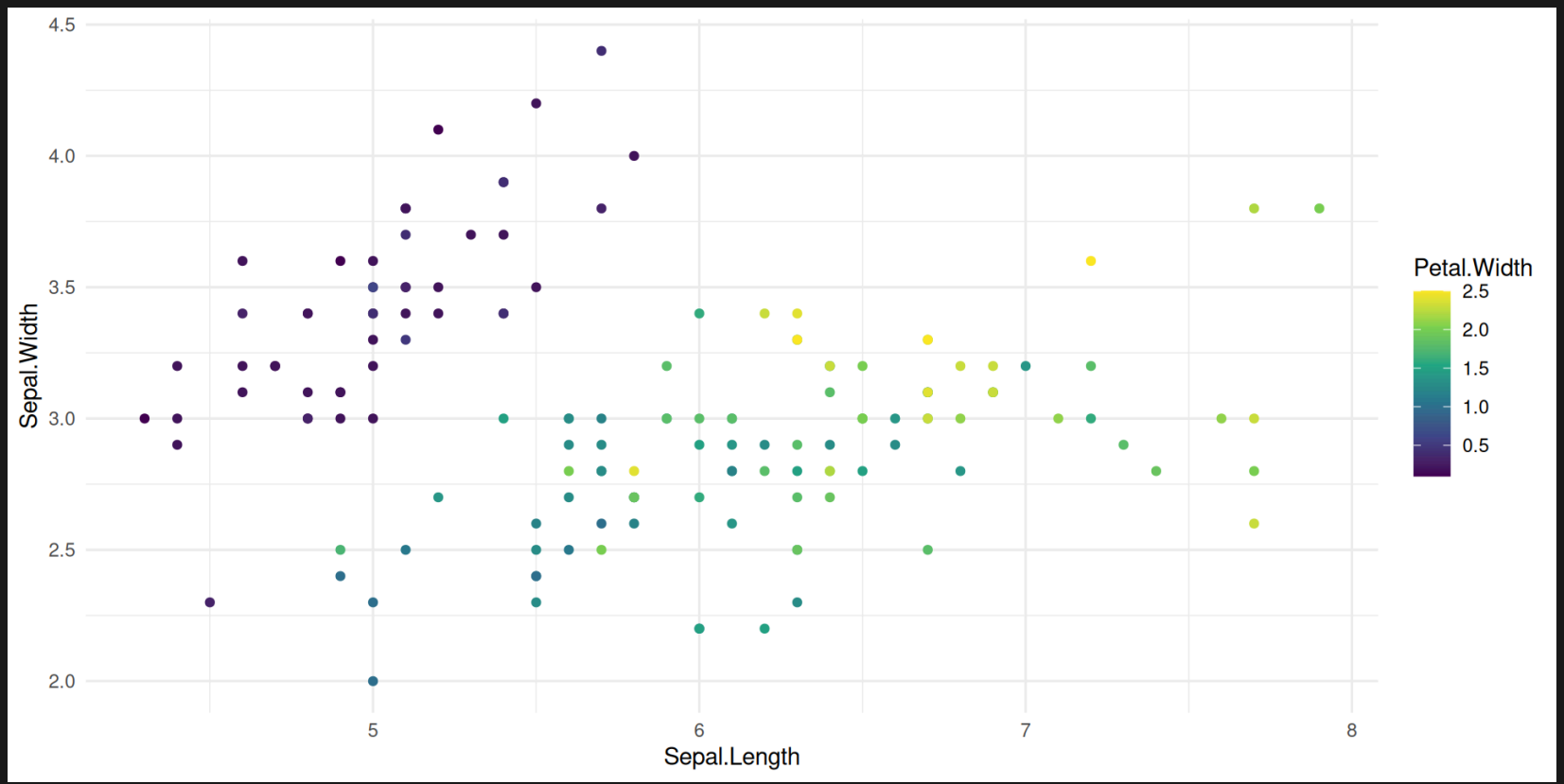
```
1  base2 + scale_color_gradient2(low = "red", mid = "grey", high = "green", mi
```



# Continuous color: gradients

> change the gradient to be colorblind proof

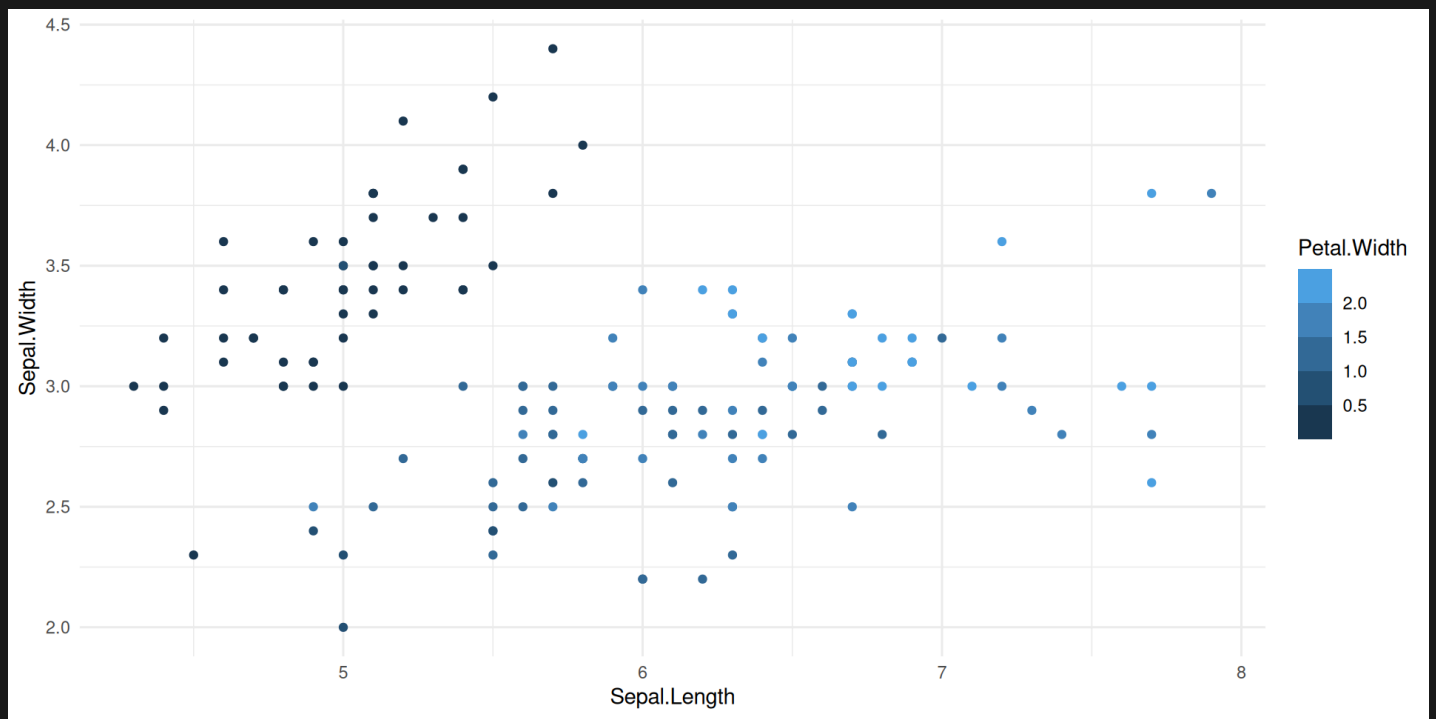
```
1  base2 + scale_color_viridis_c()
```



# Continuous *and* discrete: binned

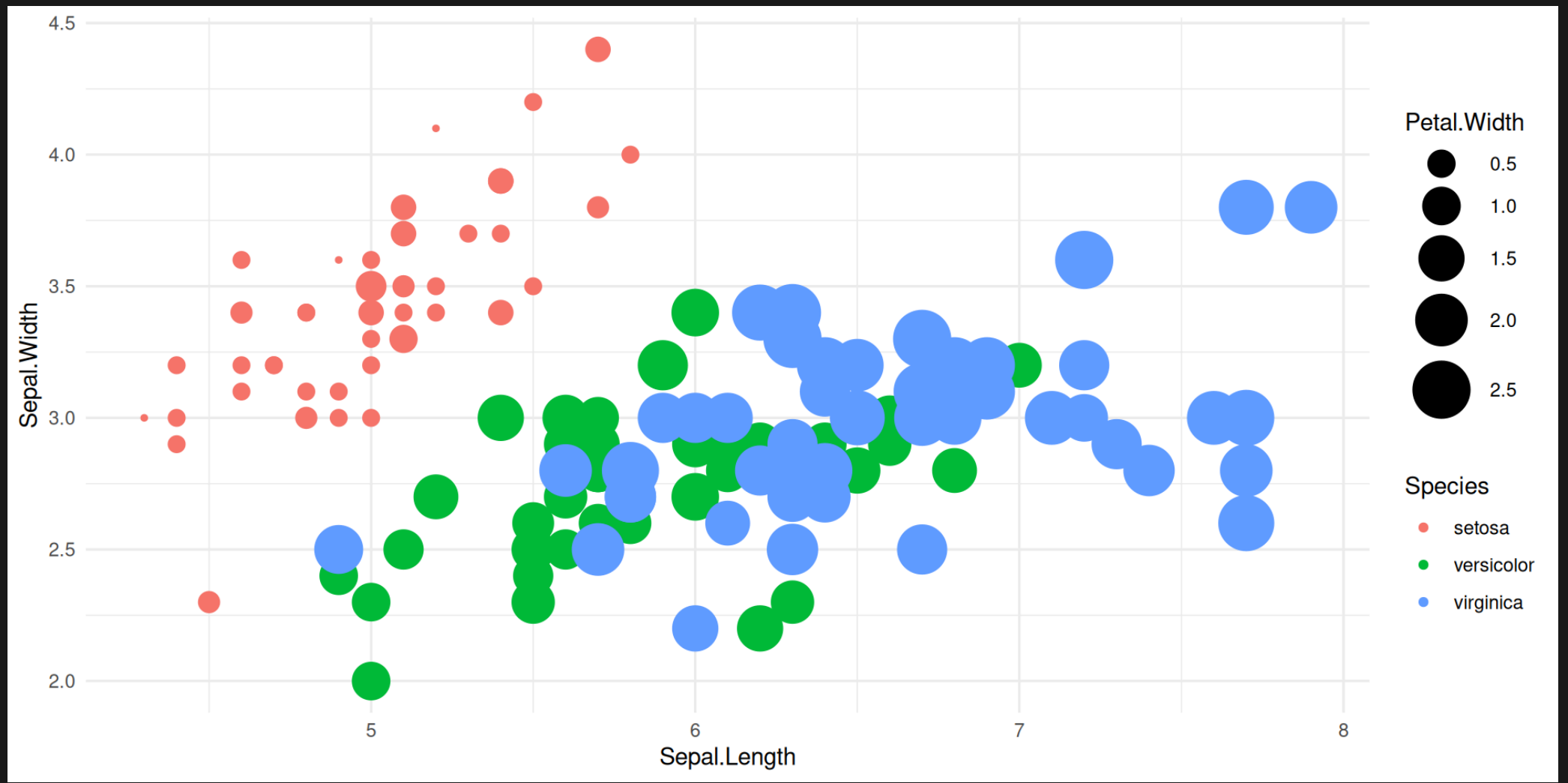
> You have continuous data but you want to treat it as discrete: `binned` scales

```
1  base2 + scale_color_binned()
```



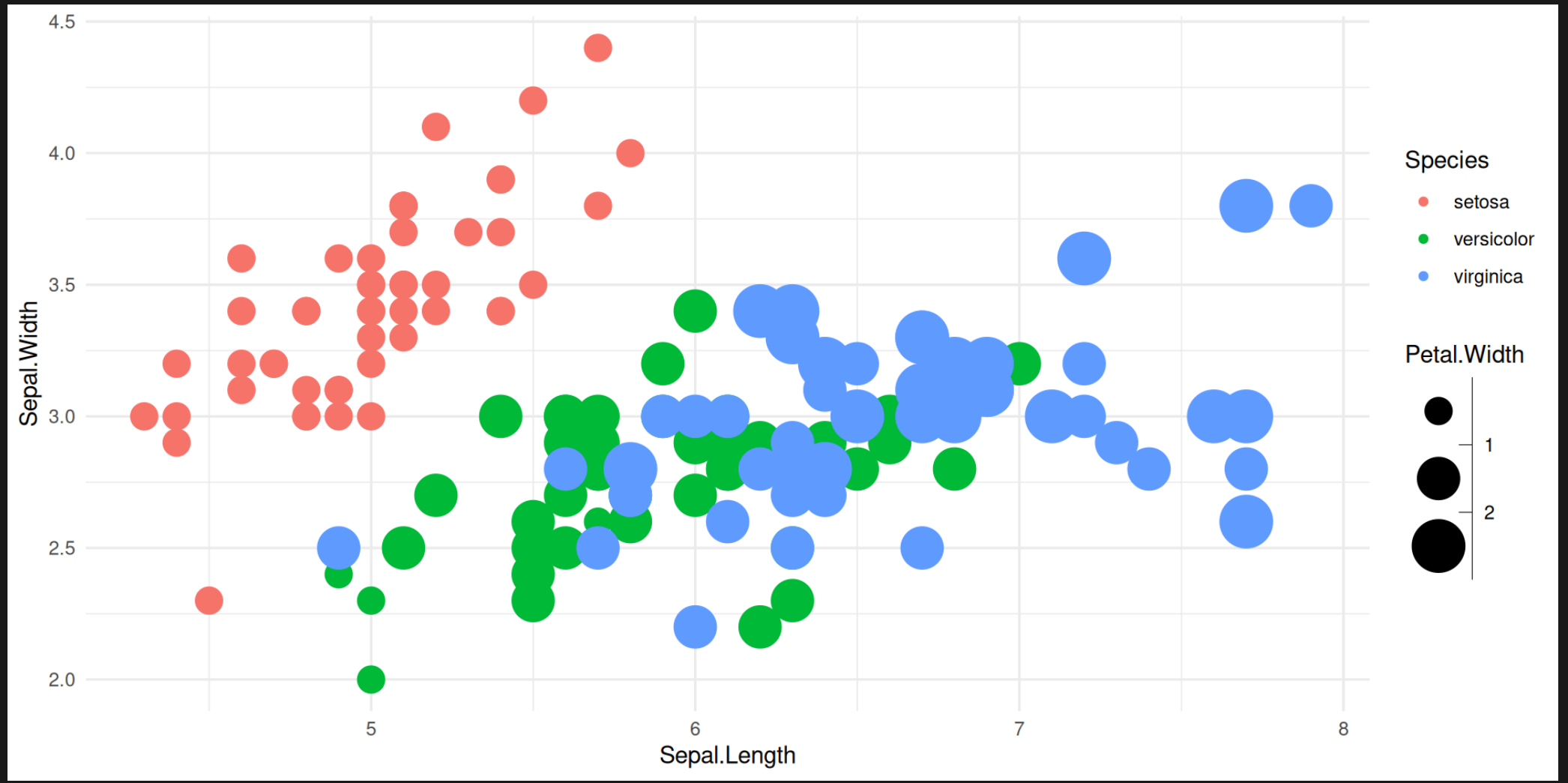
# Continuous size

```
1  base + scale_size_continuous(range = c(1,12))
```



# Binned size

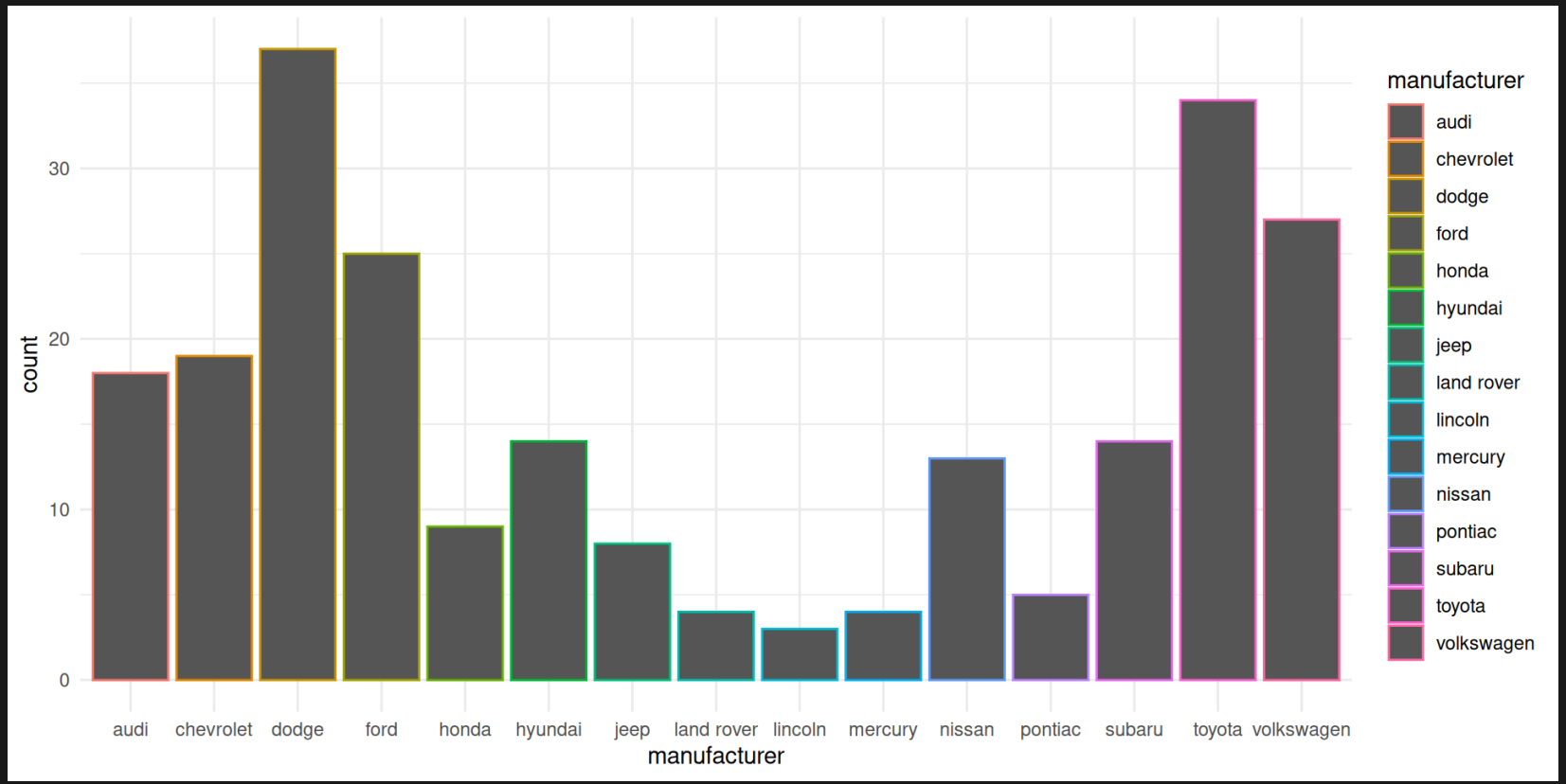
```
1  base + scale_size_binned(n.breaks = 3, range = c(1,12))
```



# Controlling barplots: fill

> in barplots, the relevant variable is the fill

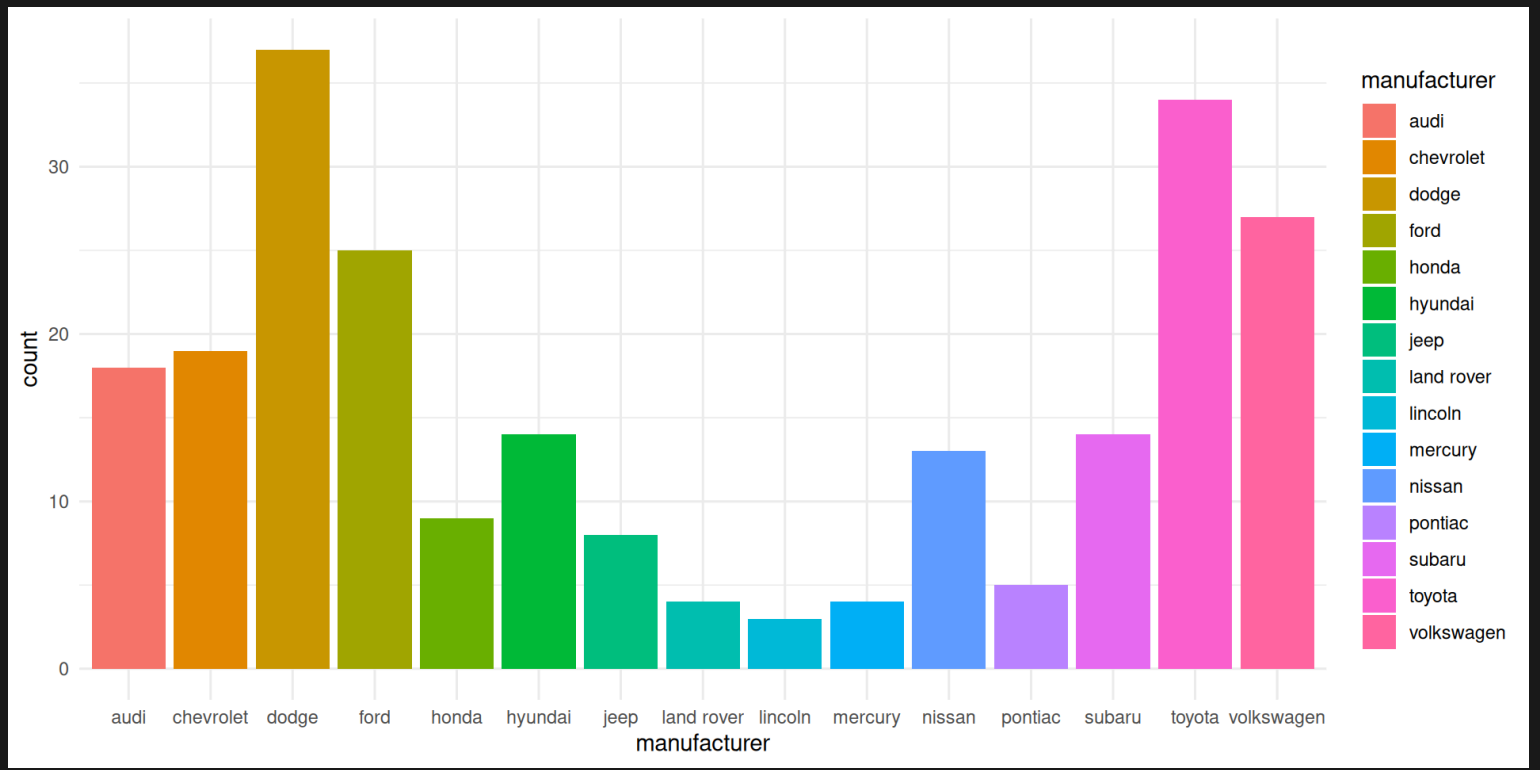
```
1  ggplot(mpg, aes(x = manufacturer, color = manufacturer)) + geom_bar()
```



# Controlling barplots: fill

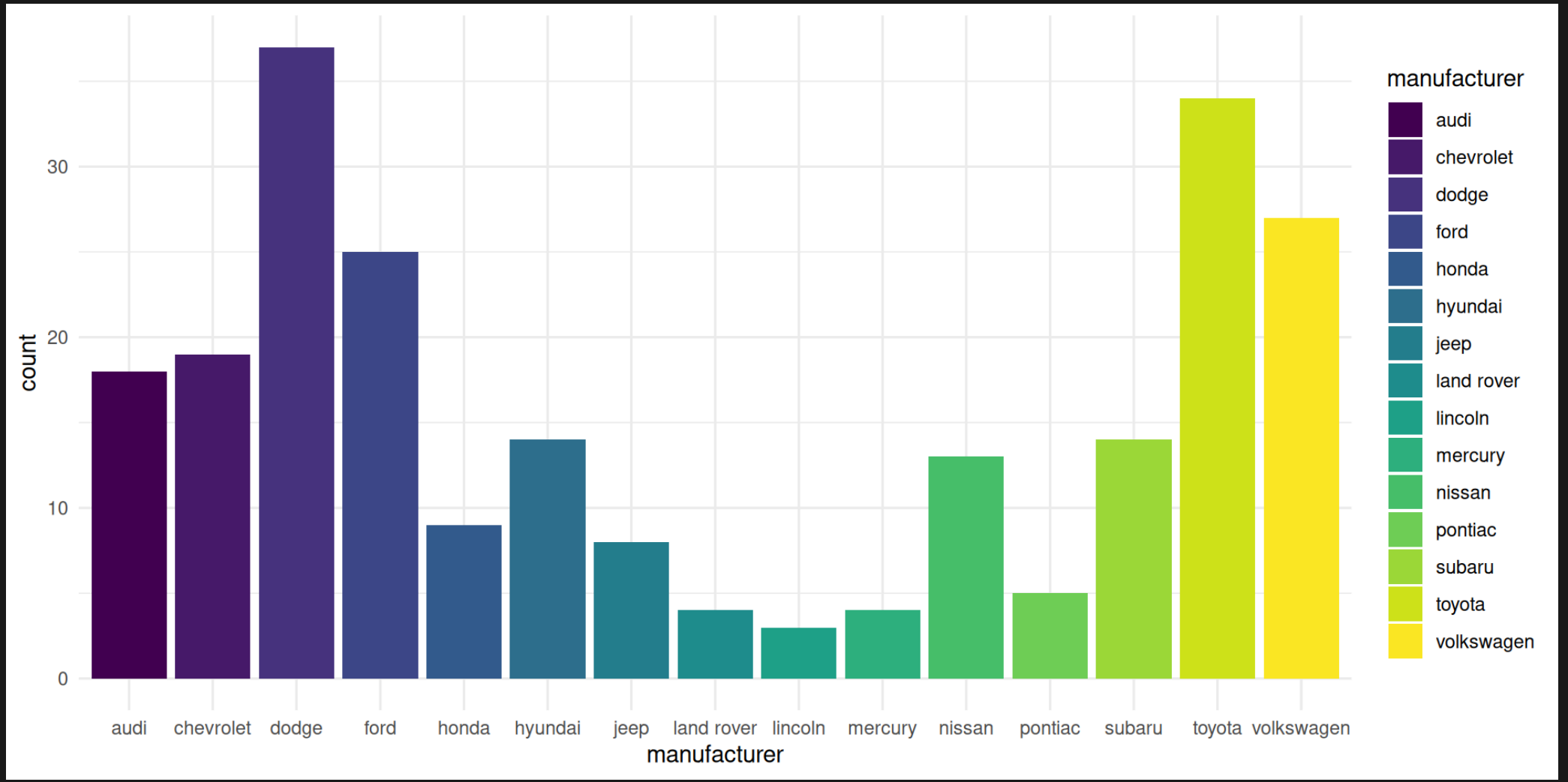
> in barplots, the relevant variable is the fill

```
1  prod <- ggplot(mpg, aes(x = manufacturer, fill = manufacturer)) + geom_bar(
2  prod
```



# Controlling barplots: fill

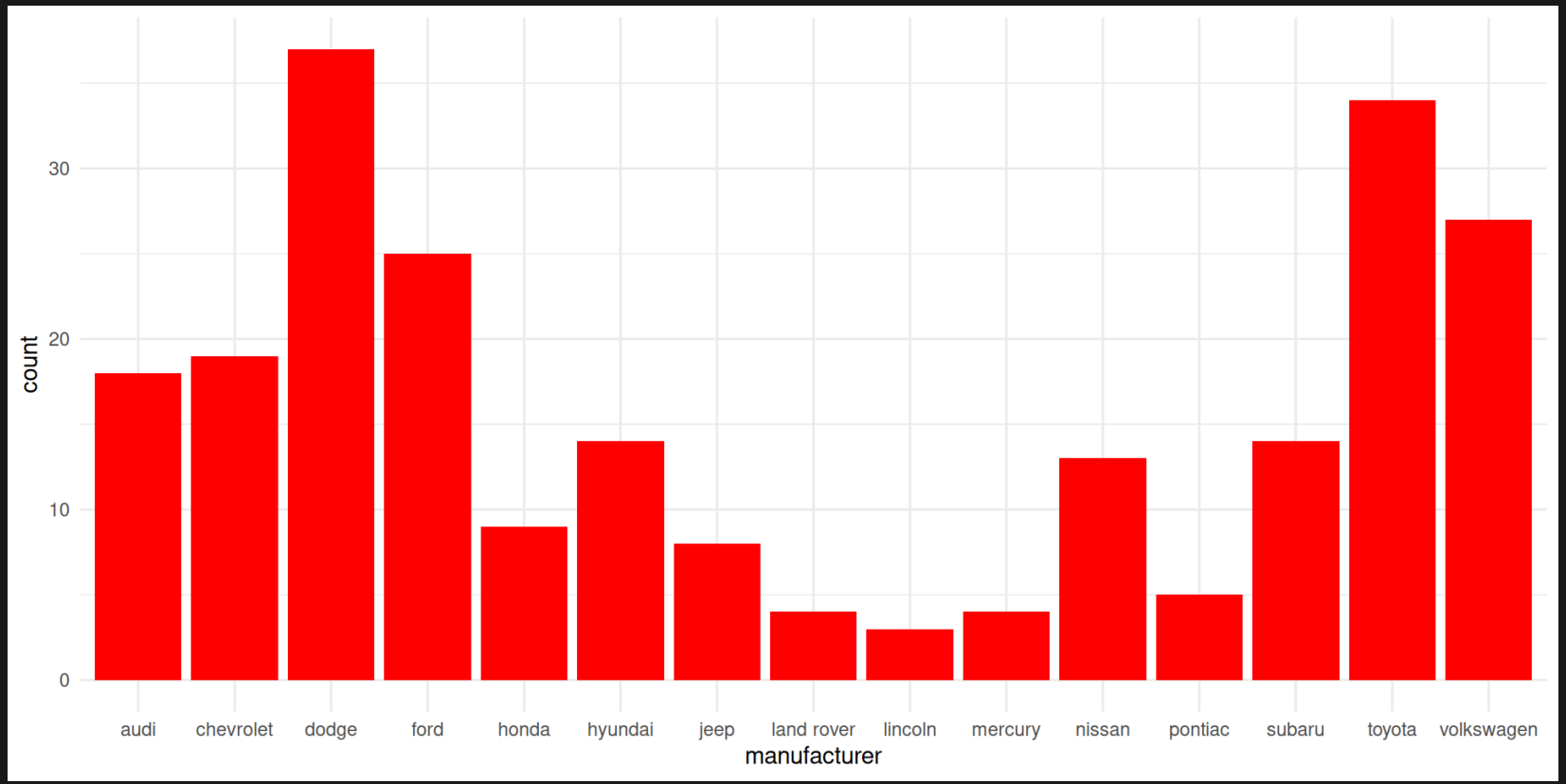
```
1  prod + scale_fill_viridis_d()
```



# Not-mapped appearance

> if appearance **does not vary** then pass it directly:

```
1  ggplot(mpg, aes(x = manufacturer)) + geom_bar(fill = "red")
```



# 3 · Theme

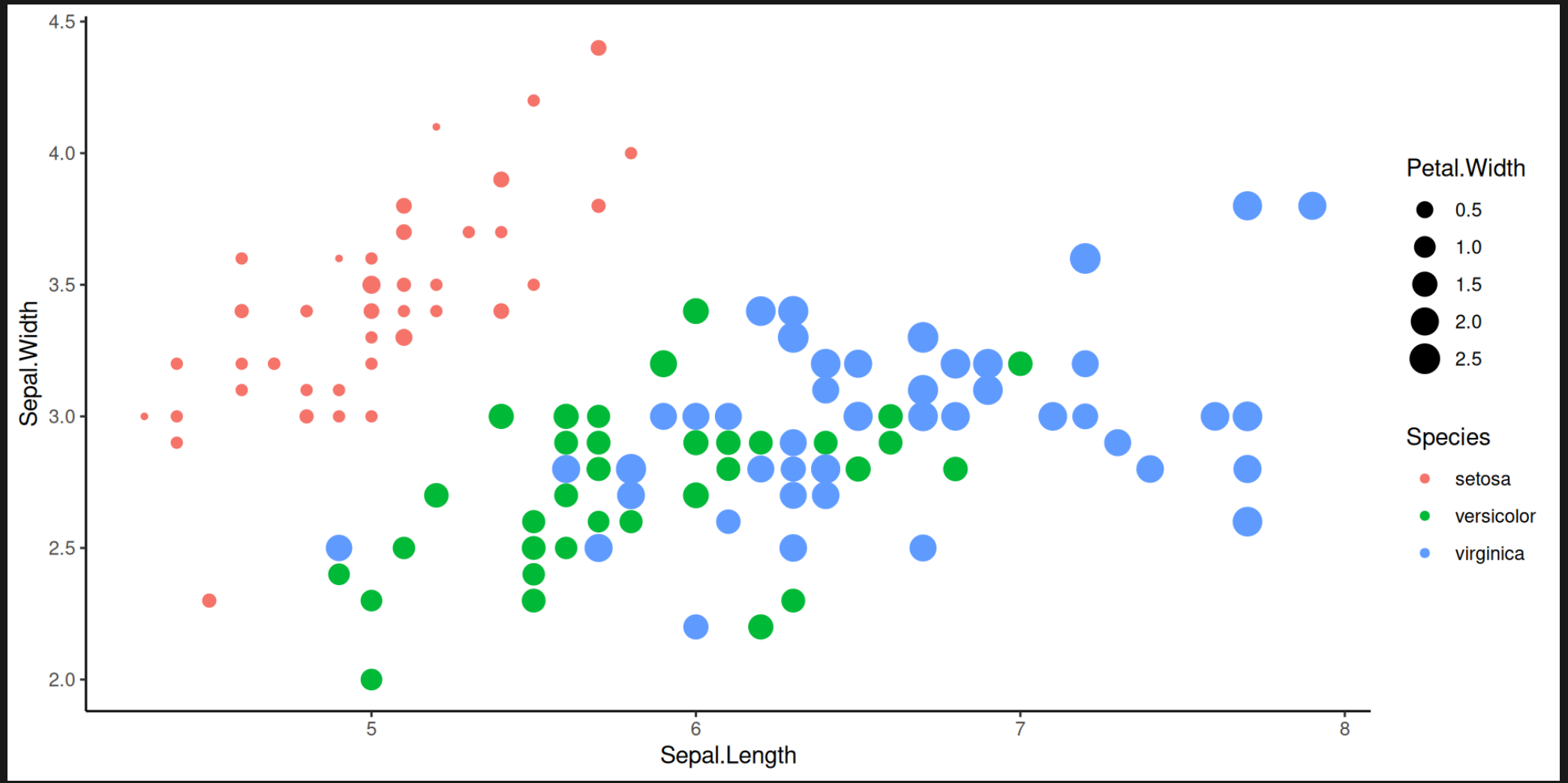
# Theming the plot

> `theme` functions refer to the appearance of **everything** but the mapping

- pre-installed themes (e.g. `theme_minimal()`)
- extra themes (e.g. `ggthemes`, `hrbrthemes`)
- fine_tuning the themes: `theme()`
- changing the labels: `labs()`

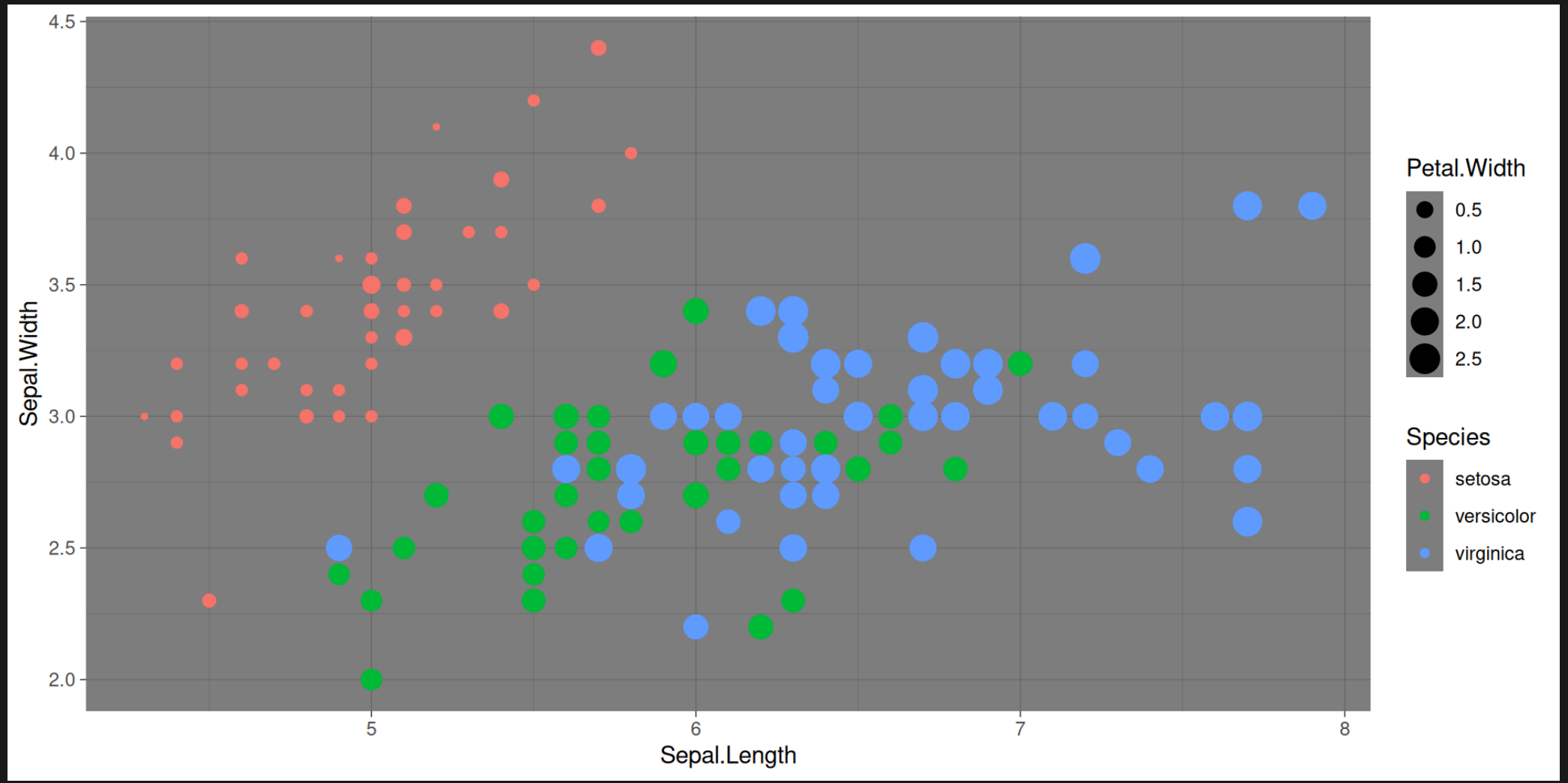
# pre-installed themes gallery

```
1  base + theme_classic()
```



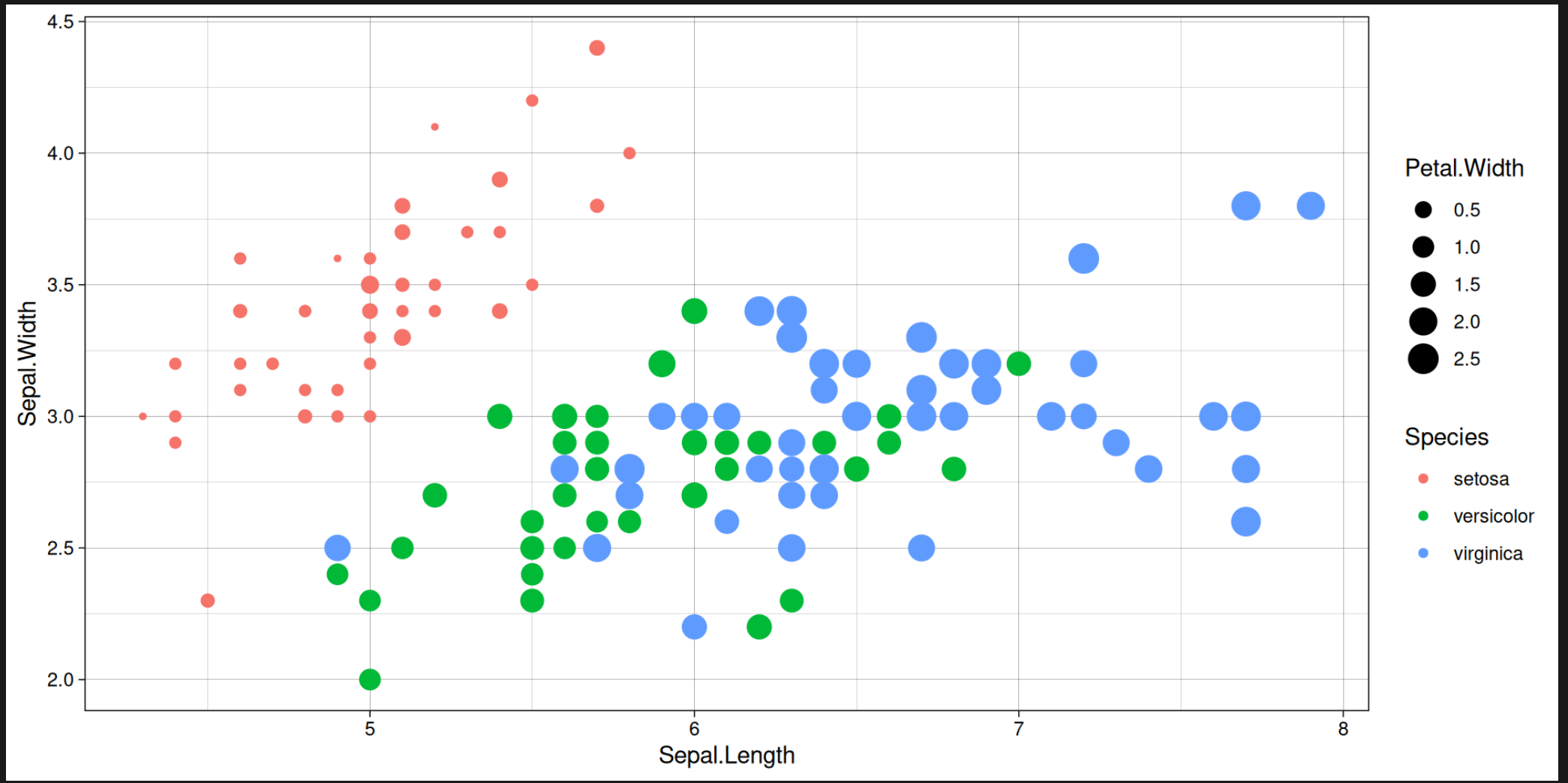
# pre-installed themes gallery

```
1  base + theme_dark()
```



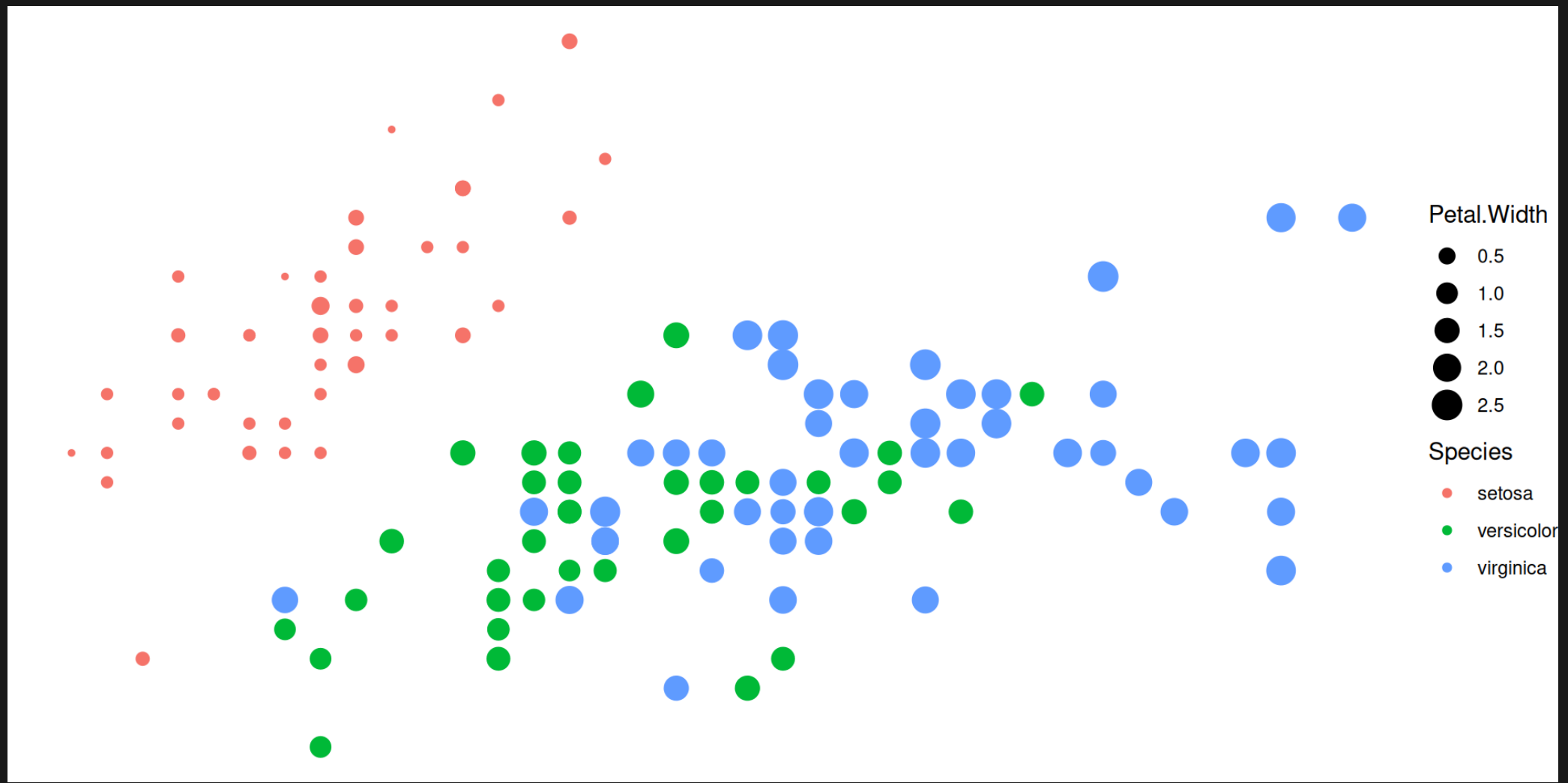
# pre-installed themes gallery

```
1  base + theme_linedraw()
```



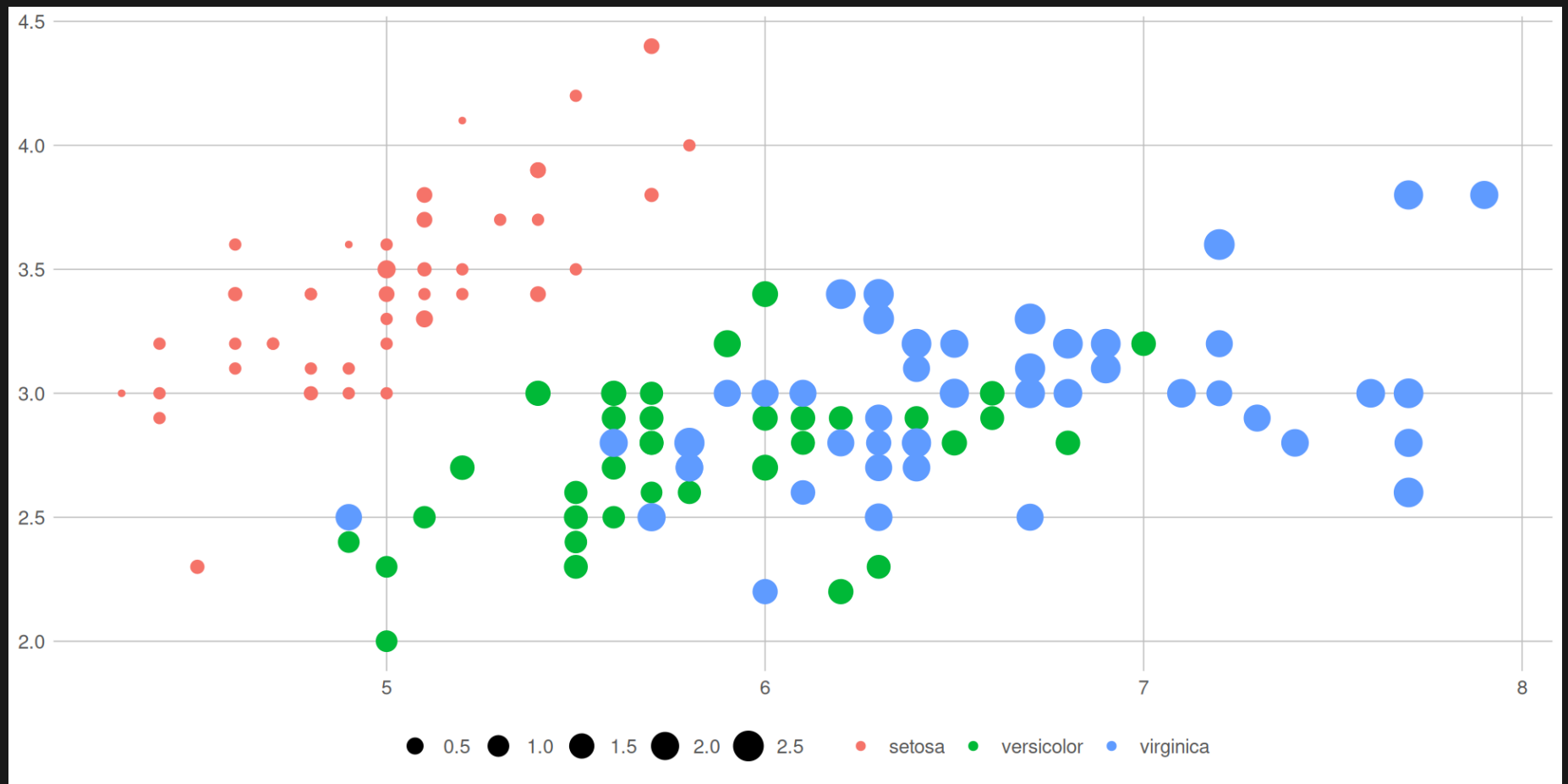
# pre-installed themes gallery

```
1  base + theme_void()
```



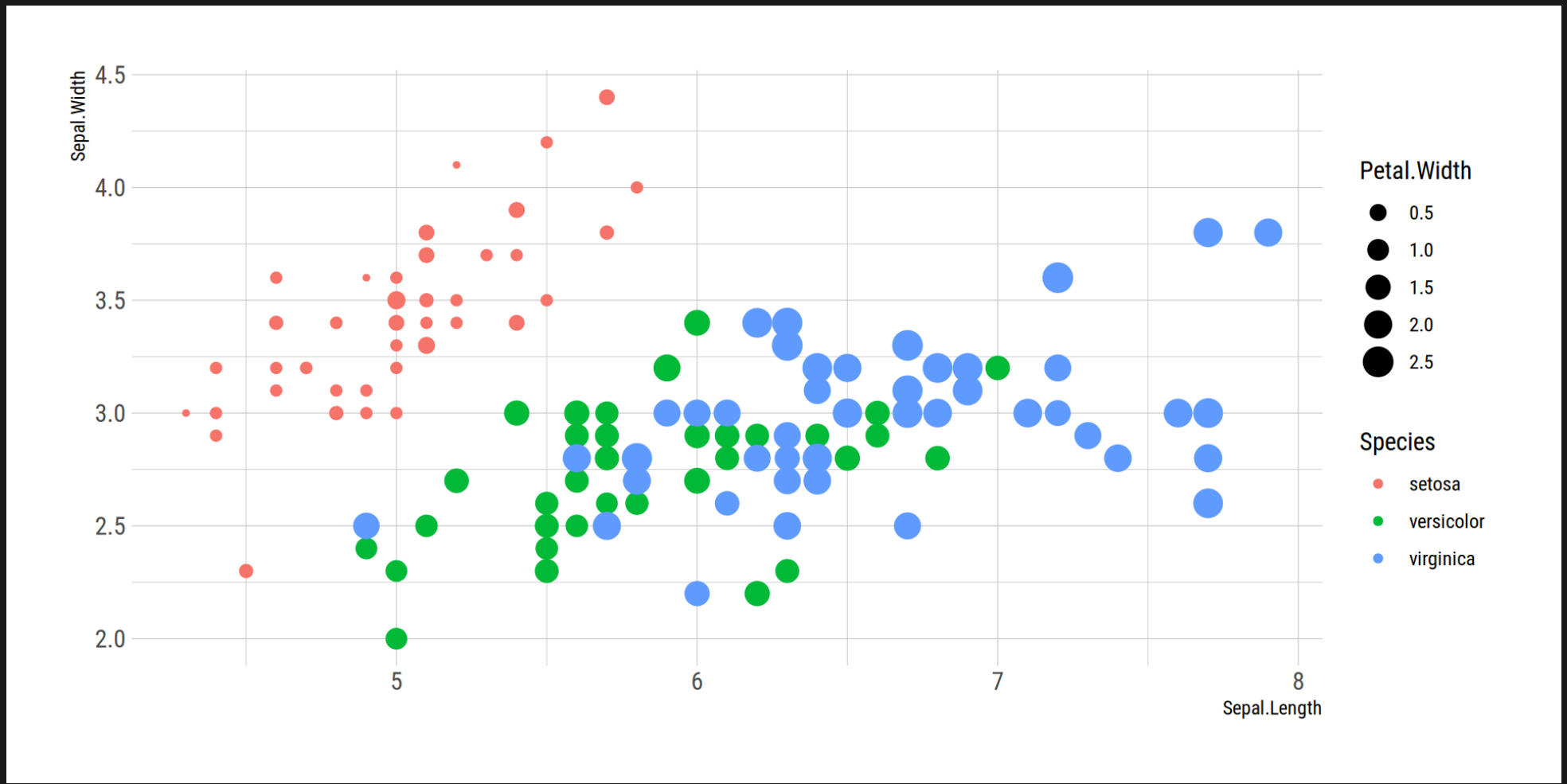
# install new themes and use theme

```
1 install.packages("ggthemes")
2 library(ggthemes)
3 base + theme_excel_new()
```



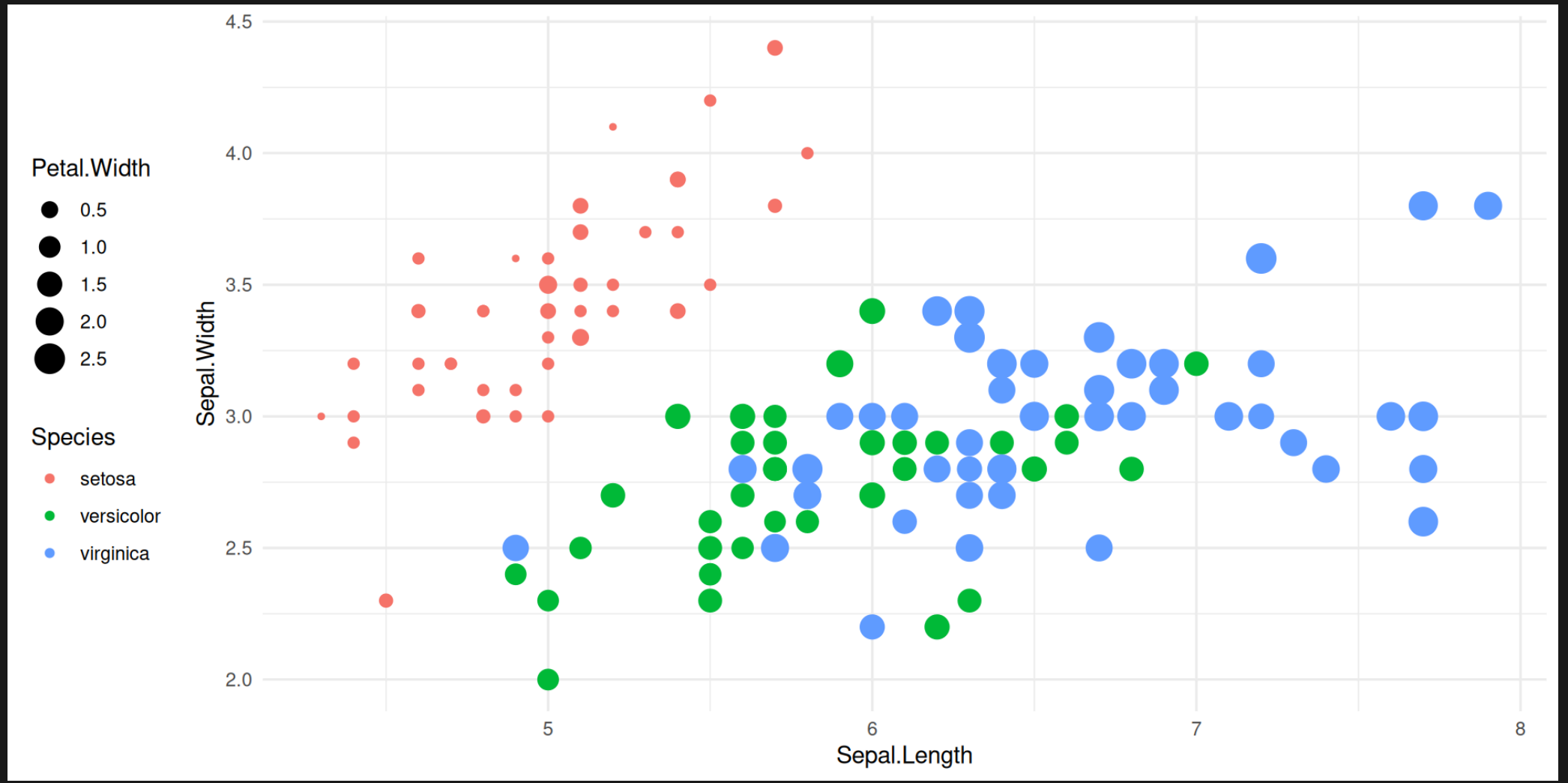
# install new themes and use them

```
1  library(hrbrthemes)
2  base + theme_ipsum_rc()
```



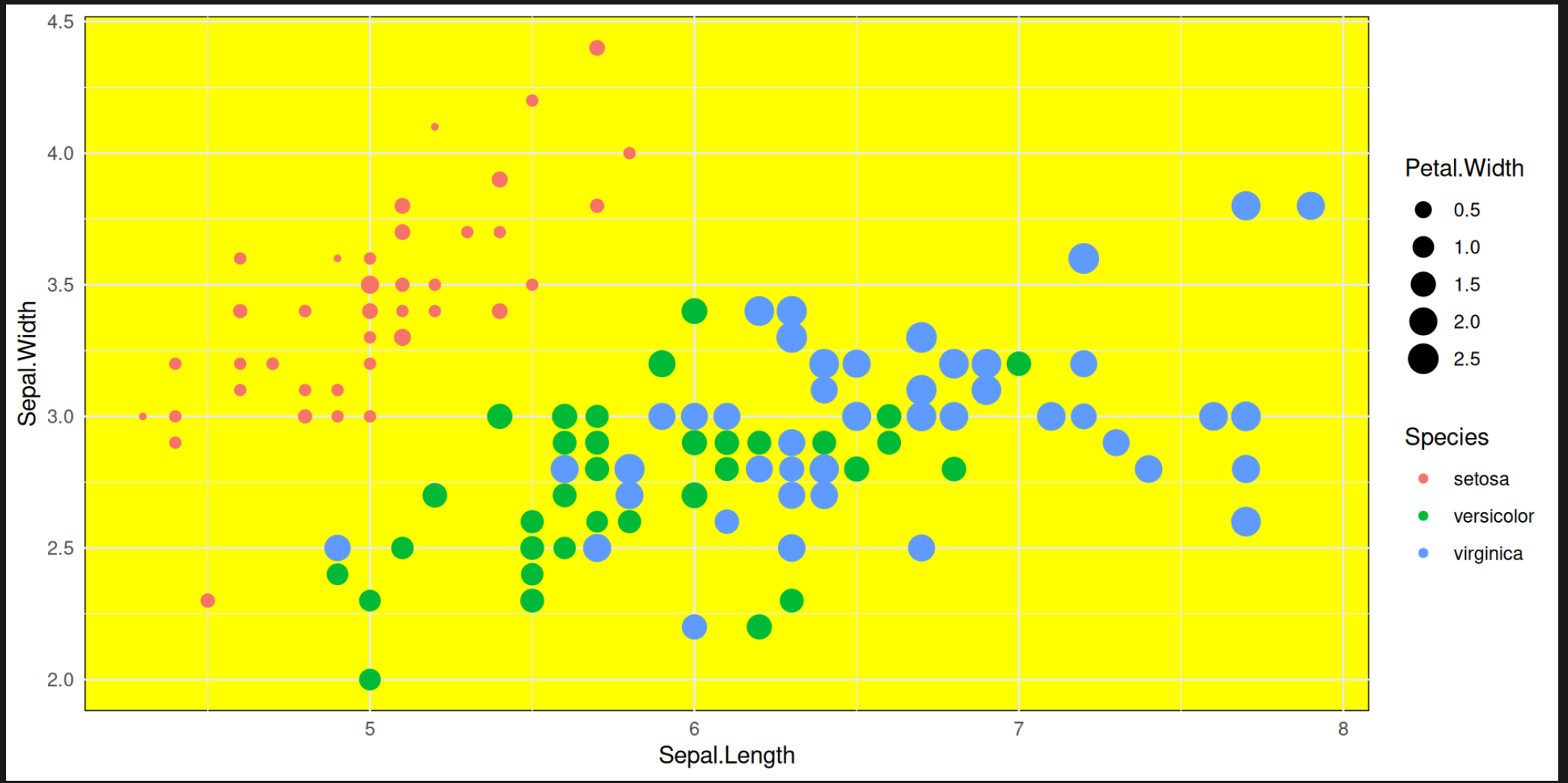
# fine-tuning themes: `theme()`

```
1  base + theme(legend.position = "left")
```



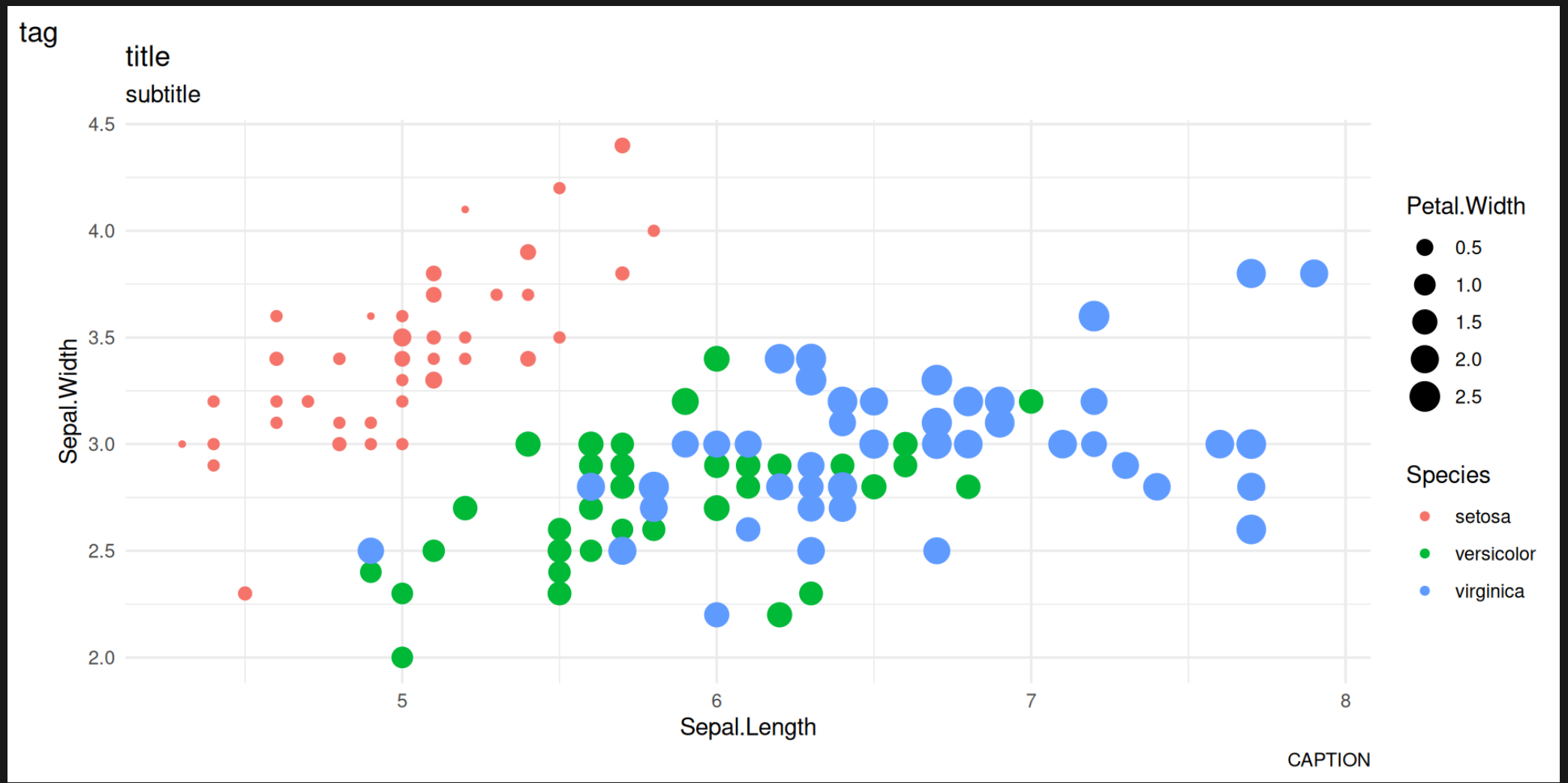
# fine-tuning themes: `theme()`

```
1  base + theme(panel.background = element_rect(fill = "yellow"))
```



# adding titles

```
1  base + labs(title = "title", subtitle = "subtitle", caption = "CAPTION", ta
```



# saving a plot

> to save a plot to file, use `ggsave()`. By default it saves the last plot made

```
ggsave(filename, width, height, dpi)
```

# 4 · Annotations & lines

# adding non-data driven elements

> add elements that are *not* in the data

- reference lines or areas
- text
- annotations

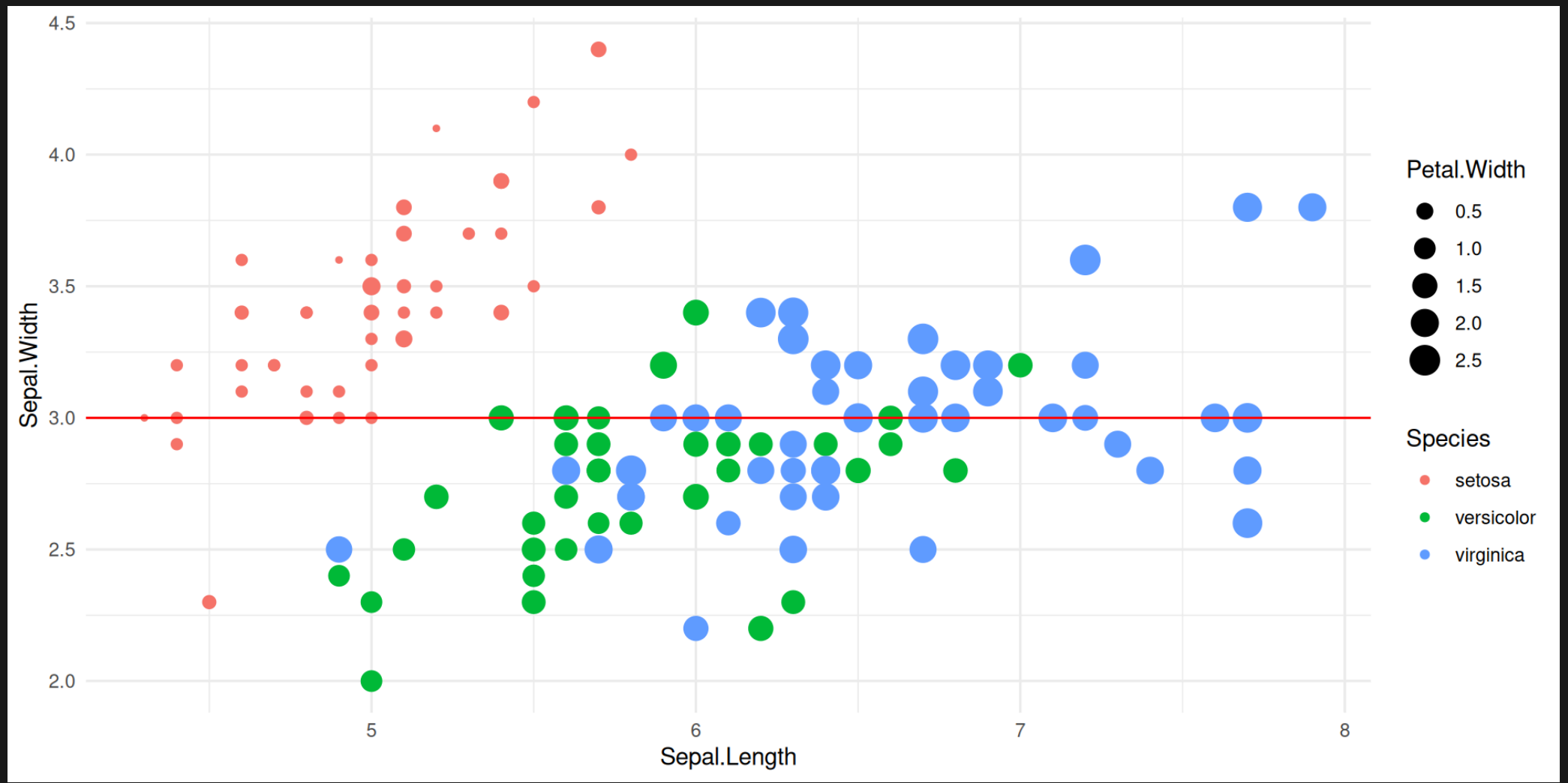
# lines

> there are three types of lines:

- horizontal: `geom_hline()`
- vertical: `geom_vline()`
- $y = ax + b$ : `geom_abline()`

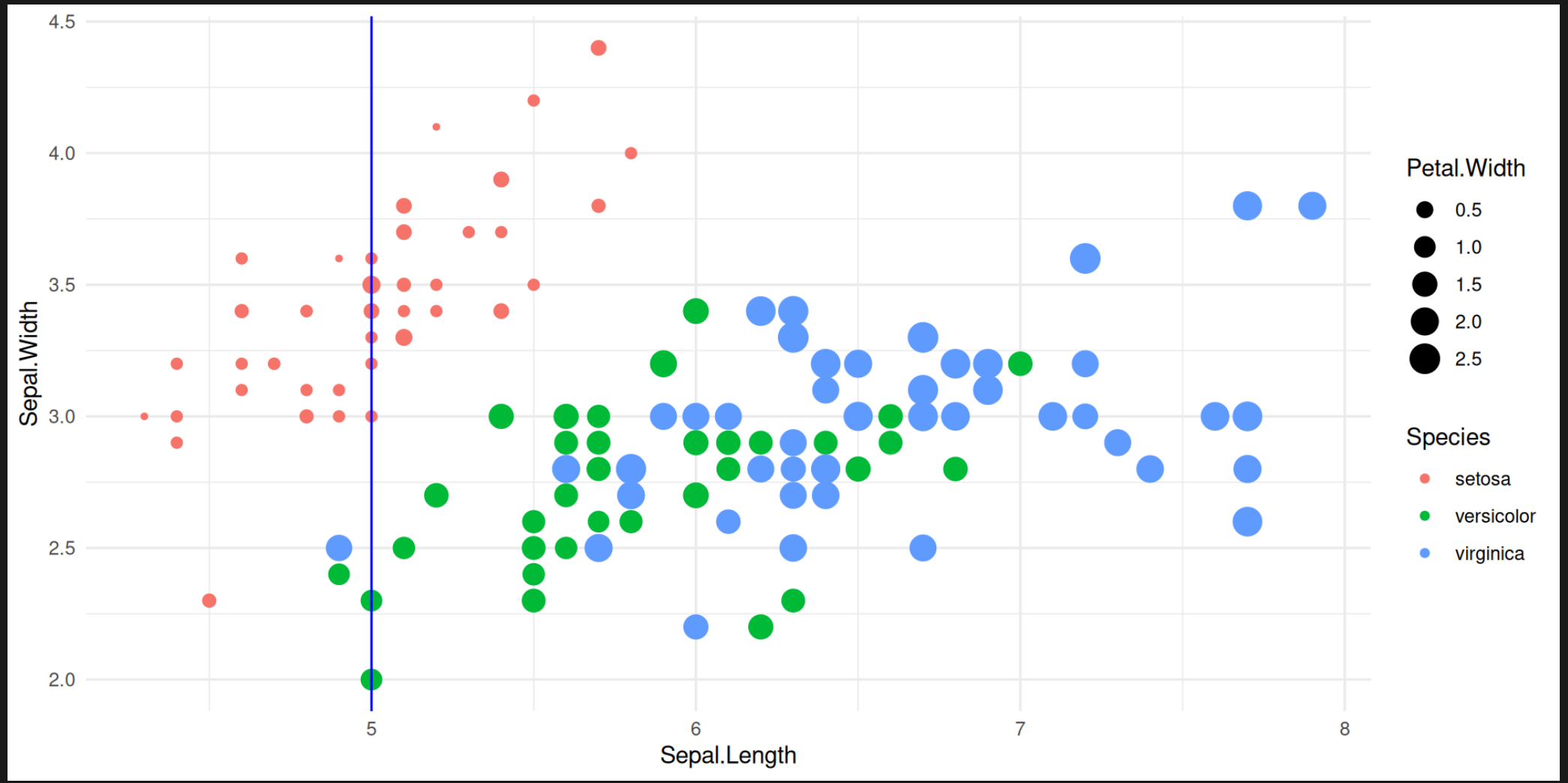
# lines

```
1  base + geom_hline(yintercept = 3, color = "red")
```



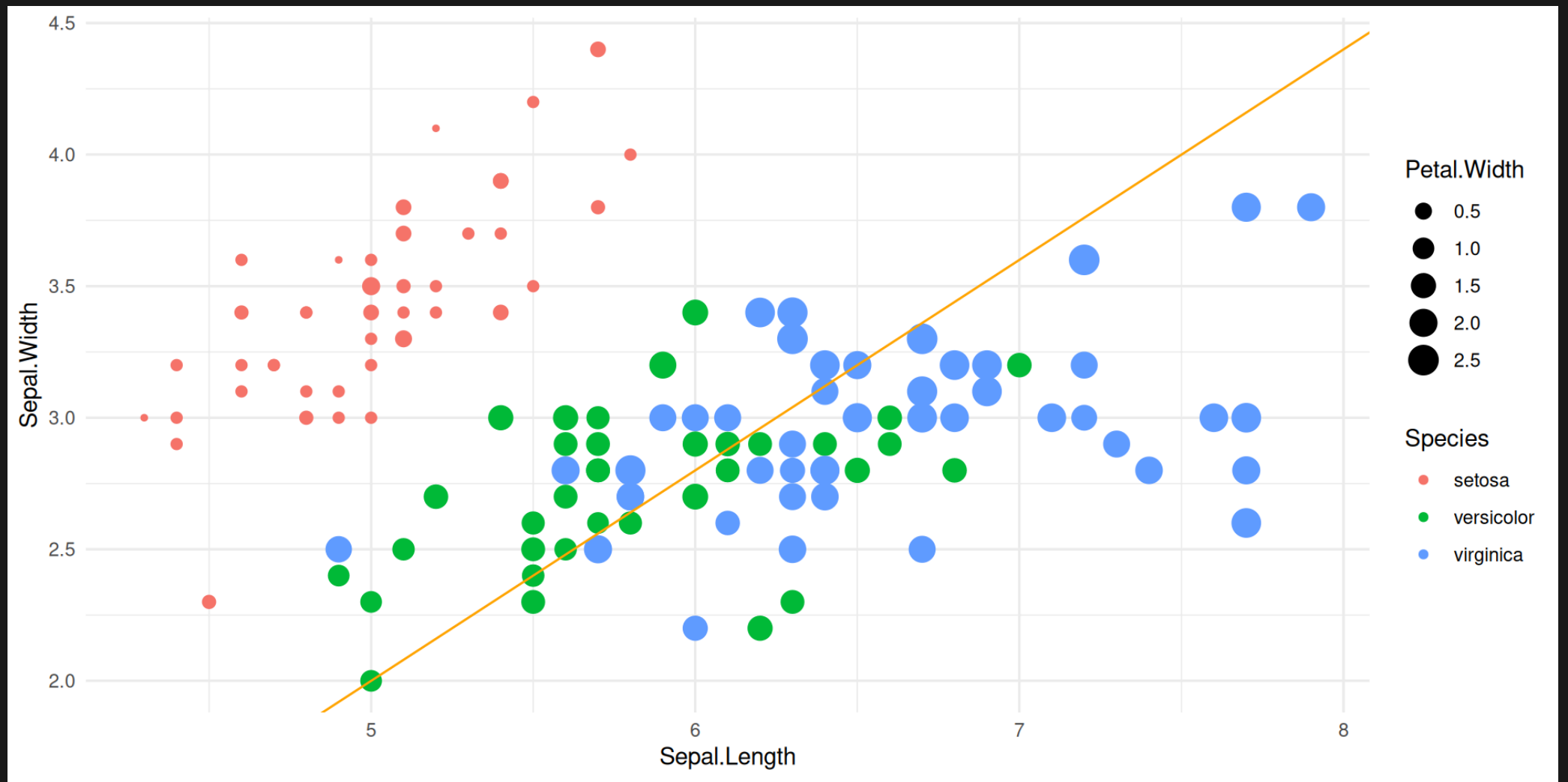
# lines

```
1  base + geom_vline(xintercept = 5, color = "blue")
```



# lines

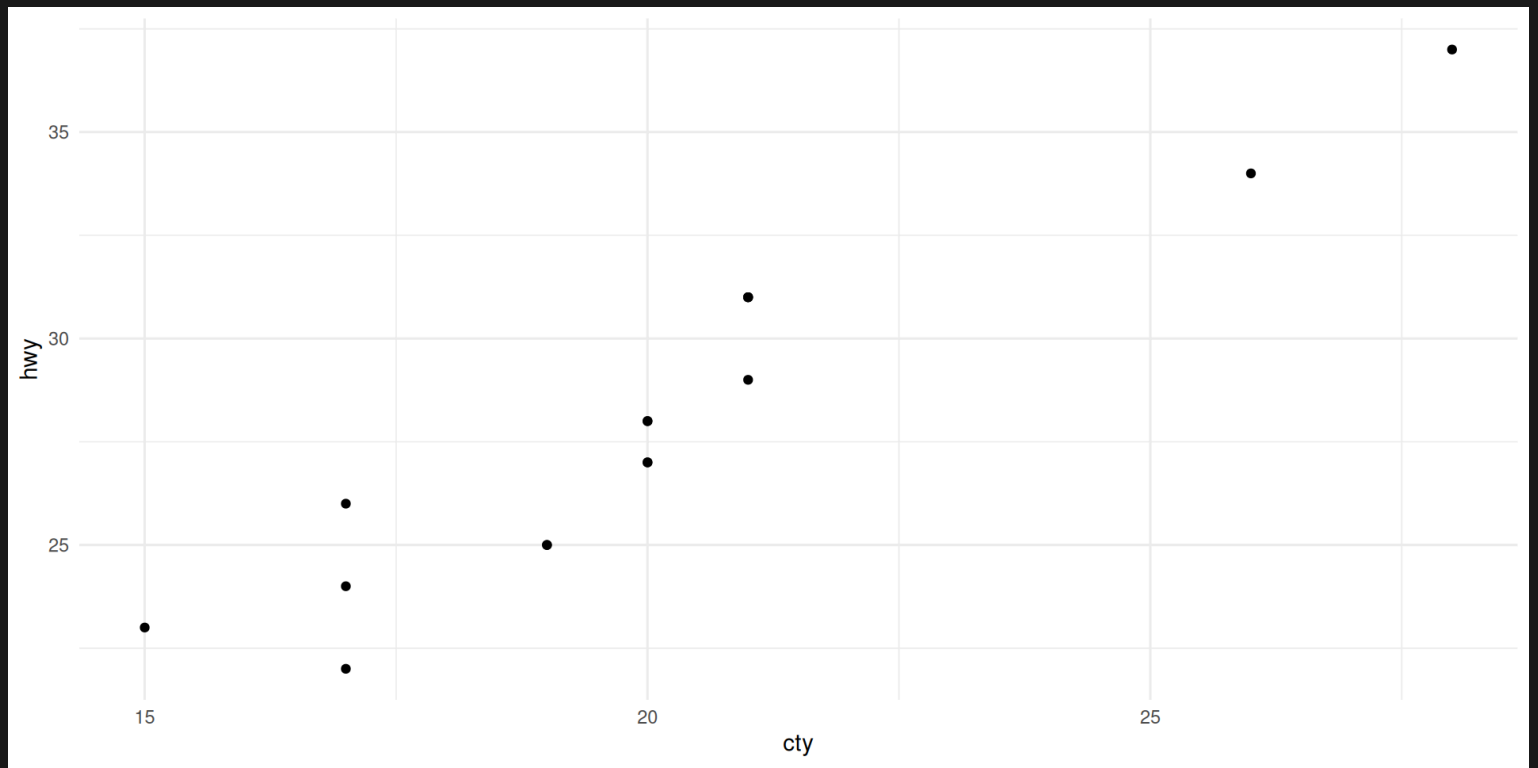
```
1  base + geom_abline(slope = 0.8, intercept = -2, color = "orange")
```



# Text as a `geom`

> there are two text geoms: `text` and `label`

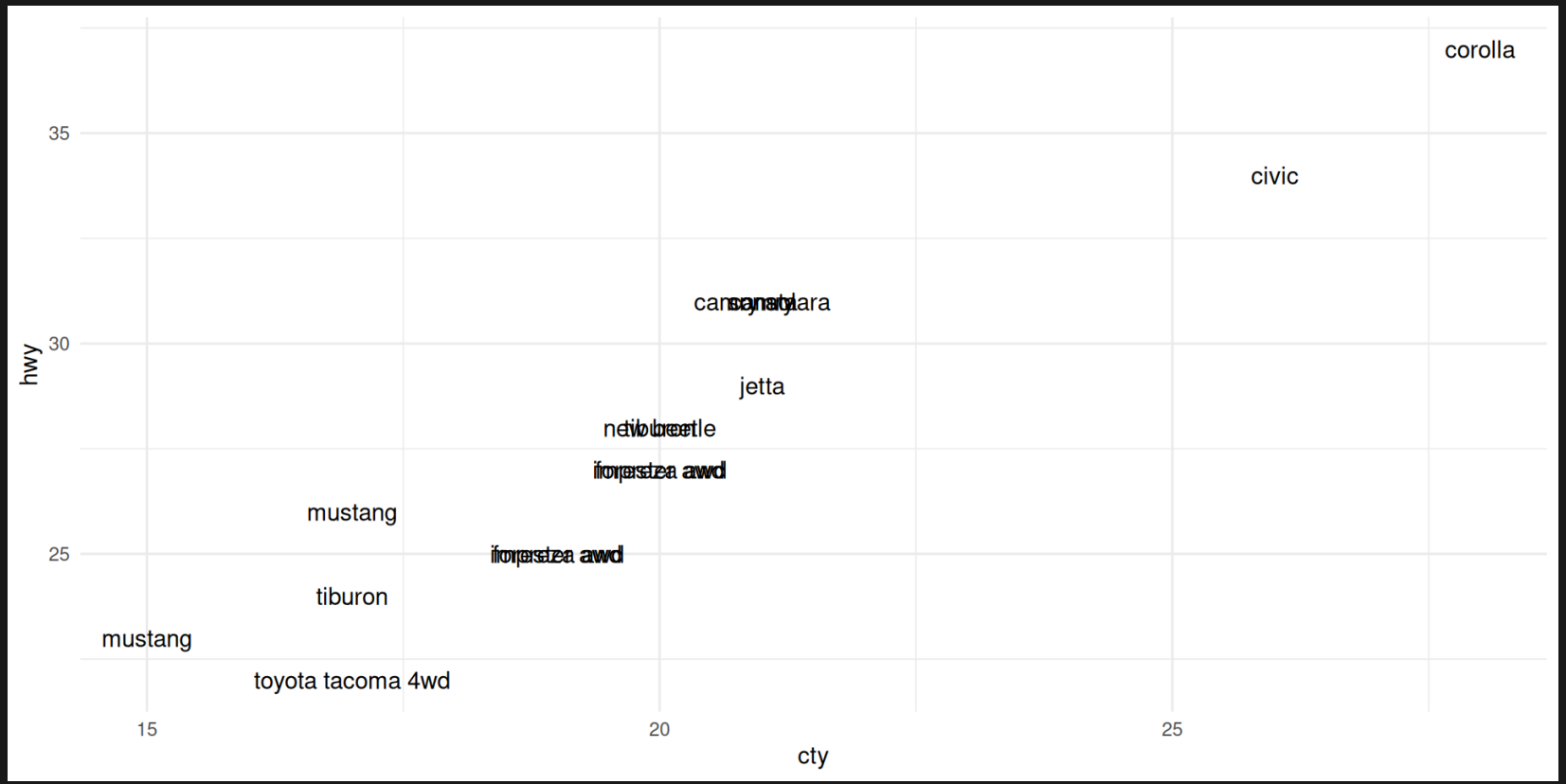
```
1  carspeed <- mpg %>%  filter(year == 2008 & trans == "manual(m5)") %>% ggplo
2  carspeed + geom_point()
```



# Text as a geom

> labeling cars by their name: text

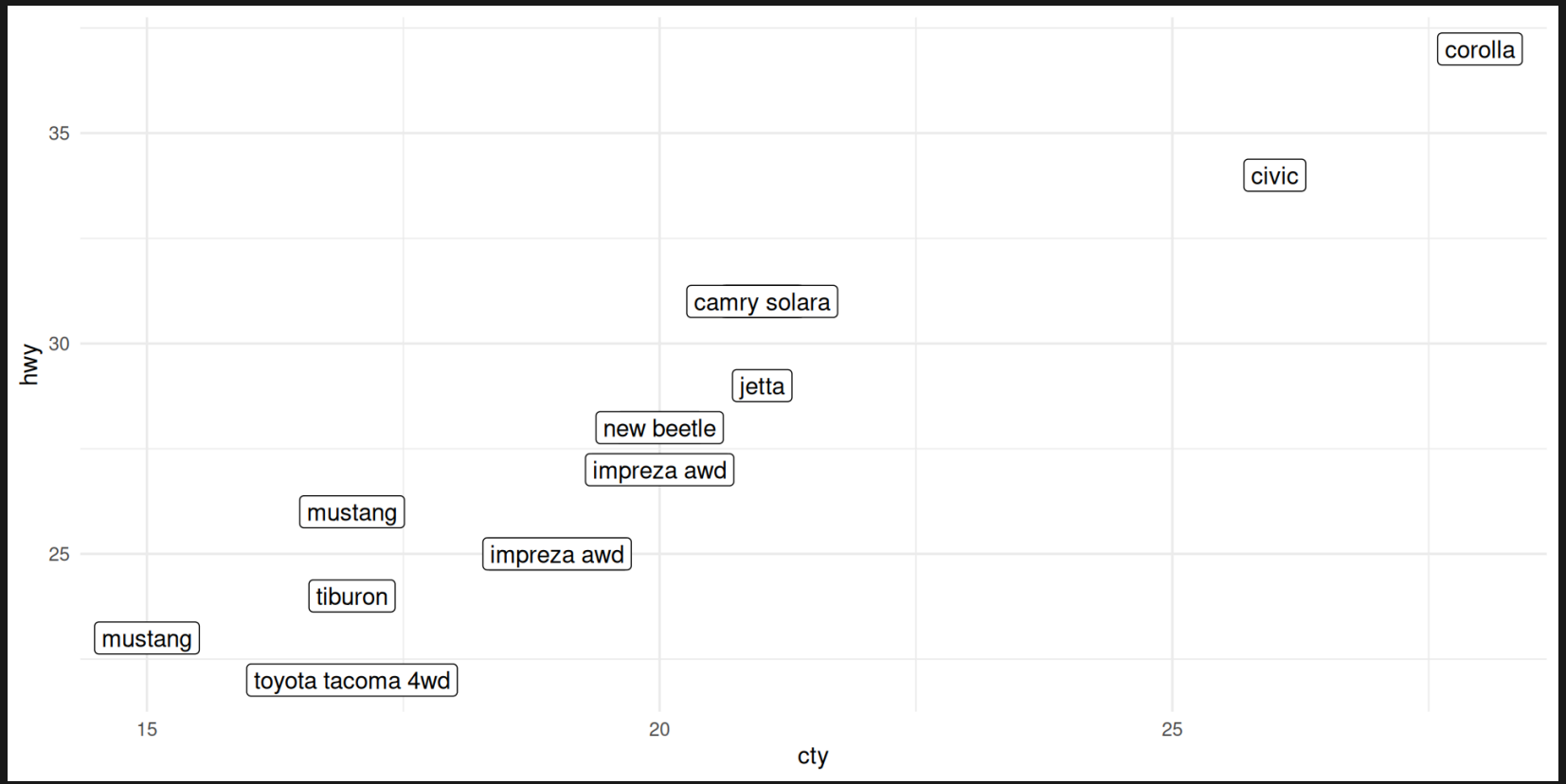
```
1  carspeed + geom_text(aes(label = model))
```



# Text as a geom

> labeling cars by their name: label

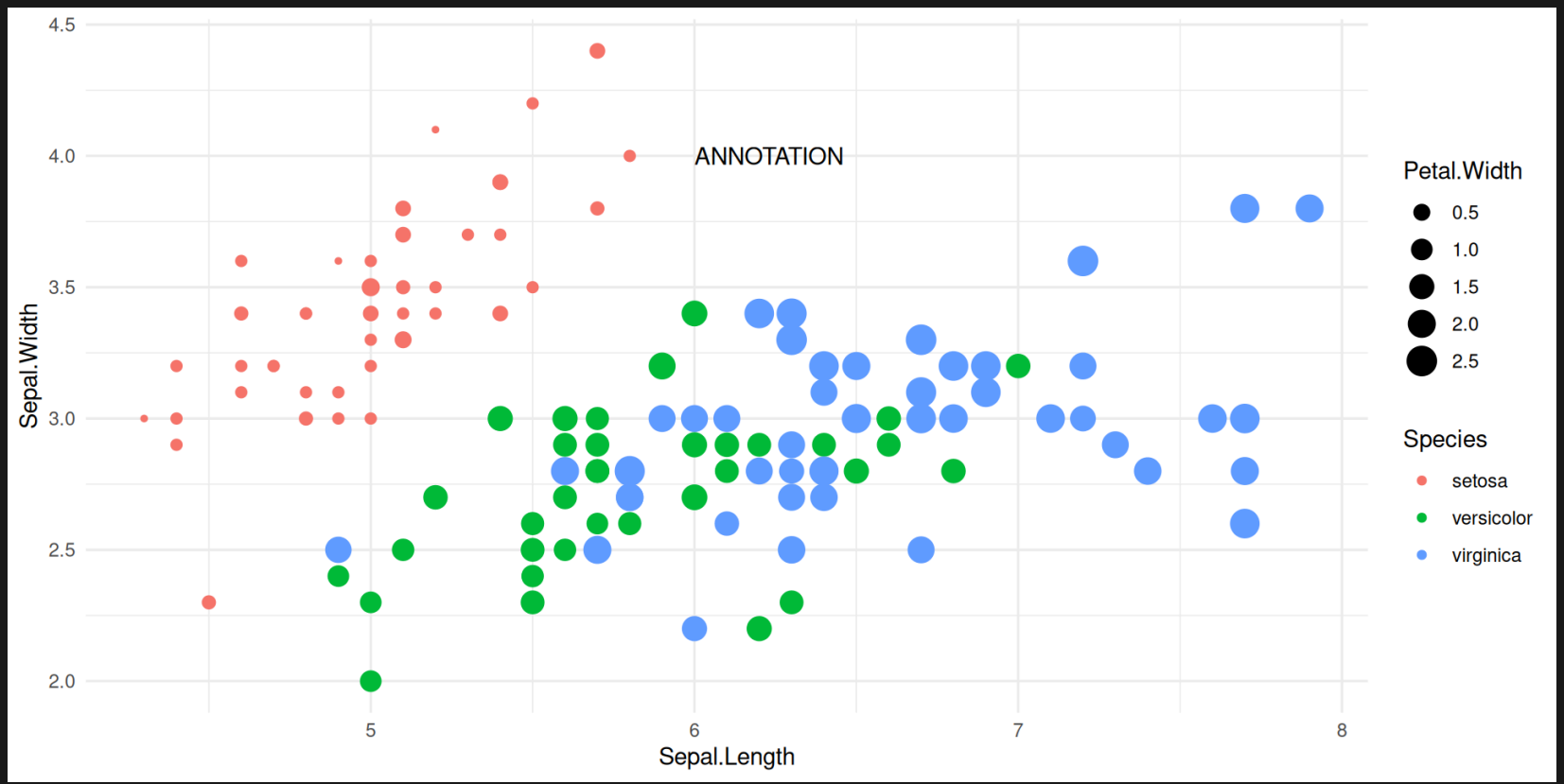
```
1  carspeed + geom_label(aes(label = model))
```



# Text as an annotation

> just put a custom text somewhere

```
1  base + annotate(geom = "text", label = "ANNOTATION", x = 6, y = 4, hjust =
```



# To know more...

# Additional resources

- the ggplot **cheatsheet** is your friend (Help -> cheatsheets)
- **stack overflow** helps out for trickier questions
- not feeling inspired? 50 cool visualisations here:

  http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html
- need a complete tour of possibilities? https://www.r-graph-gallery.com/