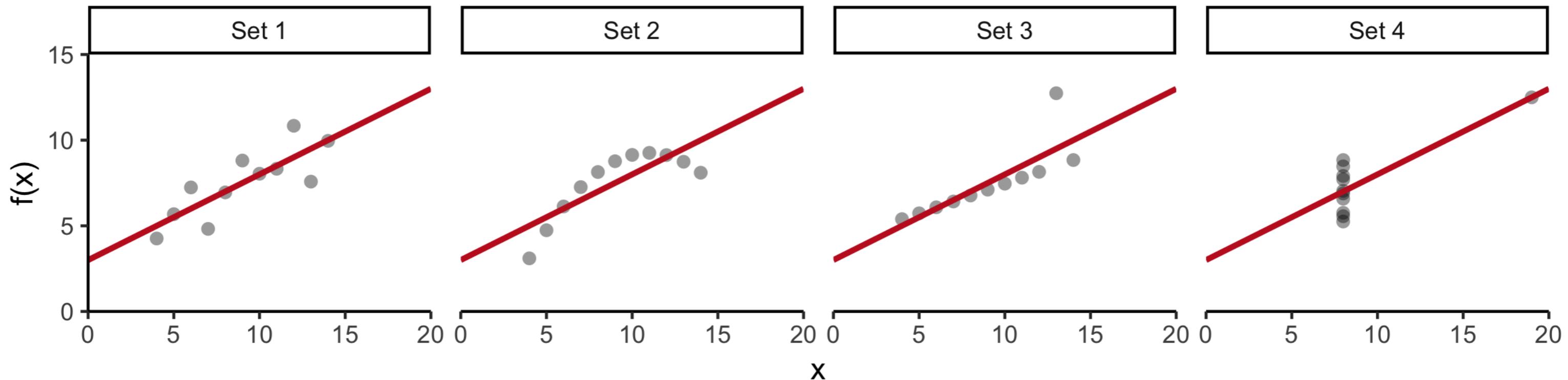


# Anscombe's plots

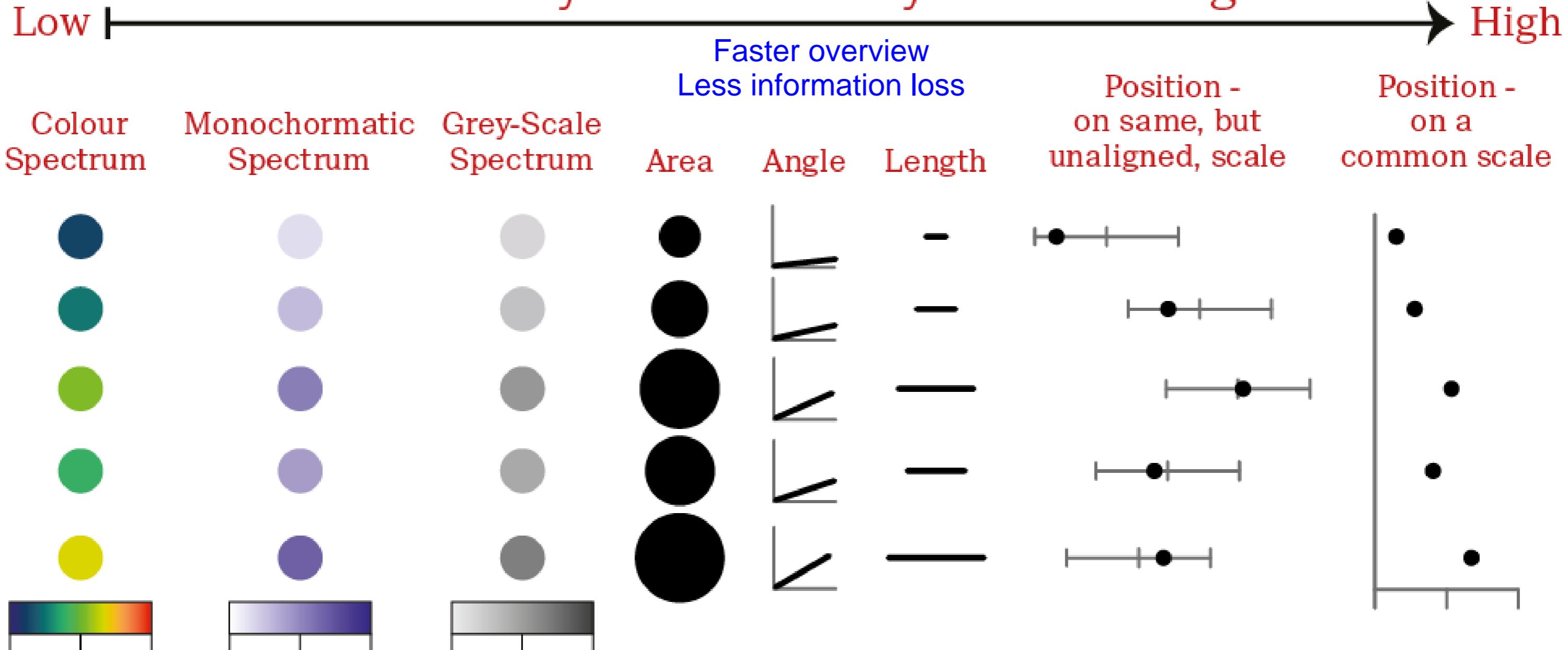


# Typical visible aesthetics

Aesthetic	Description
x	X axis position
y	Y axis position
fill	Fill color
color	Color of points, outlines of other geoms
size	Area or radius of points, thickness of lines

Aesthetic	Description
alpha	Transparency
linetype	line dash pattern
labels	Text on a plot or axes
shape	Shape

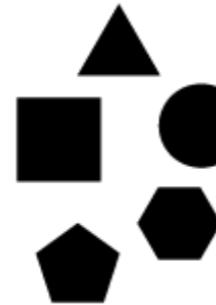
# Efficiency and Accuracy of Decoding



## Efficiency in Decoding Separate Groups

Low → High

Filled Shapes



Sequential Colours



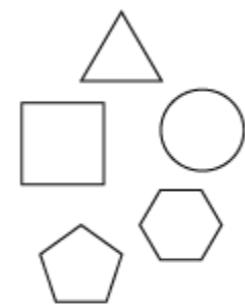
Qualitative Colours



Hatching



Shape Outlines



Labels

ANT1  
FRG2  
FRG1  
Gapdh  
DUX4

Line Width



Line Type



Line Colours



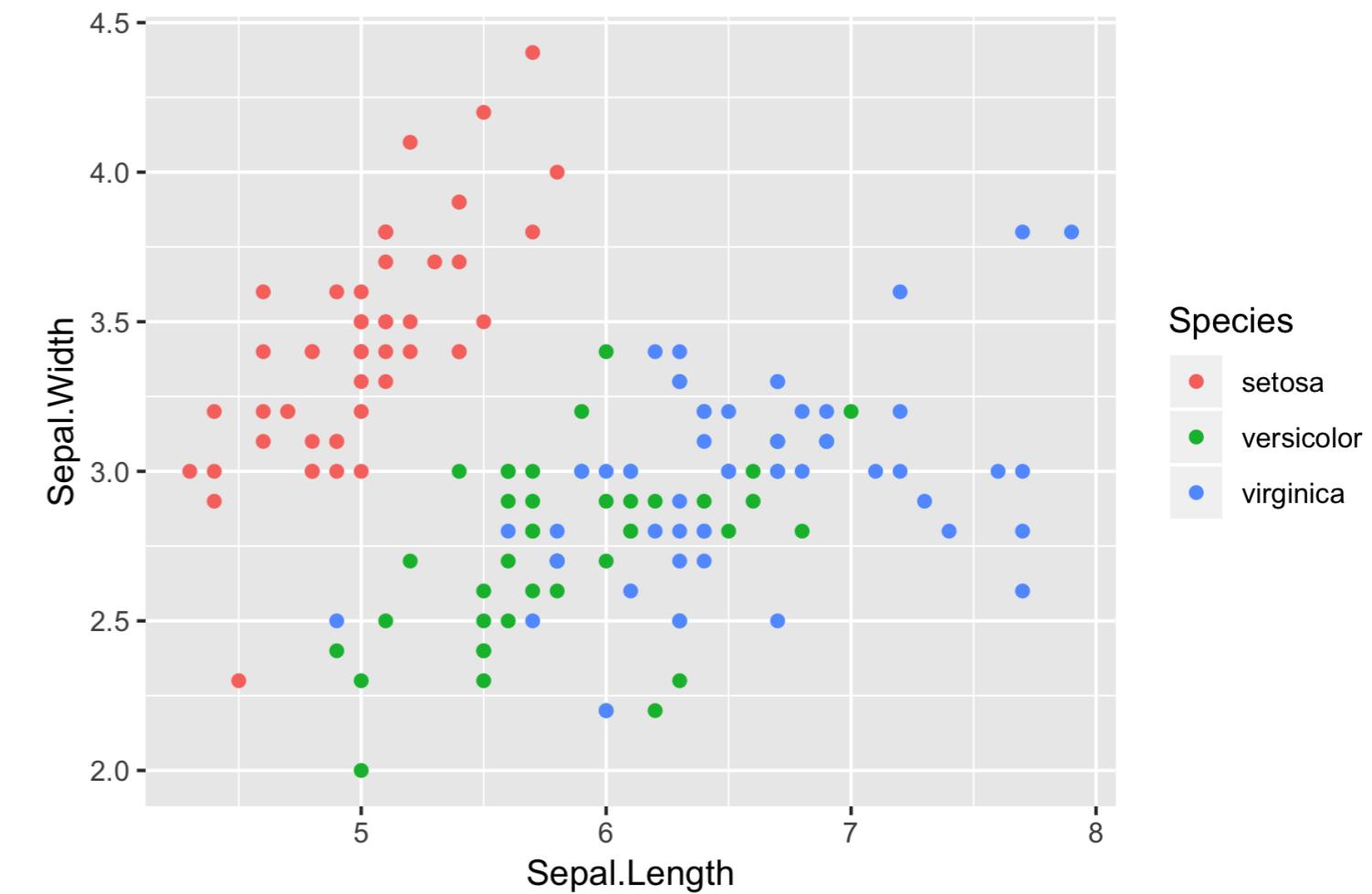
# Positions

Adjustment for overlapping

- identity
- dodge
- stack
- fill
- jitter
- jitterdodge
- nudge

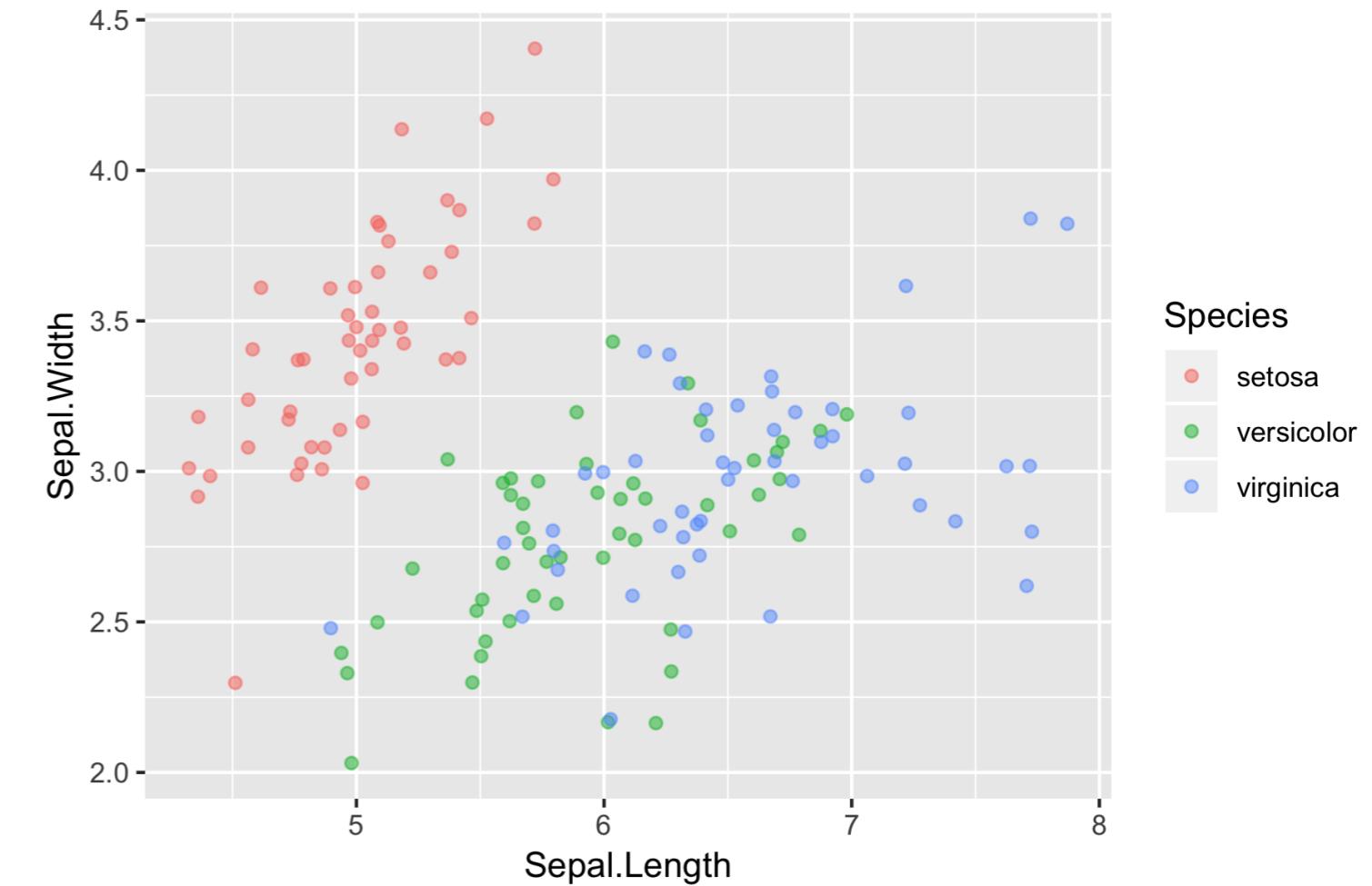
# position = "identity" (default)

```
ggplot(iris, aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 color = Species)) +  
  geom_point(position = "identity")
```



# position = "jitter"

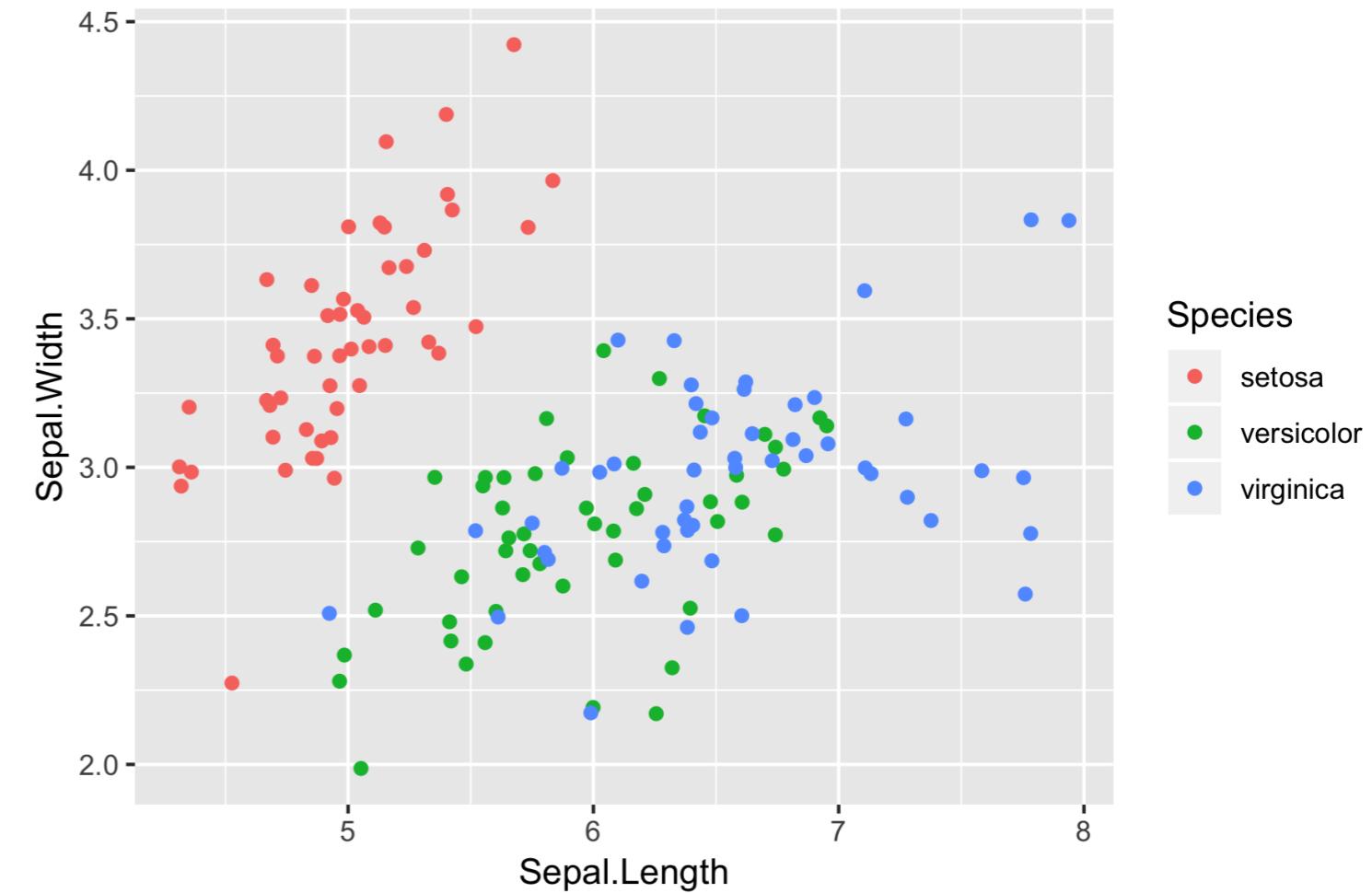
```
ggplot(iris, aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 color = Species)) +  
  geom_point(position = "jitter")
```



# position\_jitter()

```
posn_j <- position_jitter(0.1,  
                           seed = 136)  
  
ggplot(iris, aes(x = Sepal.Length,  
                  y = Sepal.Width,  
                  color = Species)) +  
  geom_point(position = posn_j)
```

- Set arguments for the position
- Consistency across plots & layers

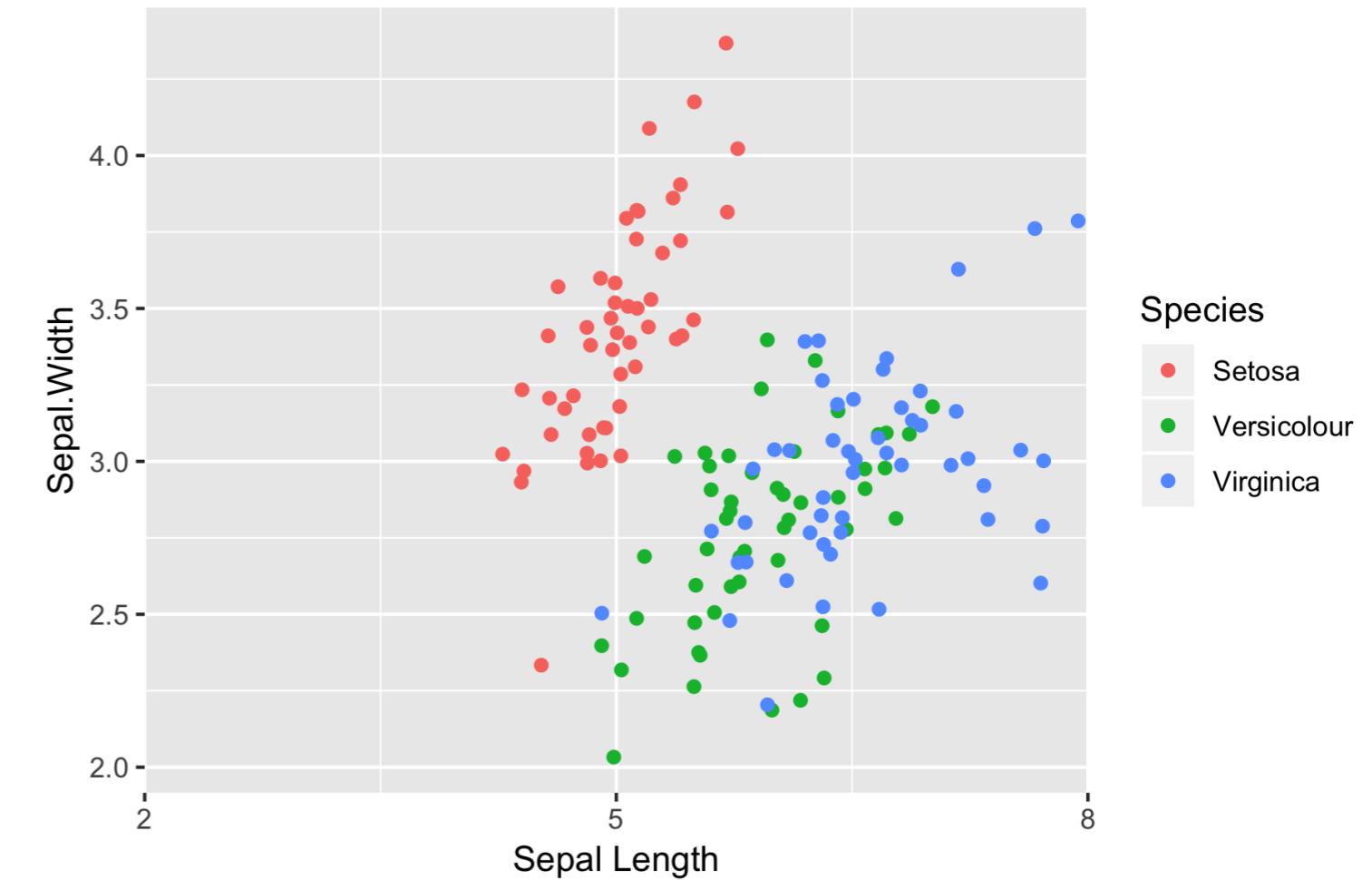


# Scale functions

- `scale_x_continuous()`
- `scale_y_*`(`)`
- `scale_color_discrete()`
  - Alternatively, `scale_colour_*`(`)`
- `scale_fill_*`(`)`
- `scale_shape_*`(`)`
- `scale_linetype_*`(`)`
- `scale_size_*`(`)`

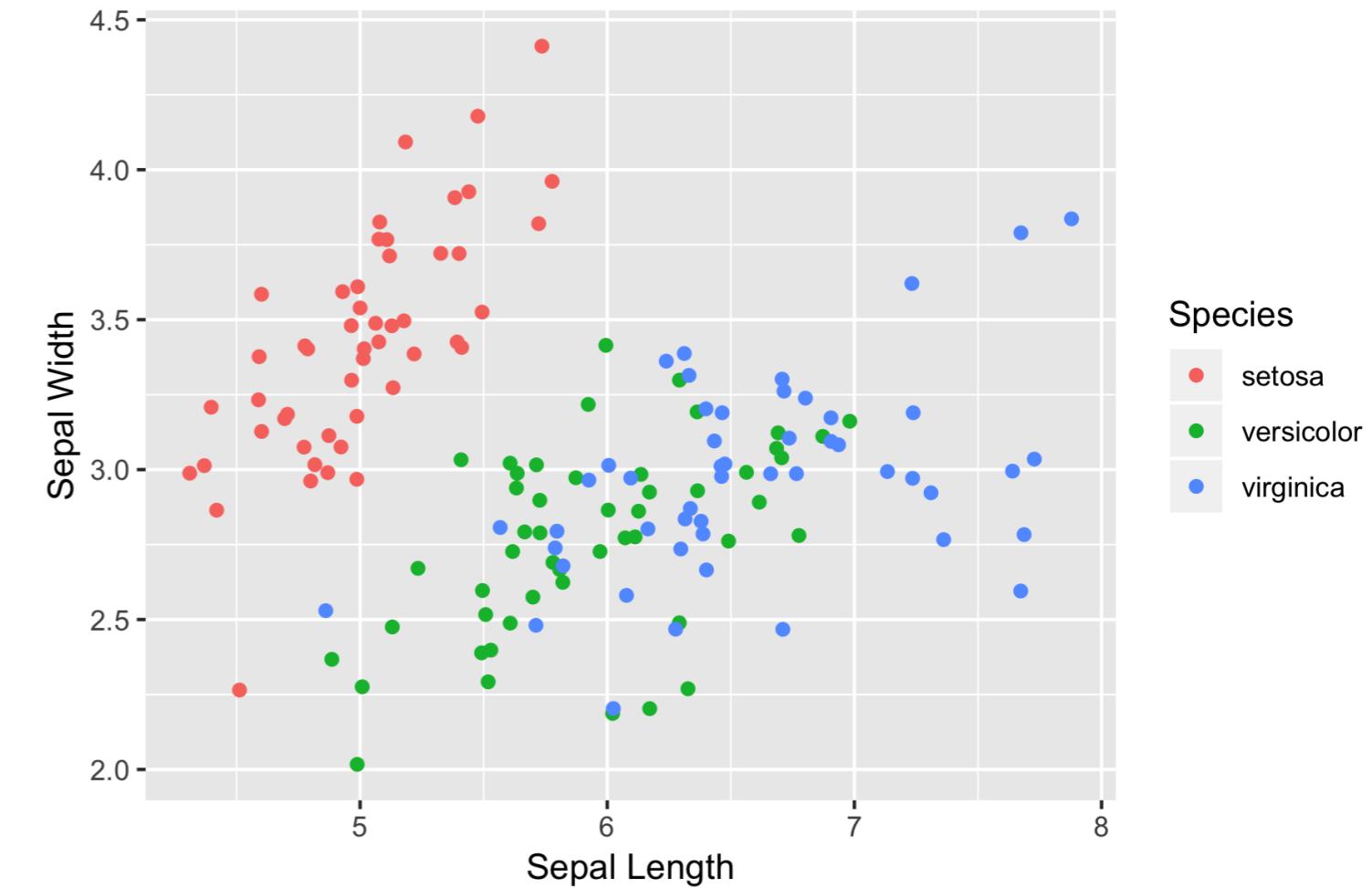
# The labels argument

```
ggplot(iris, aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 color = Species)) +  
  geom_point(position = "jitter") +  
  scale_x_continuous("Sepal Length",  
                     limits = c(2, 8),  
                     breaks = seq(2, 8, 3),  
                     expand = c(0, 0),  
                     labels = c("Setosa",  
                               "Versicolor",  
                               "Virginica")) +  
  scale_color_discrete("Species")
```

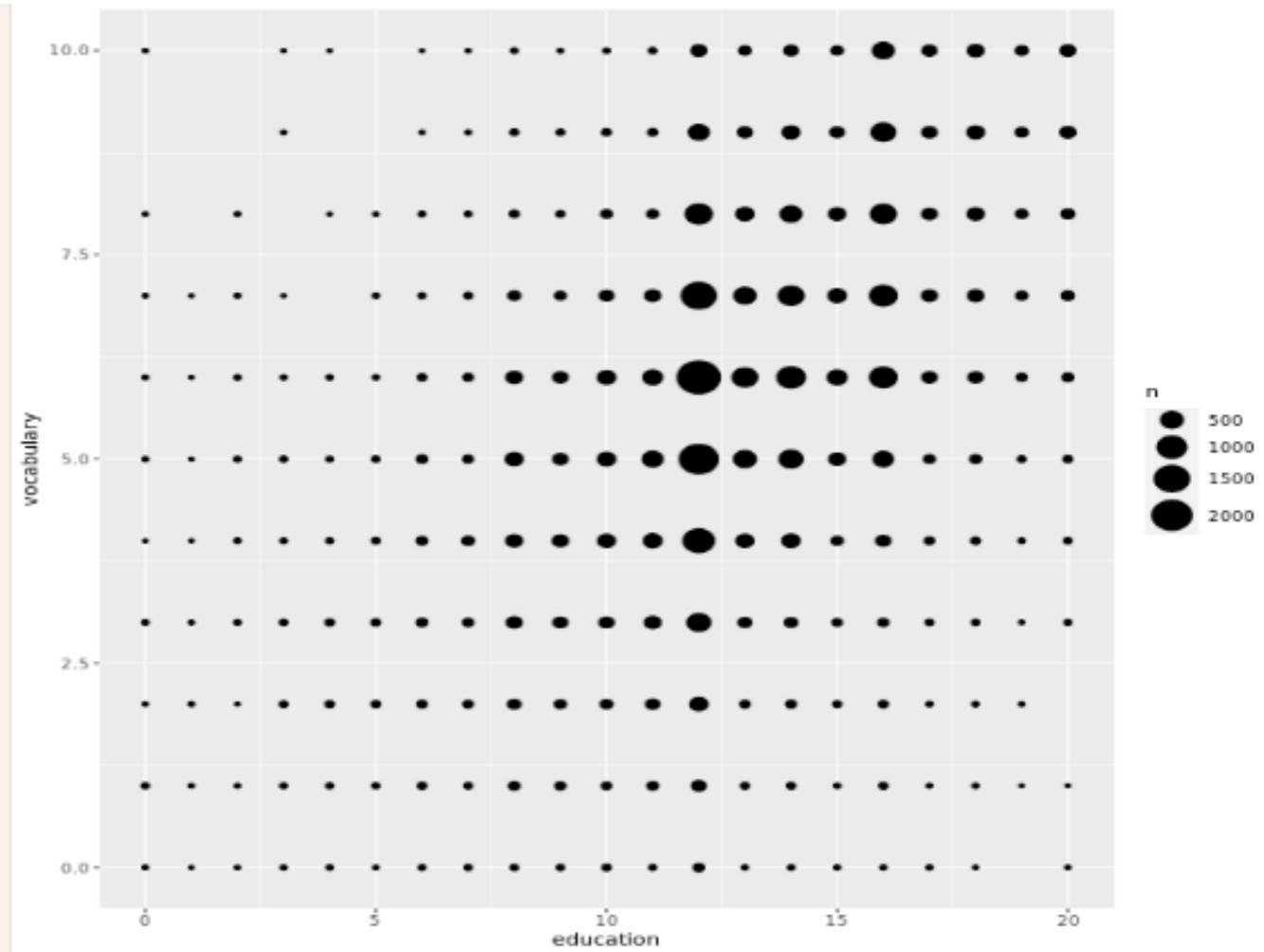


# labs()

```
ggplot(iris, aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 color = Species)) +  
  geom_point(position = "jitter") +  
  labs(x = "Sepal Length",  
       y = "Sepal Width",  
       color = "Species")
```



```
ggplot(Vocab, aes(x = education, y =  
vocabulary)) +  
  stat_sum() +  
  # Add a size scale, from 1 to 10  
  scale_size(range = c(1,10))
```



# 48 geometries

geom_*						
abline	contour	dotplot	jitter	pointrange	ribbon	spoke
area	count	errorbar	label	polygon	rug	step
bar	crossbar	errorbarh	line	qq	segment	text
bin2d	curve	freqpoly	linerange	qq_line	sf	tile
blank	density	hex	map	quantile	sf_label	violin
boxplot	density2d	histogram	path	raster	sf_text	vline
col	density_2d	hline	point	rect	smooth	

# Shape attribute values

□ 0 ○ 1 △ 2 + 3 × 4

◇ 5 ▽ 6 × 7 \* 8 ◆ 9

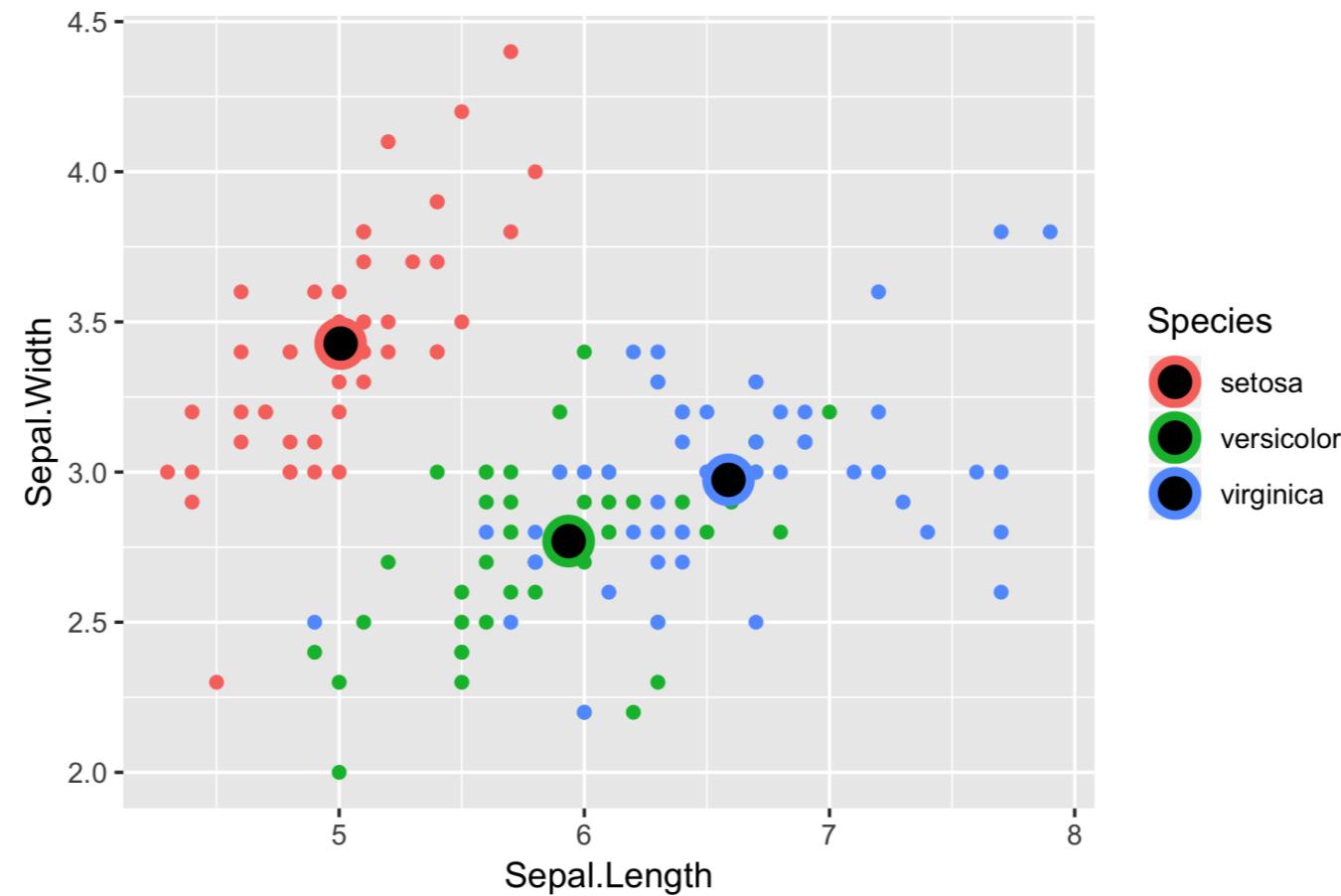
⊕ 10 △ 11 ■ 12 ⊗ 13 □ 14

■ 15 ● 16 ▲ 17 ♦ 18 ● 19

● 20 ○ 21 ■ 22 ◇ 23 ▲ 24 ▽ 25

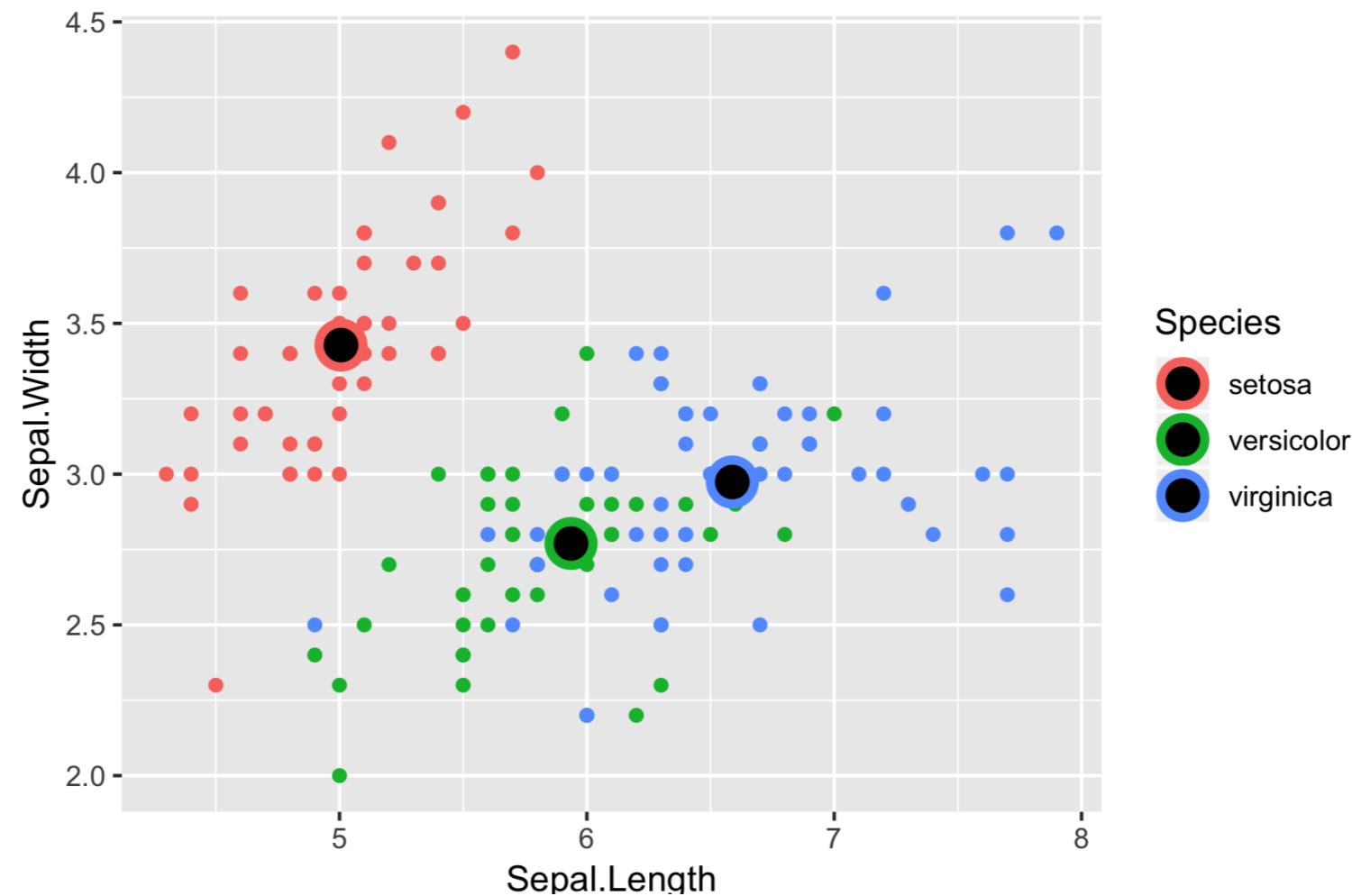
# Example

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +  
  geom_point() +  
  geom_point(data = iris.summary, shape = 21, size = 5,  
             fill = "black", stroke = 2)
```



# On-the-fly stats by ggplot2

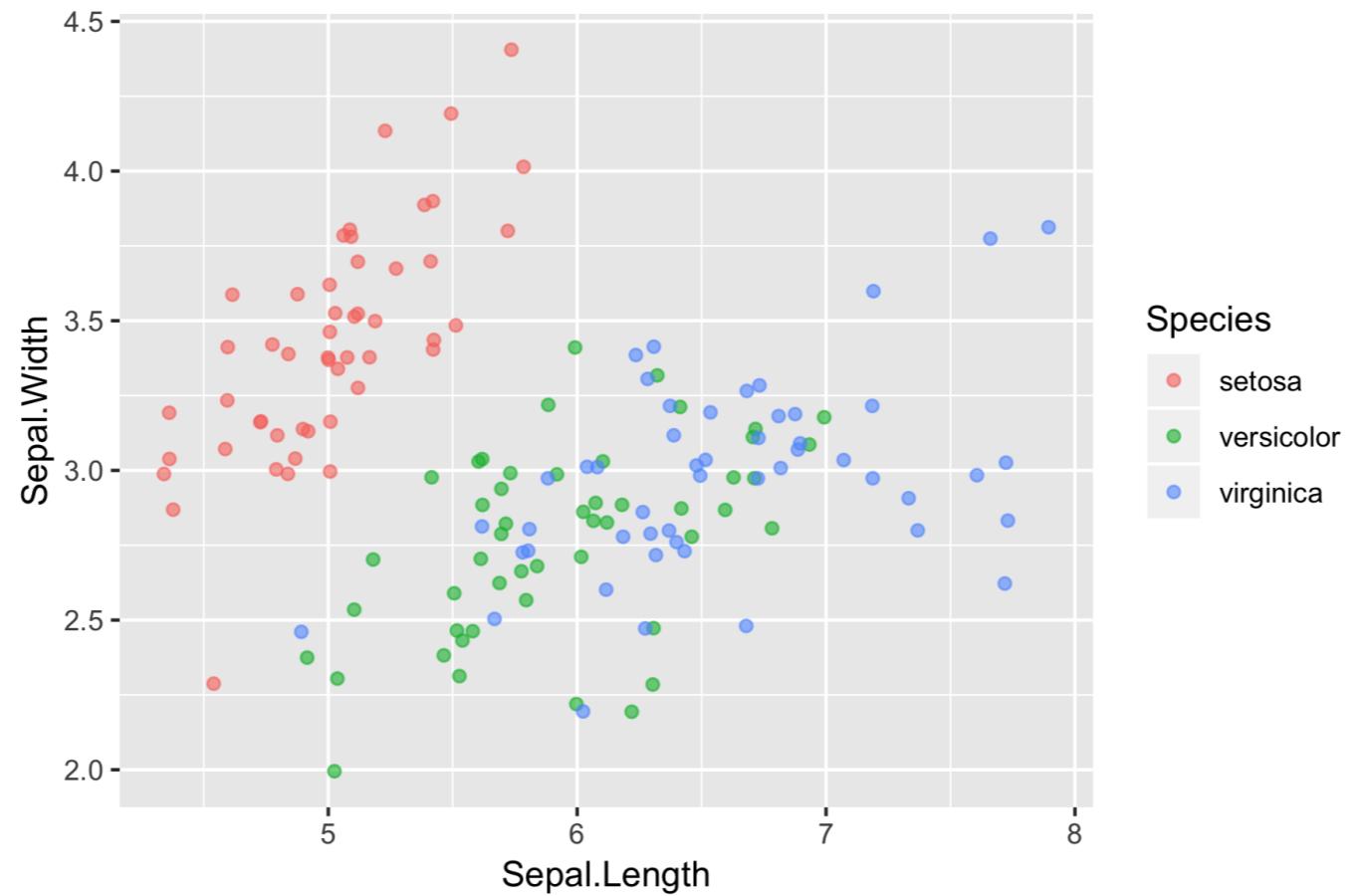
- See the second course for the stats layer.
- Note: Avoid plotting only the mean without a measure of spread, e.g. the standard deviation.



# Don't forget to adjust alpha

- Combine jittering with alpha-blending if necessary

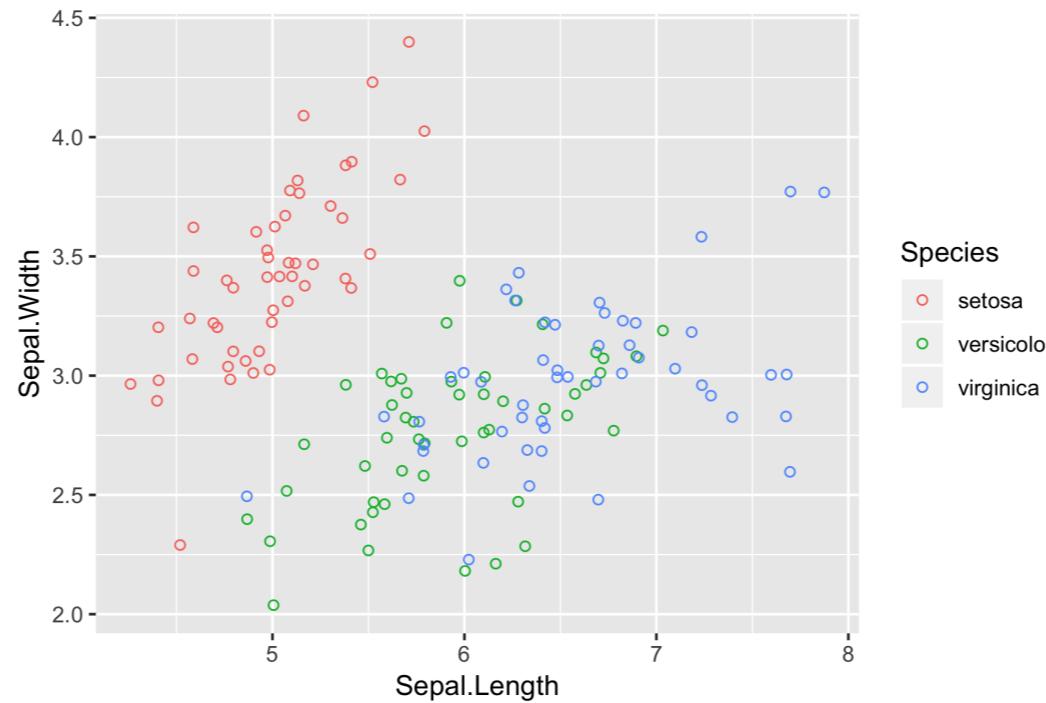
```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +  
  geom_jitter(alpha = 0.6)
```



# Hollow circles also help

- `shape = 1` is a. hollow circle.
- Not necessary to also use alpha-blending.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +  
  geom_jitter(shape = 1)
```



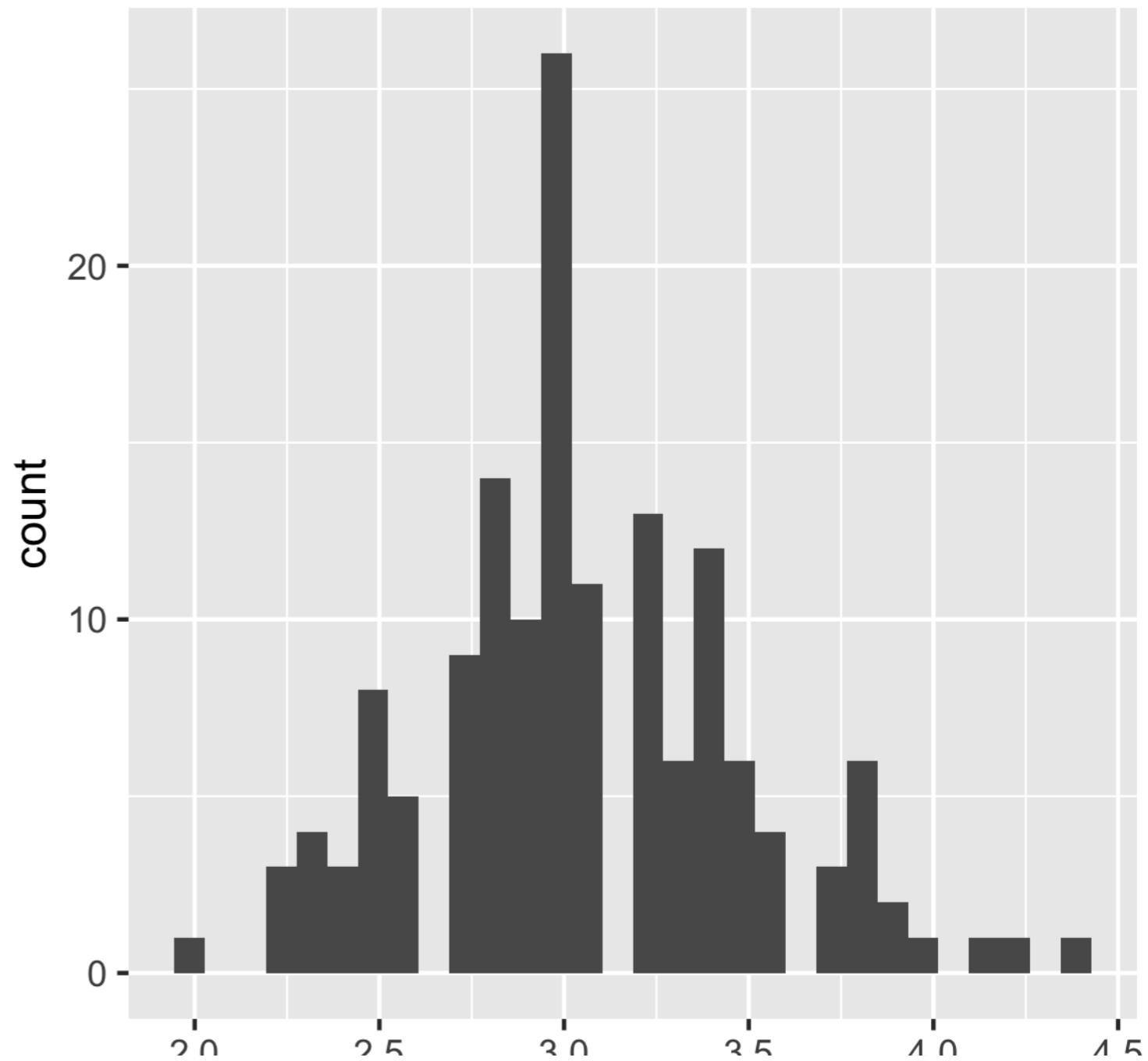
# Default of 30 even bins

```
ggplot(iris, aes(x = Sepal.Width)) +  
  geom_histogram()
```

- A plot of binned values
  - i.e. a statistical function

```
# Default bin width:  
diff(range(iris$Sepal.Width))/30
```

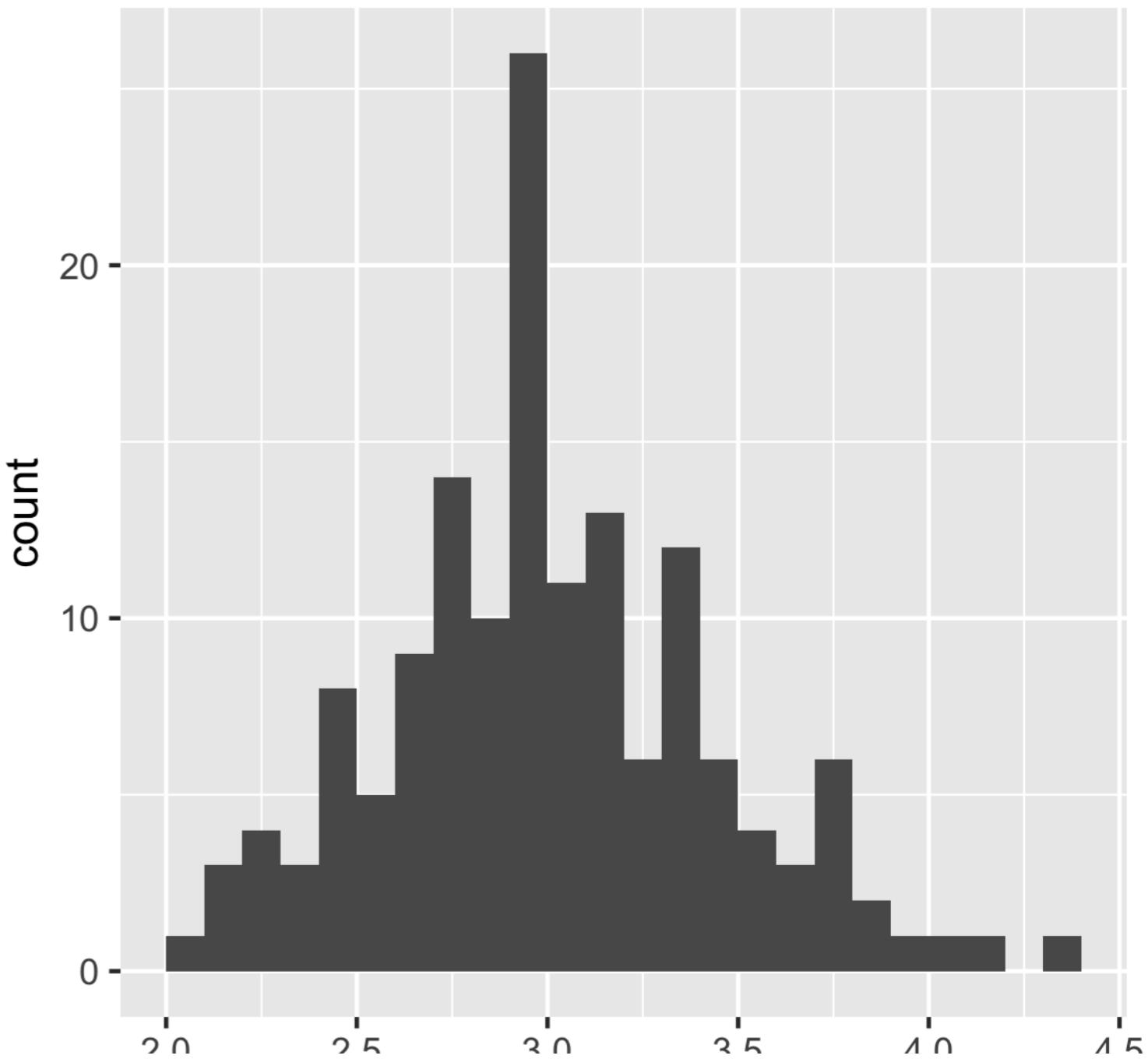
```
[1] 0.08
```



# Re-position tick marks

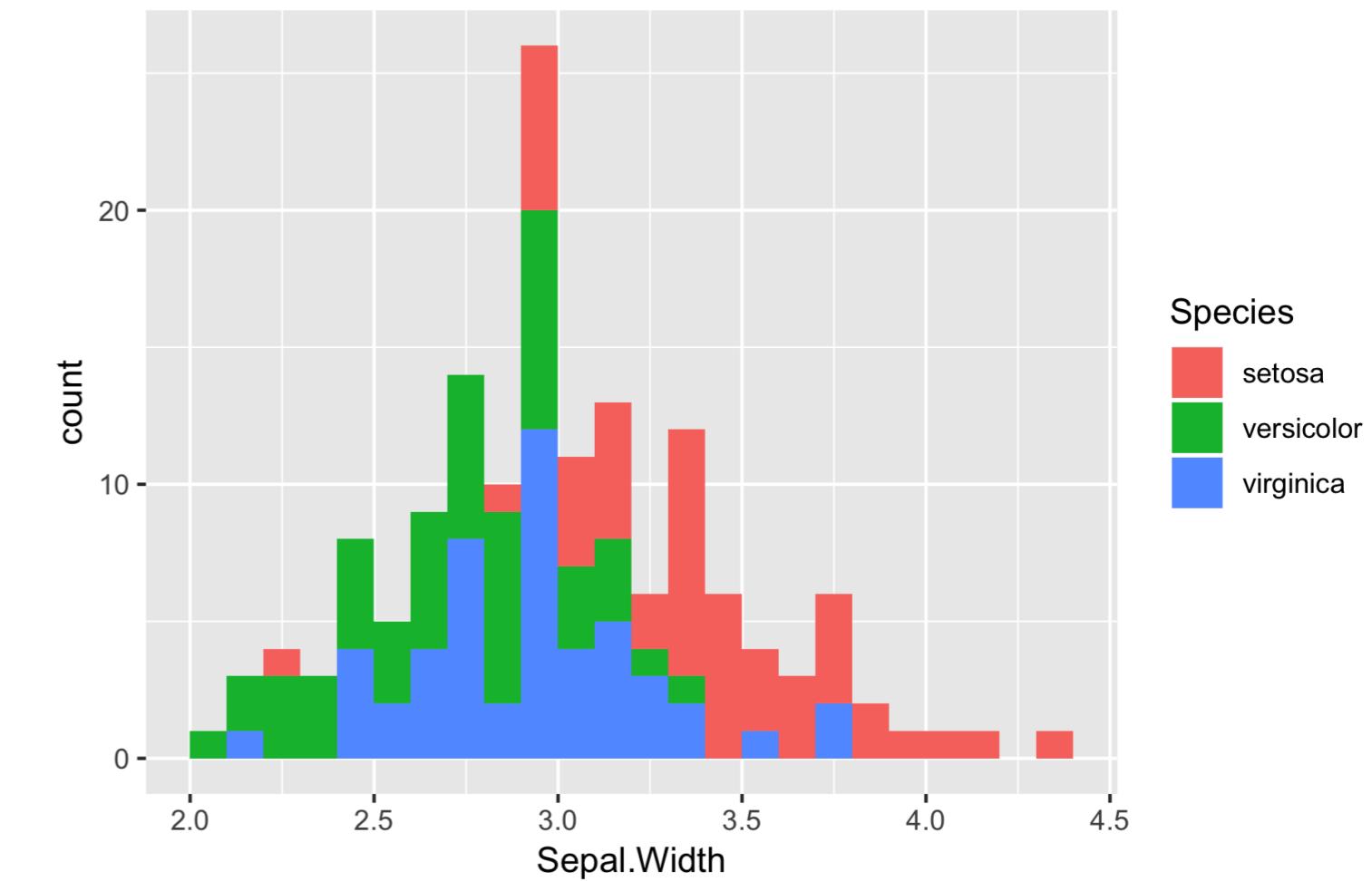
```
ggplot(iris, aes(x = Sepal.Width)) +  
  geom_histogram(binwidth = 0.1,  
                 center = 0.05)
```

- Always set a meaningful bin widths for your data.
- No spaces between bars.
- X axis labels are between bars.



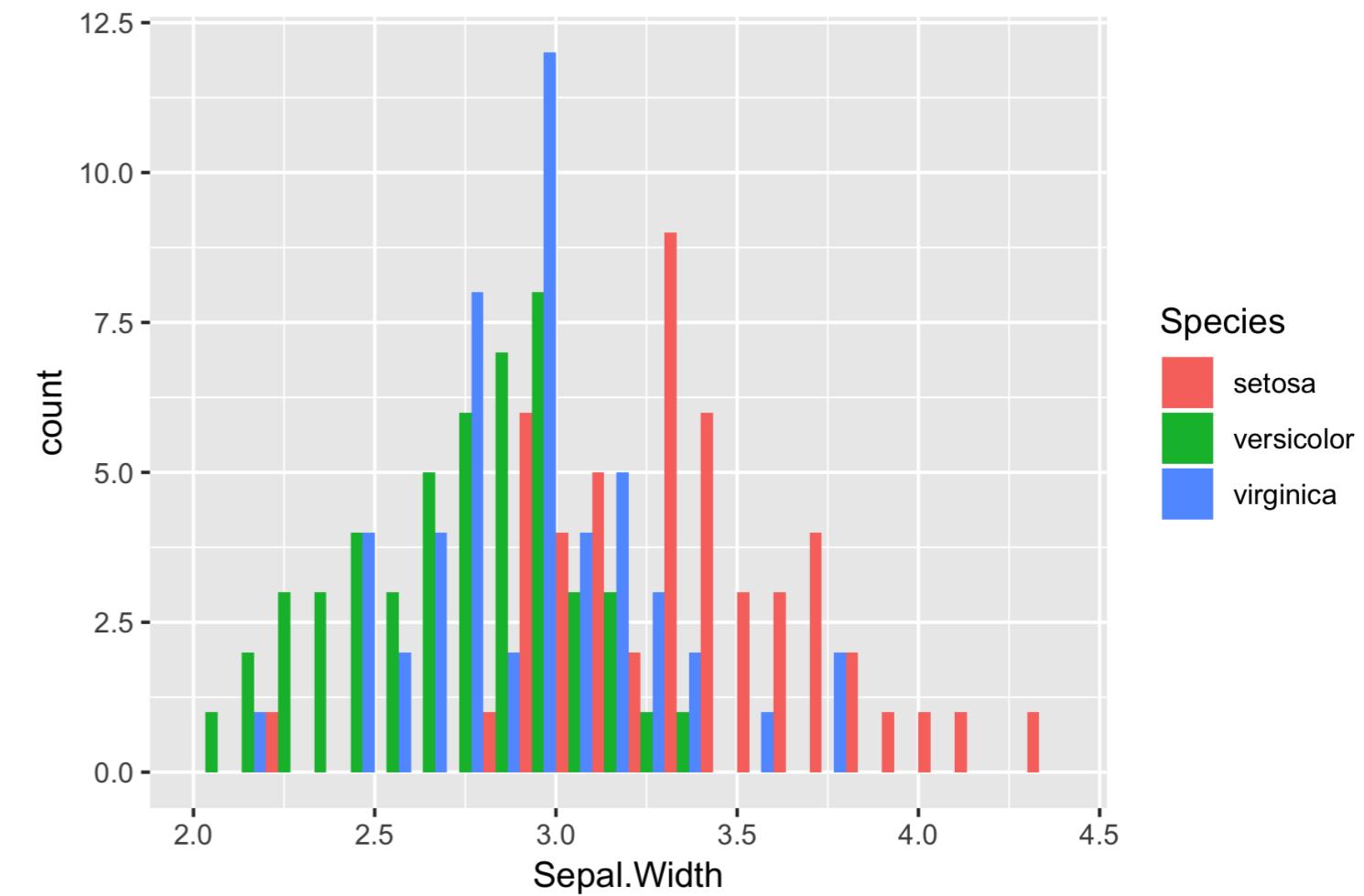
# Default position is "stack"

```
ggplot(iris, aes(x = Sepal.Width,  
                  fill = Species)) +  
  geom_histogram(binwidth = .1,  
                 center = 0.05,  
                 position = "stack")
```



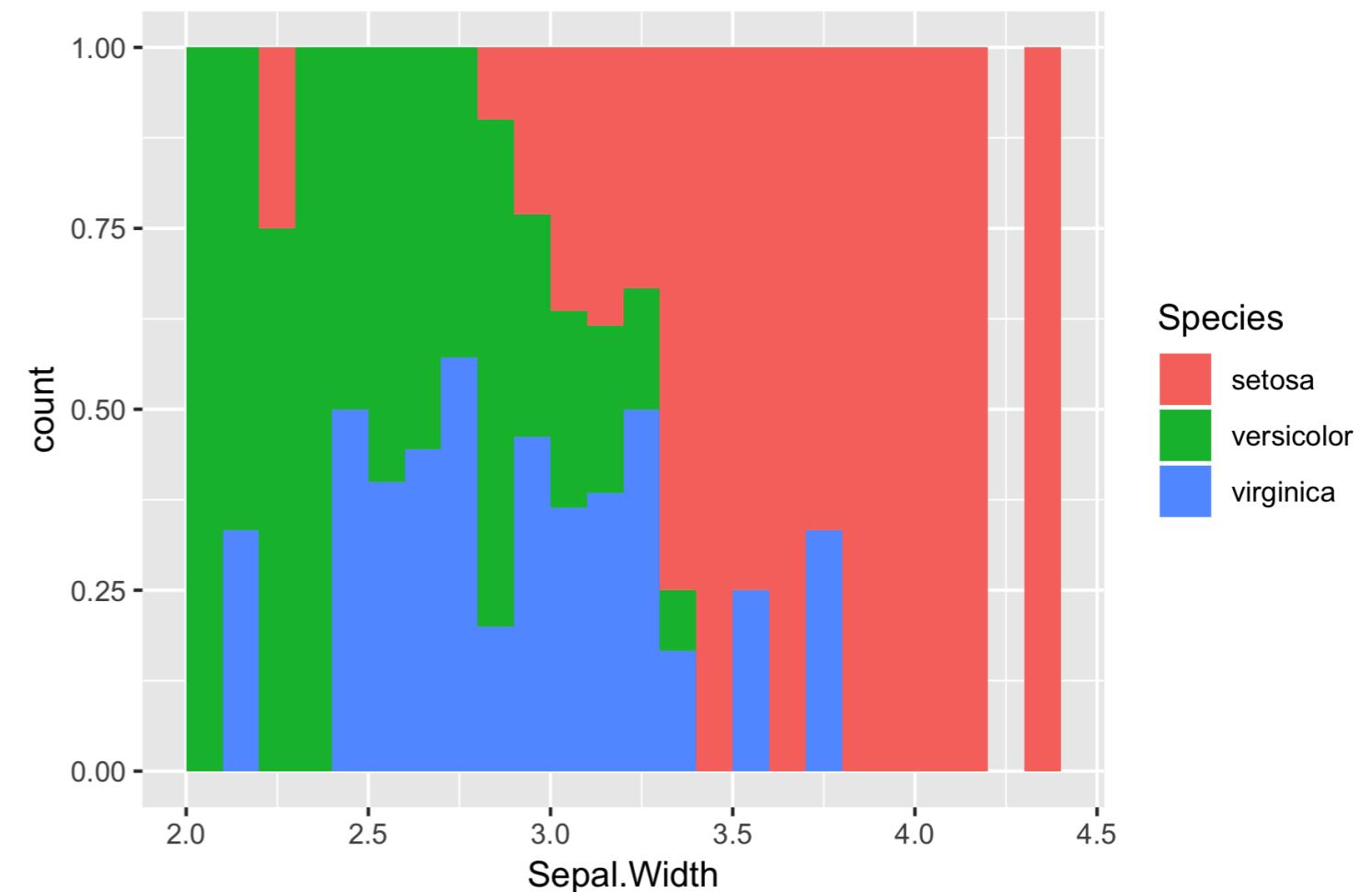
# position = "dodge"

```
ggplot(iris, aes(x = Sepal.Width,  
                 fill = Species)) +  
  geom_histogram(binwidth = .1,  
                 center = 0.05,  
                 position = "dodge")
```



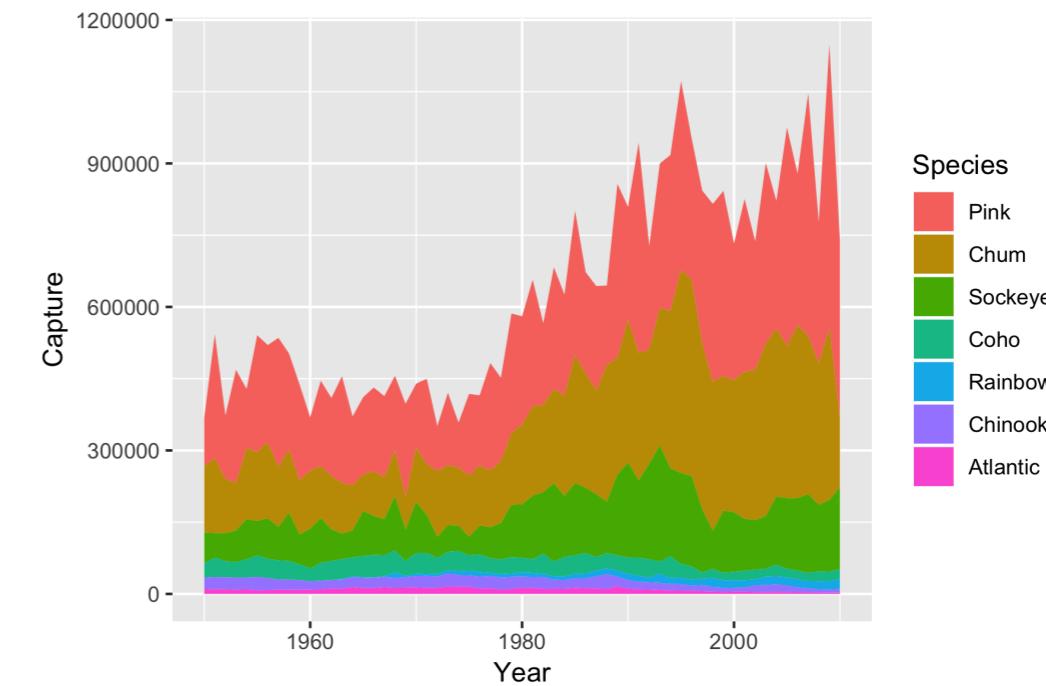
# position = "fill"

```
ggplot(iris, aes(x = Sepal.Width,  
                 fill = Species)) +  
  geom_histogram(binwidth = .1,  
                 center = 0.05,  
                 position = "fill")
```



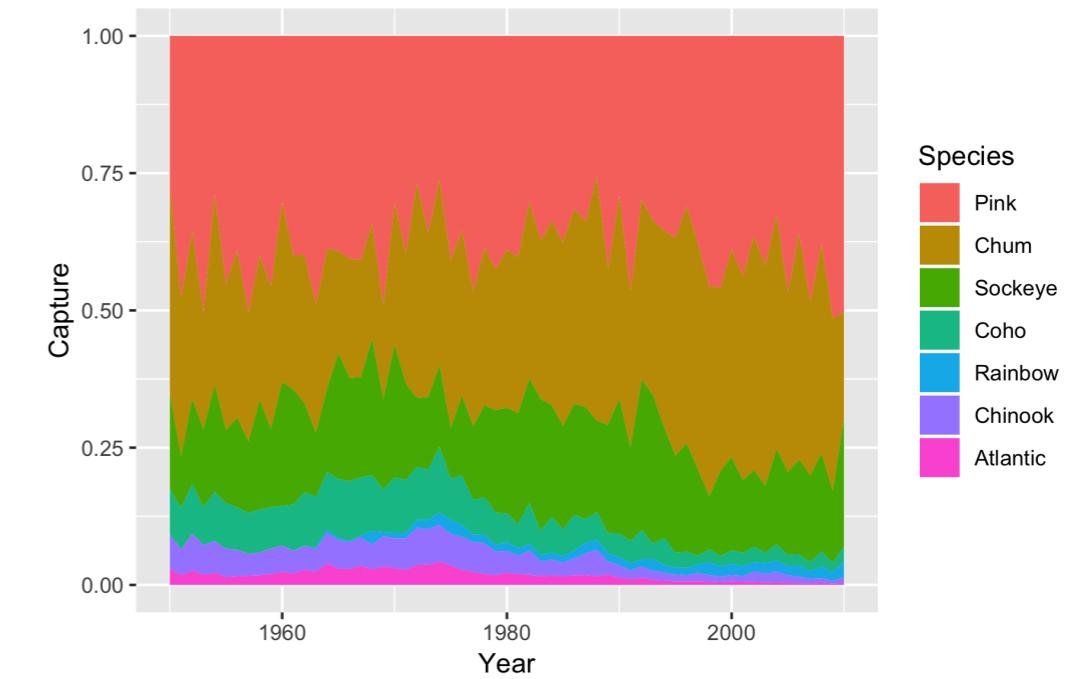
# Fill aesthetic with geom\_area()

```
ggplot(fish, aes(x = Year,  
                  y = Capture,  
                  fill = Species)) +  
  
  geom_area()
```



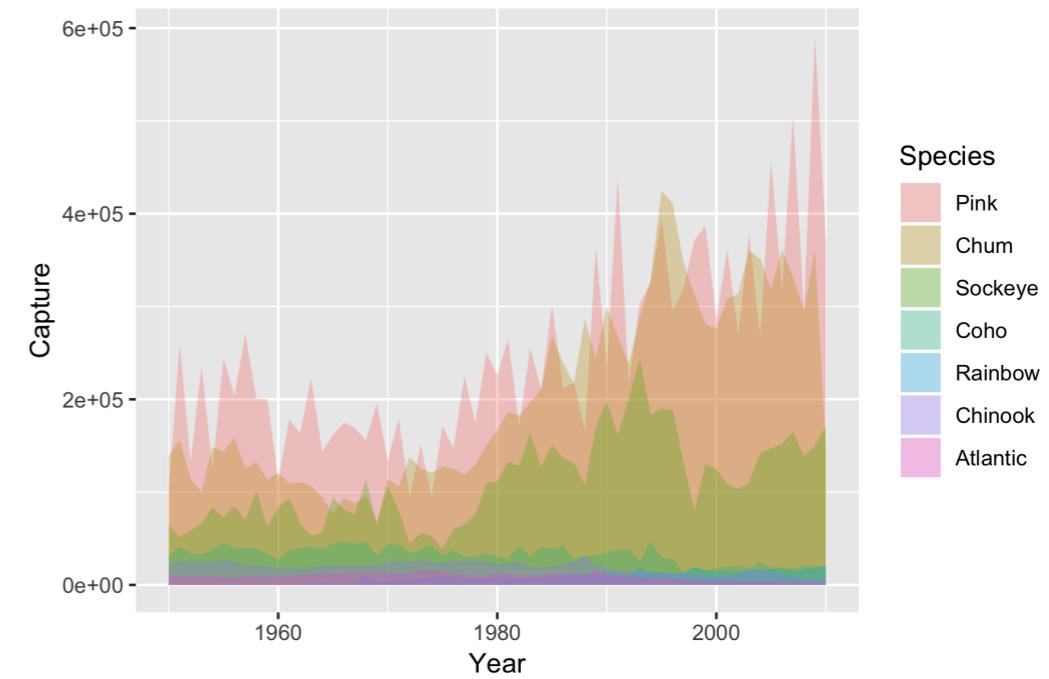
# Using position = "fill"

```
ggplot(fish, aes(x = Year,  
                  y = Capture,  
                  fill = Species)) +  
  geom_area(position = "fill")
```

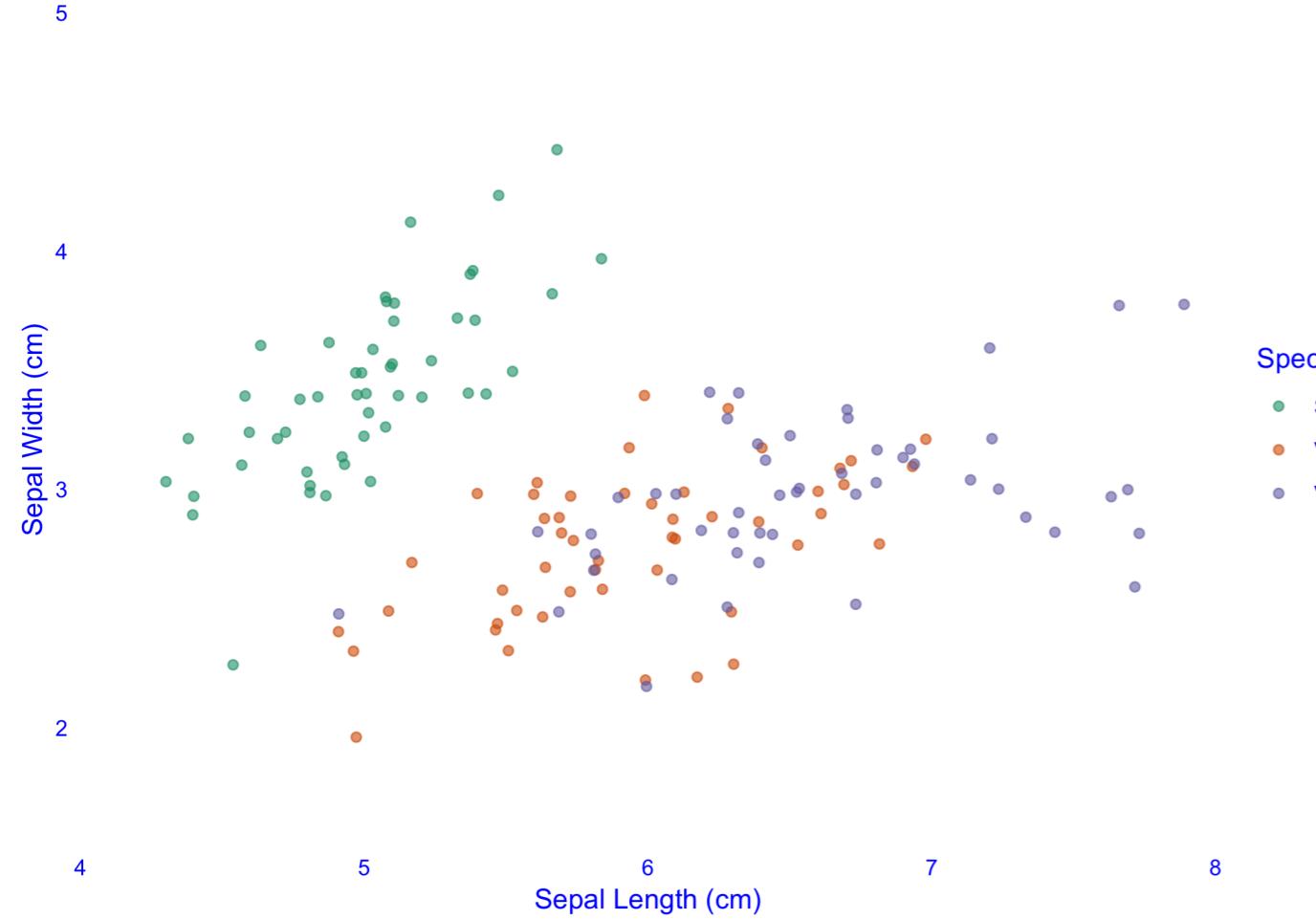


# geom\_ribbon()

```
ggplot(fish, aes(x = Year,  
                  y = Capture,  
                  fill = Species)) +  
  geom_ribbon(aes(ymax = Capture,  
                 ymin = 0),  
              alpha = 0.3)
```

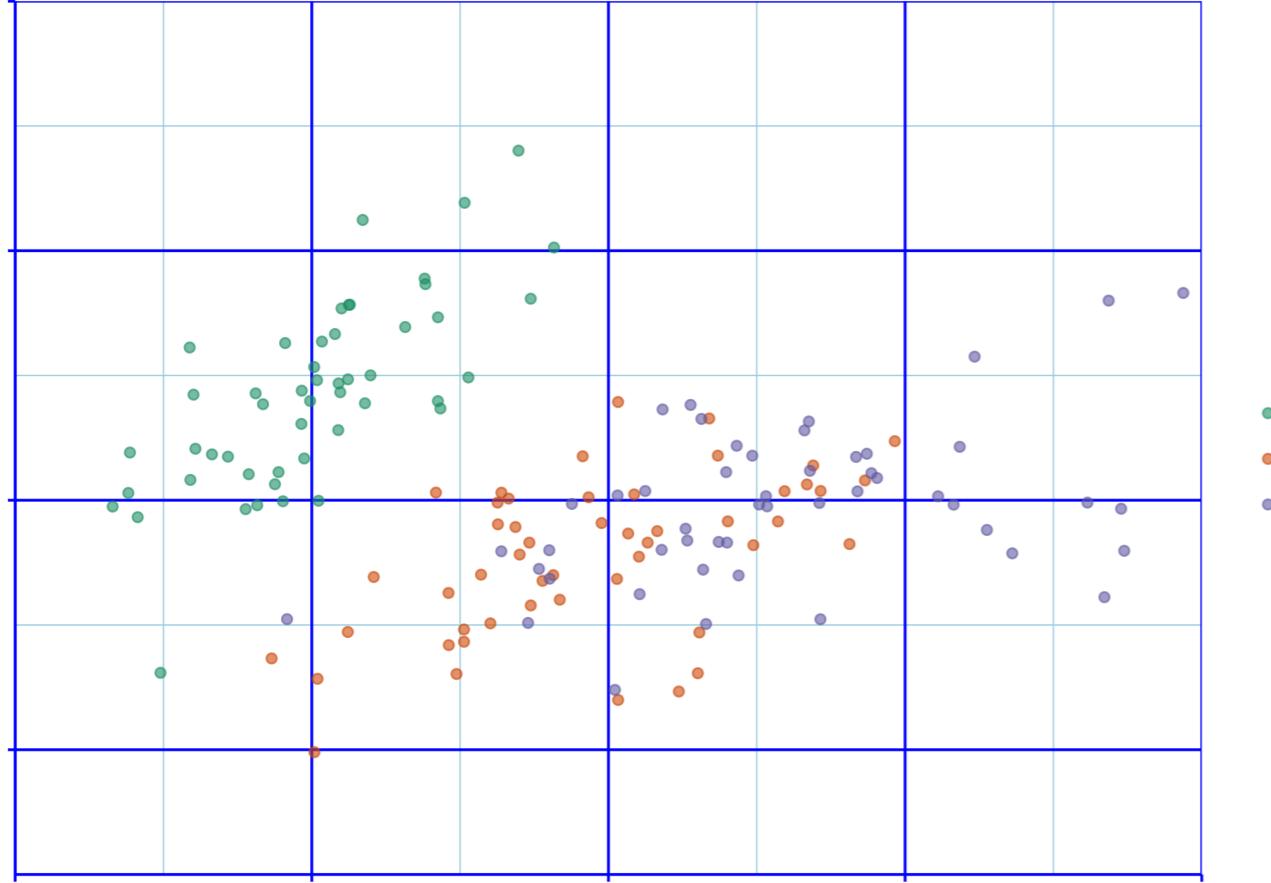


# The text elements



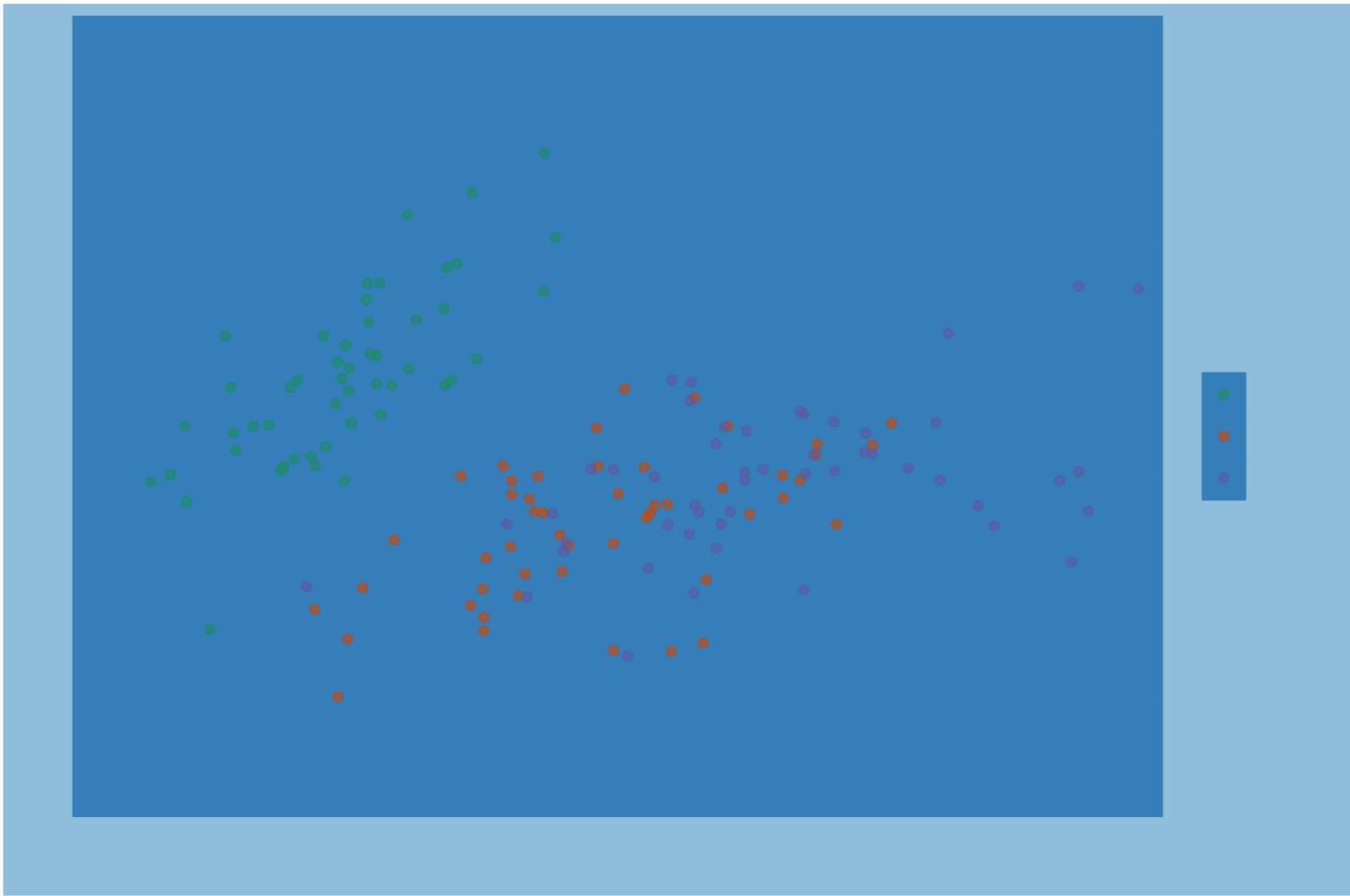
```
theme(  
  text,  
    axis.title,  
    axis.title.x,  
    axis.title.x.top,  
    axis.title.x.bottom,  
    axis.title.y,  
    axis.title.y.left,  
    axis.title.y.right,  
  title,  
    legend.title,  
    plot.title,  
    plot.subtitle,  
    plot.caption,  
    plot.tag,  
  axis.text,  
    axis.text.x,  
    axis.text.x.top,  
    axis.text.x.bottom,  
    axis.text.y,  
    axis.text.y.left,  
    axis.text.y.right,  
  legend.text,  
  strip.text,  
  strip.text.x,  
  strip.text.y)
```

# Line elements



```
theme(  
  line,  
  axis.ticks,  
  axis.ticks.x,  
  axis.ticks.x.top,  
  axis.ticks.x.bottom,  
  axis.ticks.y,  
  axis.ticks.y.left,  
  axis.ticks.y.right,  
  axis.line,  
  axis.line.x,  
  axis.line.x.top,  
  axis.line.x.bottom,  
  axis.line.y,  
  axis.line.y.left,  
  axis.line.y.right,  
  panel.grid,  
  panel.grid.major,  
  panel.grid.major.x,  
  panel.grid.major.y,  
  panel.grid.minor,  
  panel.grid.minor.x,  
  panel.grid.minor.y)
```

# Rect elements



```
theme(  
  rect,  
  legend.background,  
  legend.key,  
  legend.box.background,  
  panel.background,  
  panel.border,  
  plot.background,  
  strip.background,  
  strip.background.x,  
  strip.background.y)
```

# Modifying whitespace

**Whitespace** means all the non-visible margins and spacing in the plot.

To set a single whitespace value, use `unit(x, unit)`, where `x` is the amount and `unit` is the unit of measure.

Borders require you to set 4 positions, so use

`margin(top, right, bottom, left, unit)`. To remember the margin order, think **T**Rou**B**Le.

The default unit is "pt" (points), which scales well with text. Other options include "cm", "in" (inches) and "lines" (of text).

`plt_mpg_vs_wt_by_cyl` is available. The panel and legend are wrapped in blue boxes so you can see how they change.

```
1 # View the original plot
2 plt_mpg_vs_wt_by_cyl
3
4 plt_mpg_vs_wt_by_cyl +
5   theme(
6     # Set the axis tick length to 2 lines
7     axis.ticks.length = unit(2,"lines")
8   )
```



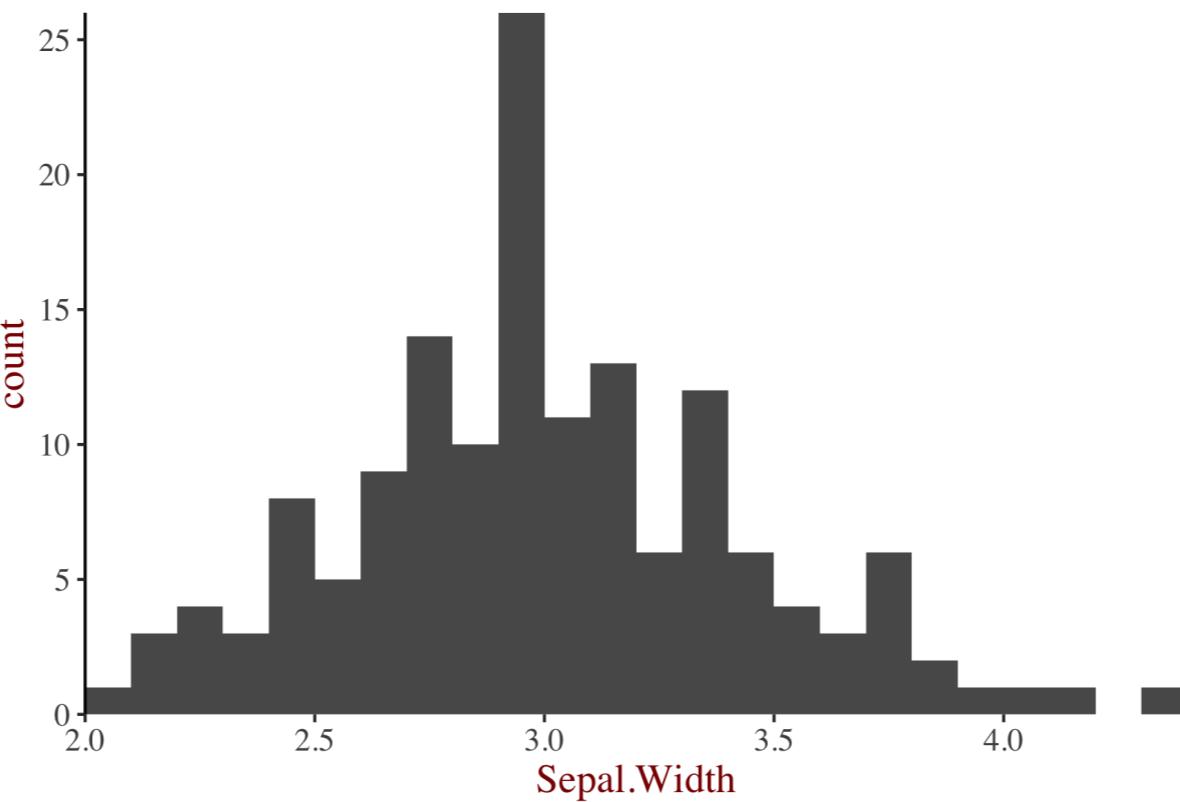
- Give the axis tick length, `axis.ticks.length`, a unit of 2 "lines".
- Give the legend key size, `legend.key.size`, a unit of 3 centimeters ("cm").
- Set the `legend.margin` to 20 points ("pt") on the top, 30 pts on the right, 40 pts on the bottom, and 50 pts on the left.
- Set the plot margin, `plot.margin`, to 10, 30, 50, and 70 millimeters ("mm").

# Defining theme objects

```
theme_iris <- theme(text = element_text(family = "serif", size = 14),  
rect = element_blank(),  
panel.grid = element_blank(),  
title = element_text(color = "#8b0000"),  
axis.line = element_line(color = "black"))
```

# Reusing theme objects

```
m +  
  theme_iris +  
  theme(axis.line.x = element_blank())
```

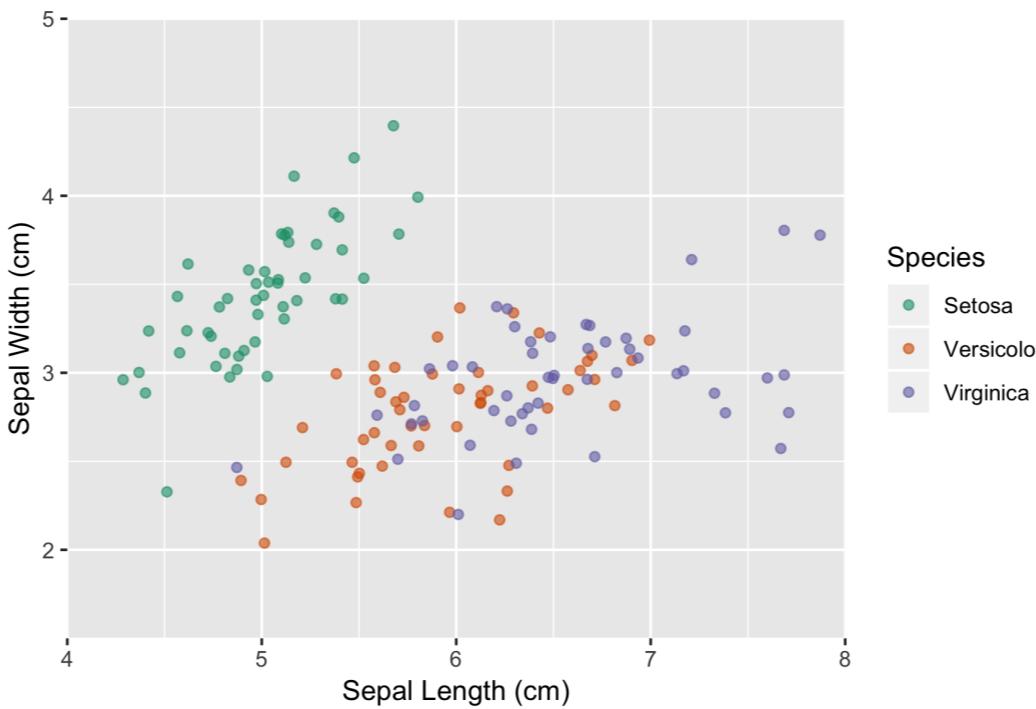


# Setting themes

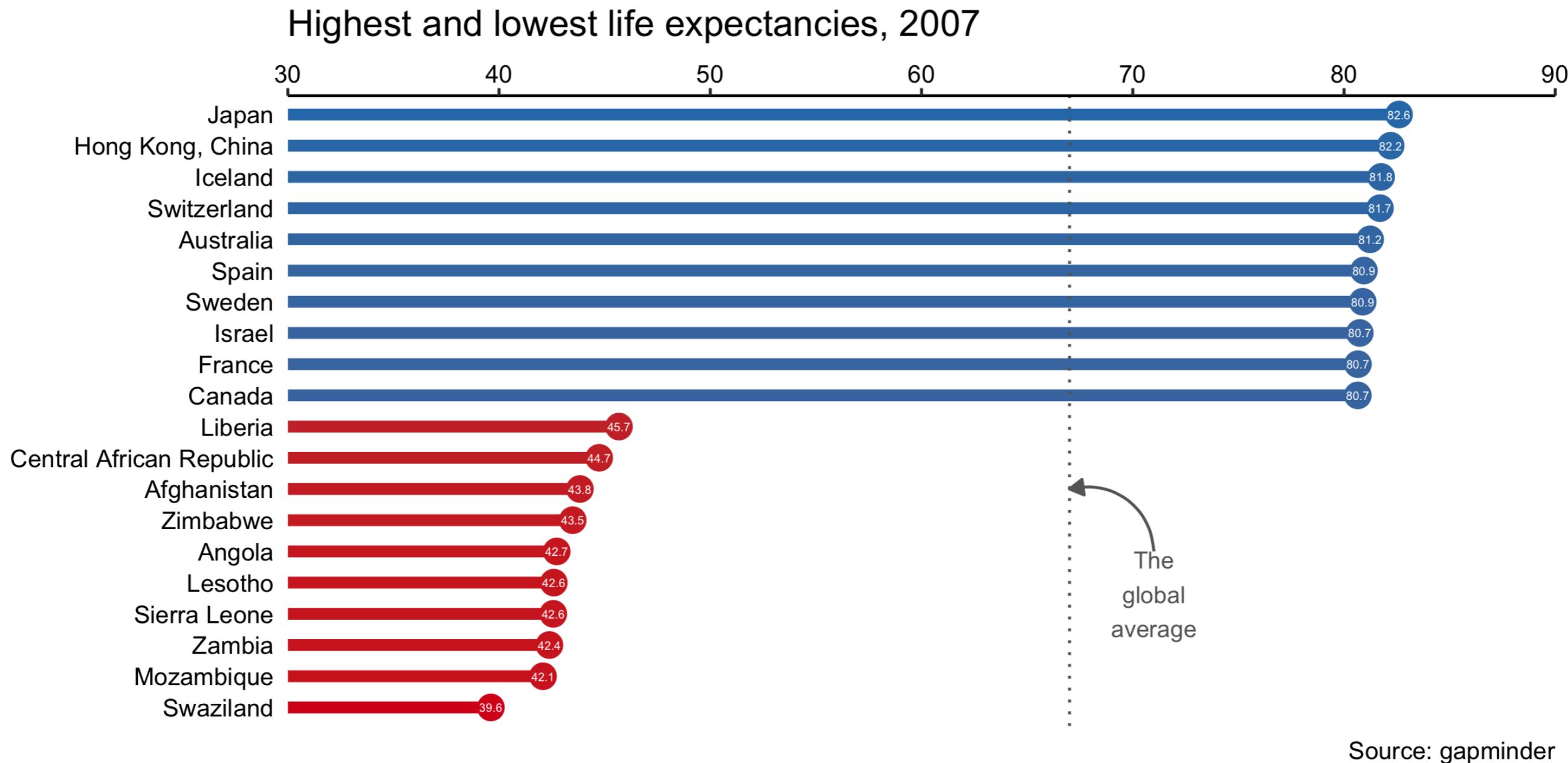
```
theme_set(original)
```

```
# Alternatively
```

```
# theme_set(theme_grey())
```



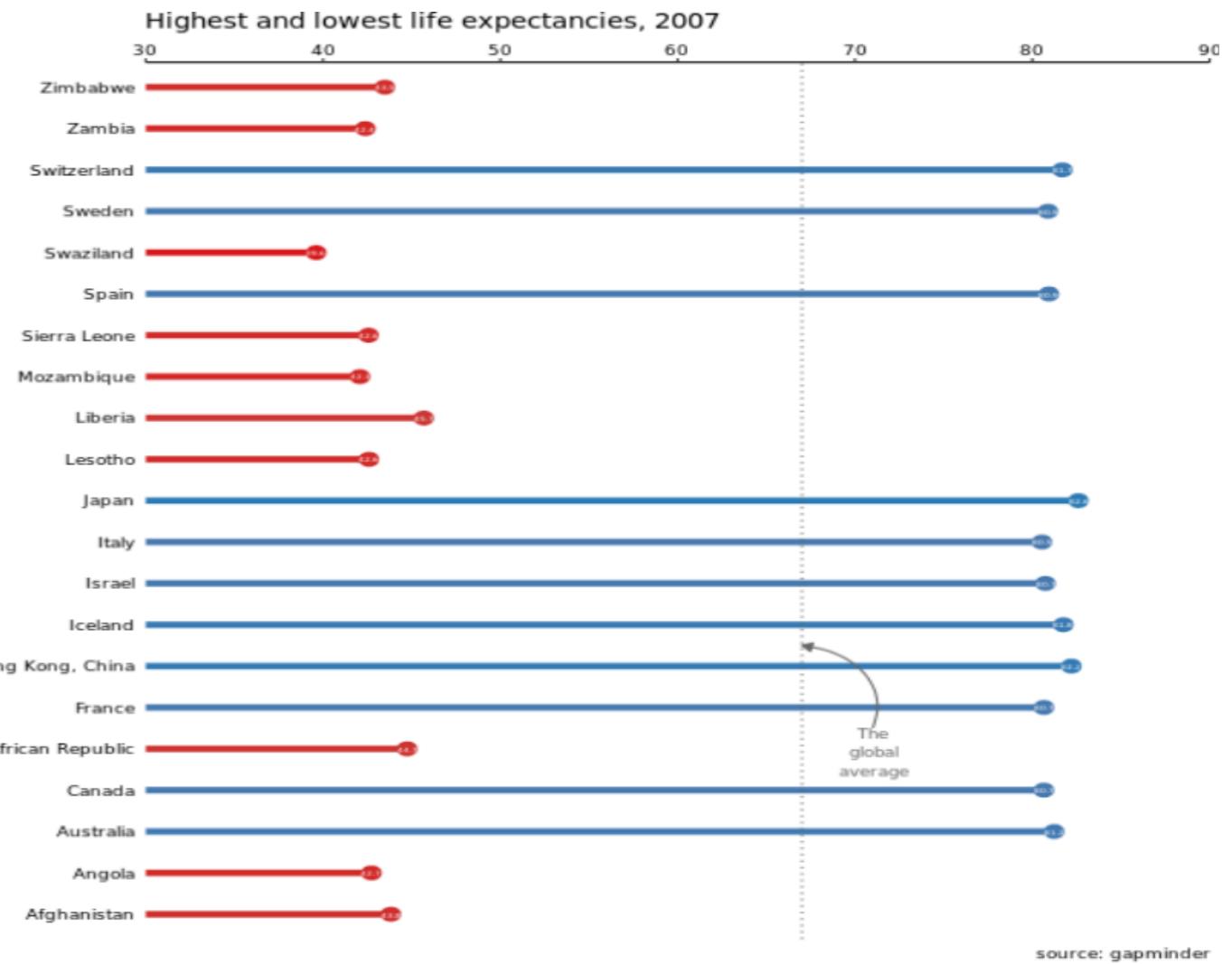
# Add embellishments



```

# Add a curve
plt_country_vs_lifeExp +
  step_1_themes +
  geom_vline(xintercept = global_mean, color = "grey40", linetype = 3) +
  step_3_annotation +
  annotate(
    "curve",
    x = x_start, y = y_start,
    xend = x_end, yend = y_end,
    arrow = arrow(length = unit(0.2, "cm"), type = "closed"),
    color = "grey40"
)

```



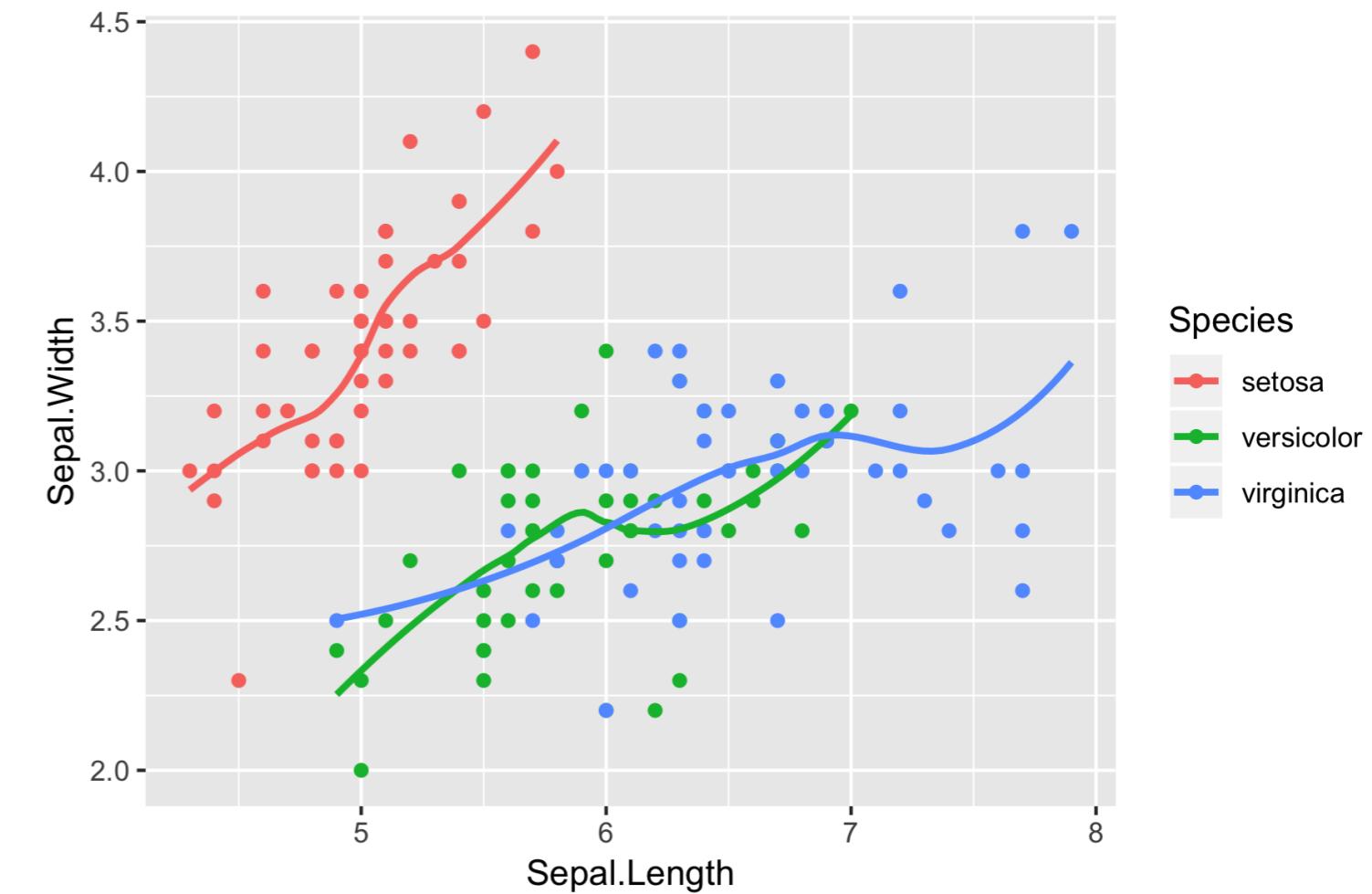
# The geom\_/stat\_ connection

stat_	geom_
stat_bin()	geom_histogram() , geom_freqpoly()
stat_count()	geom_bar()

# `stat_smooth(se = FALSE)`

```
ggplot(iris, aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 color = Species)) +  
  geom_point() +  
  geom_smooth(se = FALSE)
```

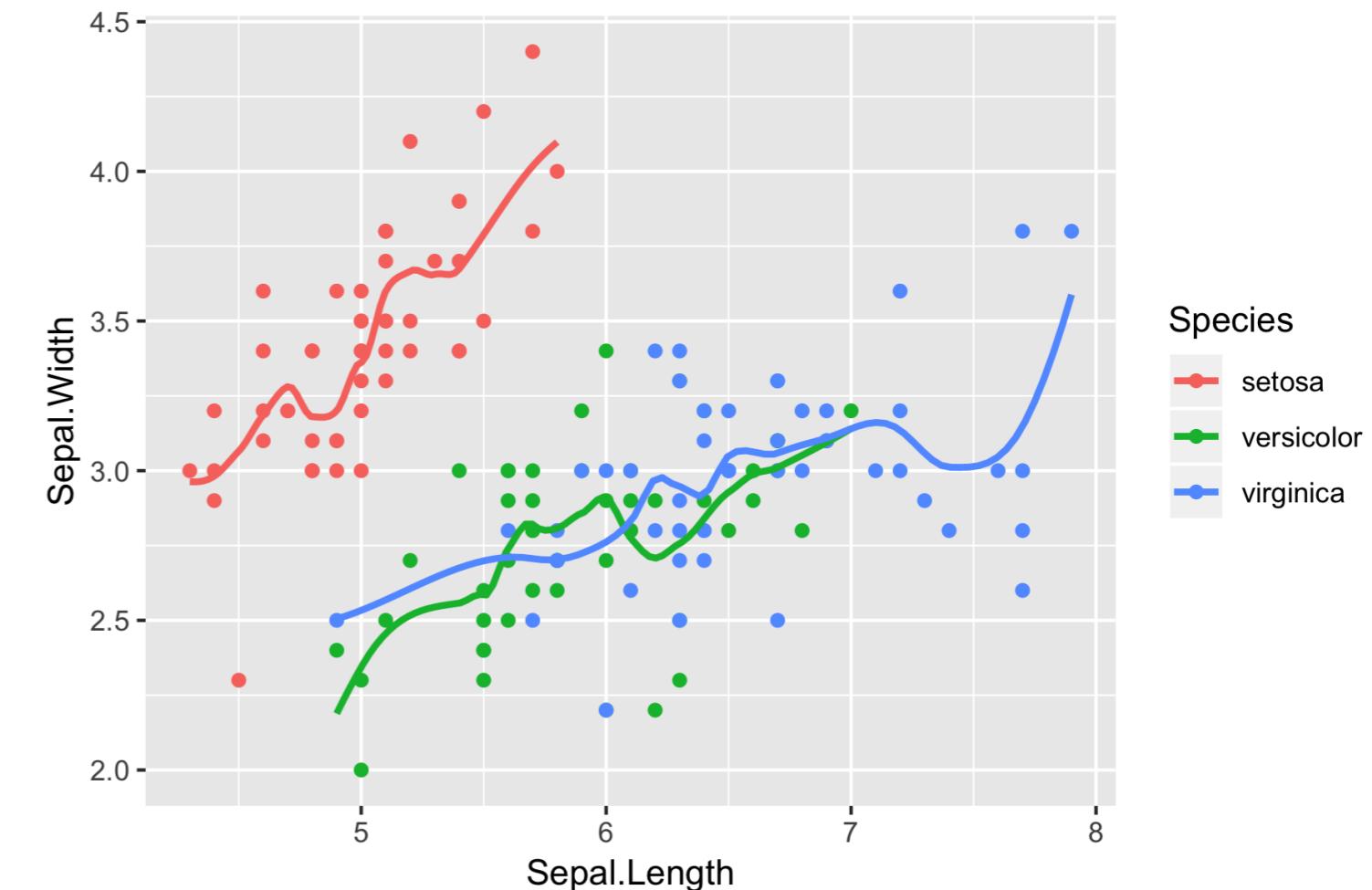
`geom_smooth()` using `method = 'loess'` and  
`formula 'y ~ x'`



# geom\_smooth(span = 0.4)

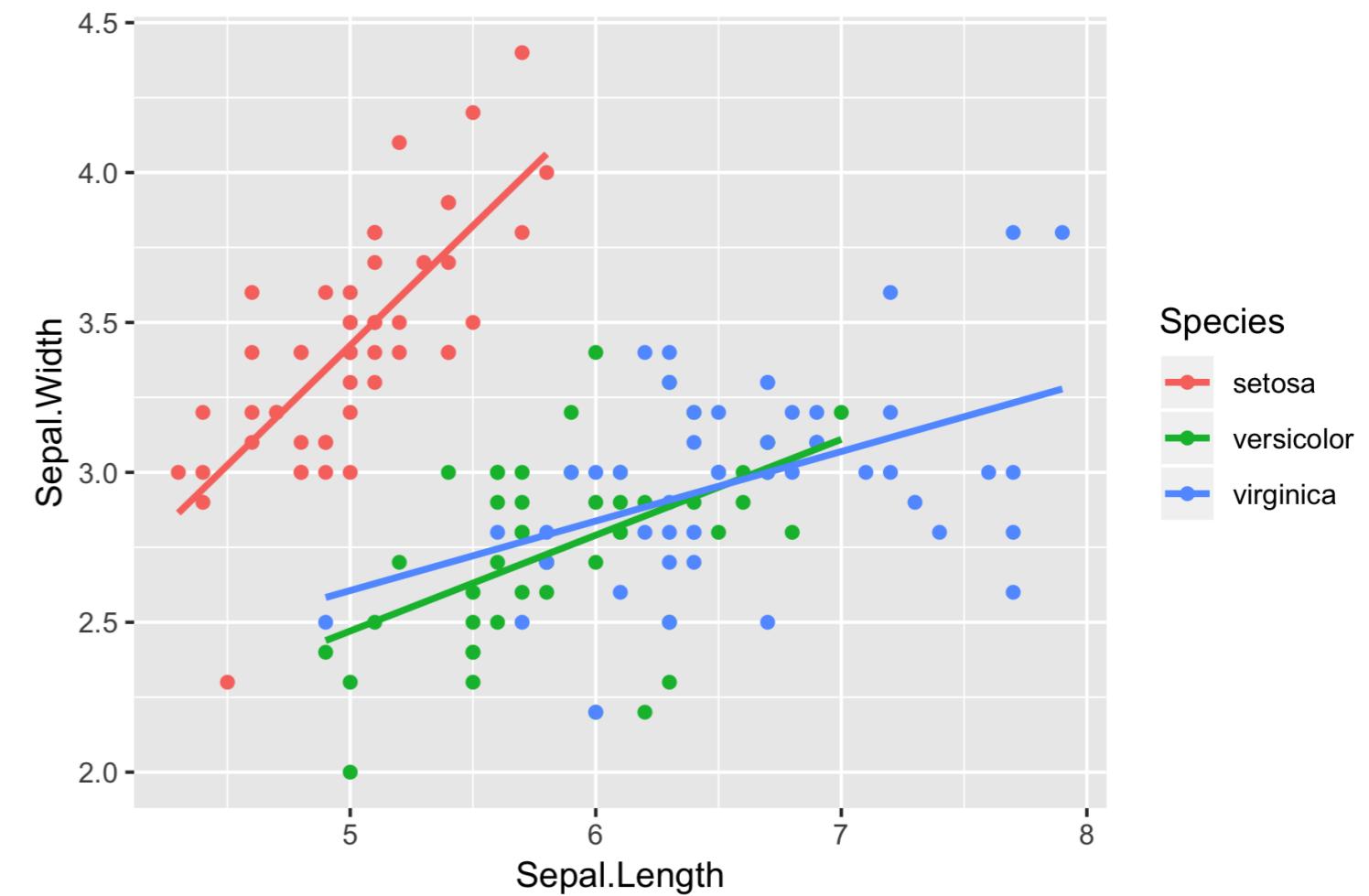
```
ggplot(iris, aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 color = Species)) +  
  geom_point() +  
  geom_smooth(se = FALSE, span = 0.4)
```

geom\_smooth() using method = 'loess' and  
formula 'y ~ x'



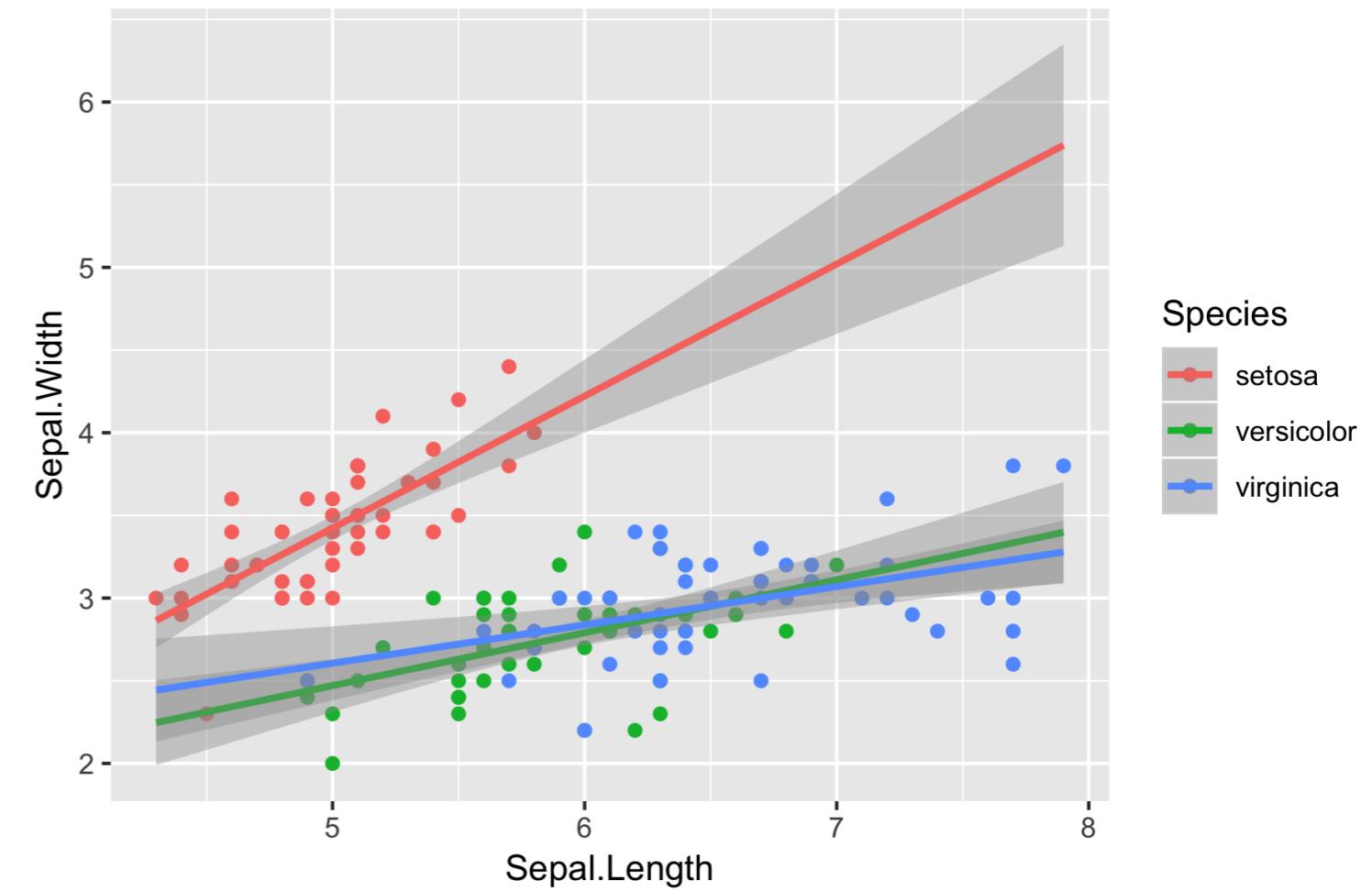
# geom\_smooth(method = "lm")

```
ggplot(iris, aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 color = Species)) +  
  
  geom_point() +  
  
  geom_smooth(method = "lm", se = FALSE)
```

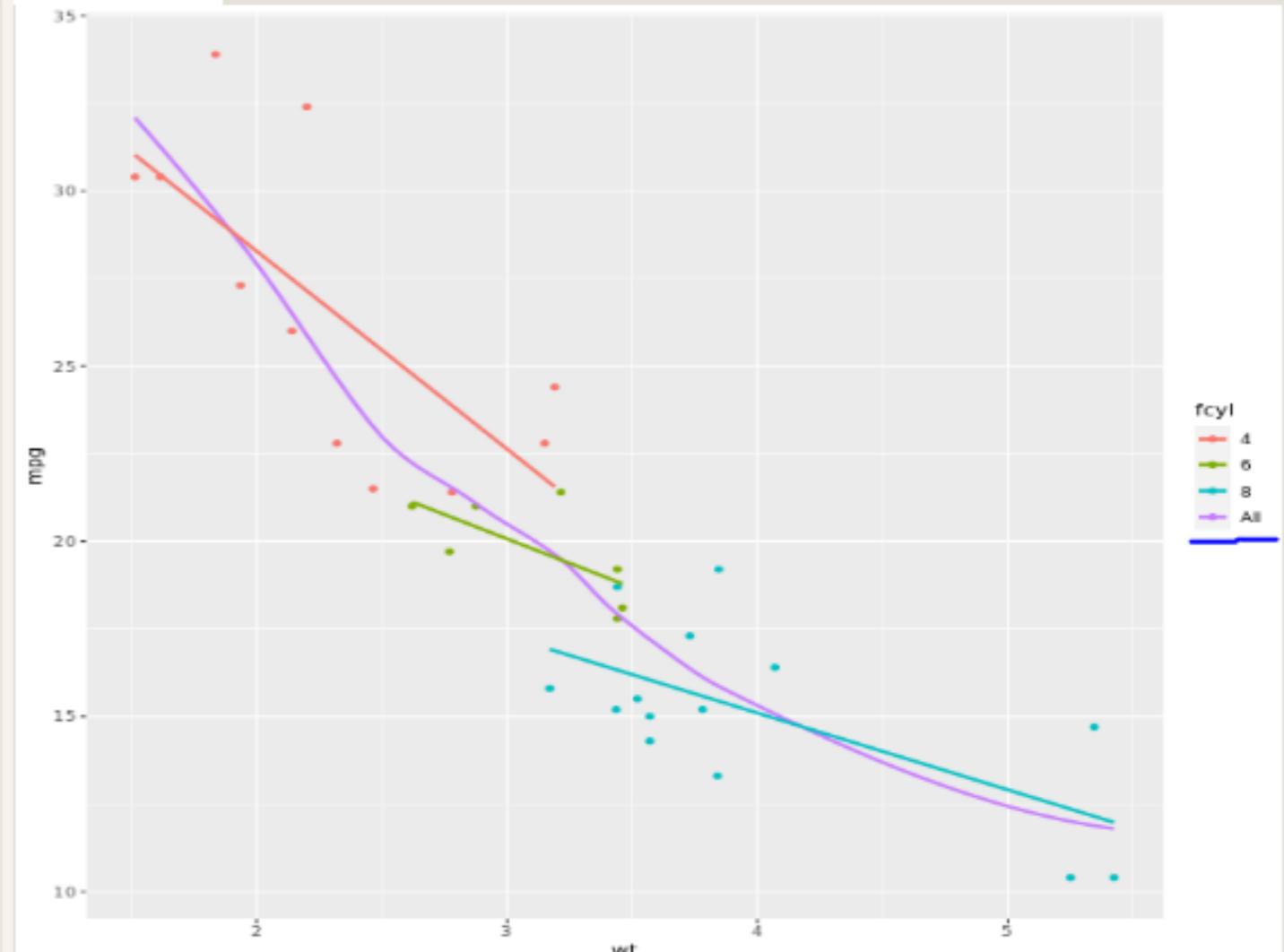


# geom\_smooth(fullrange = TRUE)

```
ggplot(iris, aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 color = Species)) +  
  
  geom_point() +  
  geom_smooth(method = "lm",  
              fullrange = TRUE)
```



```
1 # Amend the plot
2 ggplot(mtcars, aes(x = wt, y = mpg,
3 |   color = fcyl)) +
4   geom_point() +
5   # Map color to dummy variable "All"
6   stat_smooth(aes(color = "All"),
7   |   |   |   |   se = FALSE) +
8   stat_smooth(method = "lm", se = FALSE)
```



Run Code

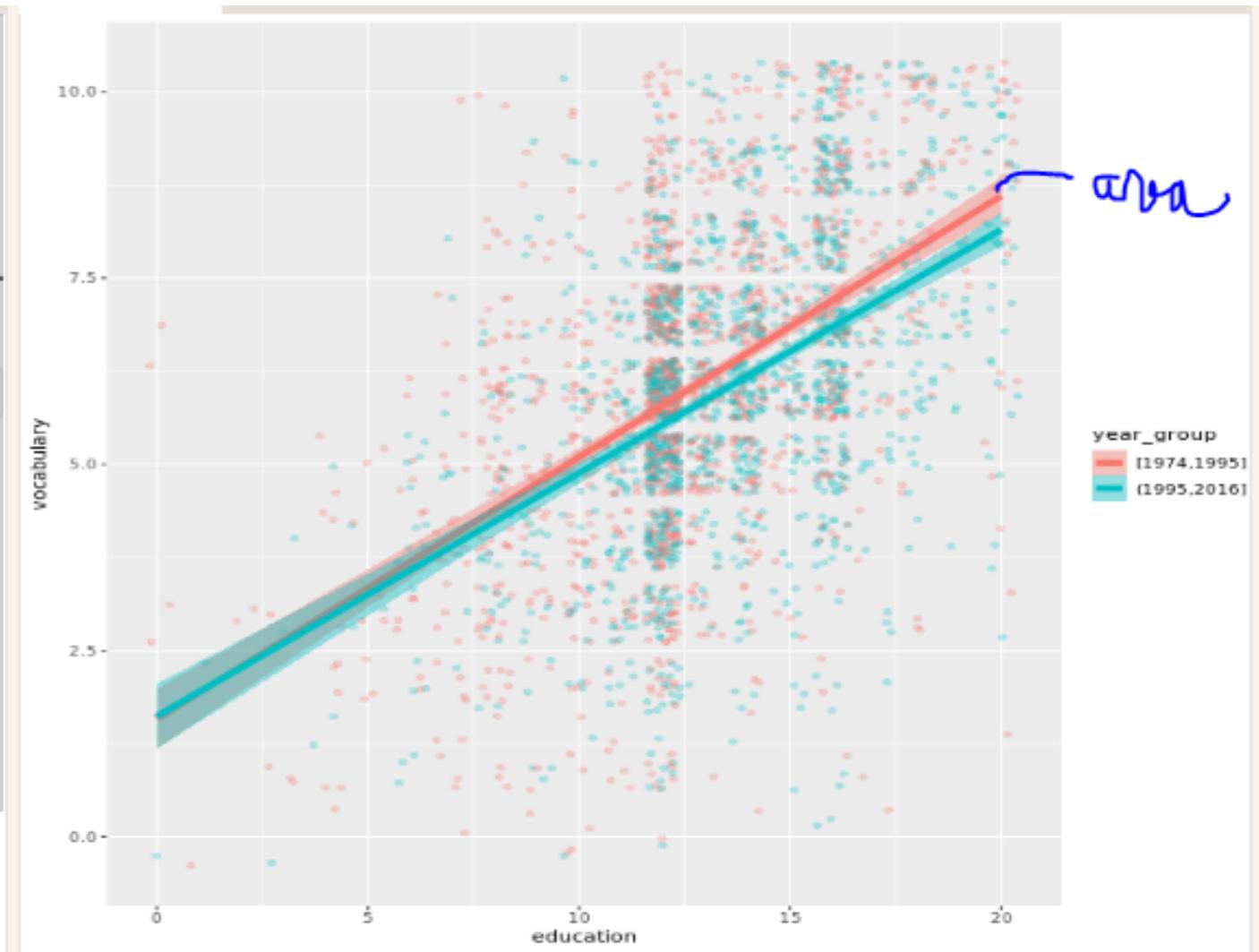
Submit Answer

← Previous Plot

4/4

→ Next Plot

```
1 # Amend the plot
2 ggplot(Vocab, aes(x = education, y =
3   vocabulary, color = year_group)) +
4   geom_jitter(alpha = 0.25) +
5   # Map the fill color to year_group, set
6   # the line size to 2
7   stat_smooth(aes(fill = year_group),
8   method = "lm", size = 2)
```



# Other stat\_ functions

stat_	geom_
stat_boxplot()	geom_boxplot()
stat_bindot()	geom_dotplot()
stat_bin2d()	geom_bin2d()
stat_binhex()	geom_hex()

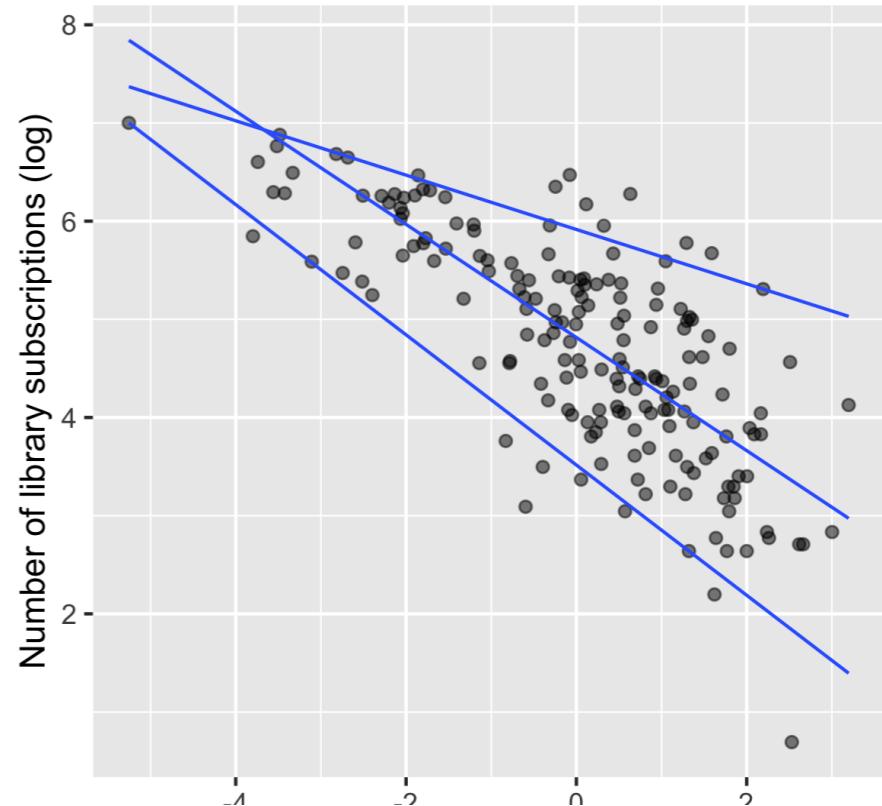
# Other stat\_ functions

stat_	geom_
stat_boxplot()	geom_boxplot()
stat_bindot()	geom_dotplot()
stat_bin2d()	geom_bin2d()
stat_binhex()	geom_hex()
stat_contour()	geom_contour()
stat_quantile()	geom_quantile()
stat_sum()	geom_count()

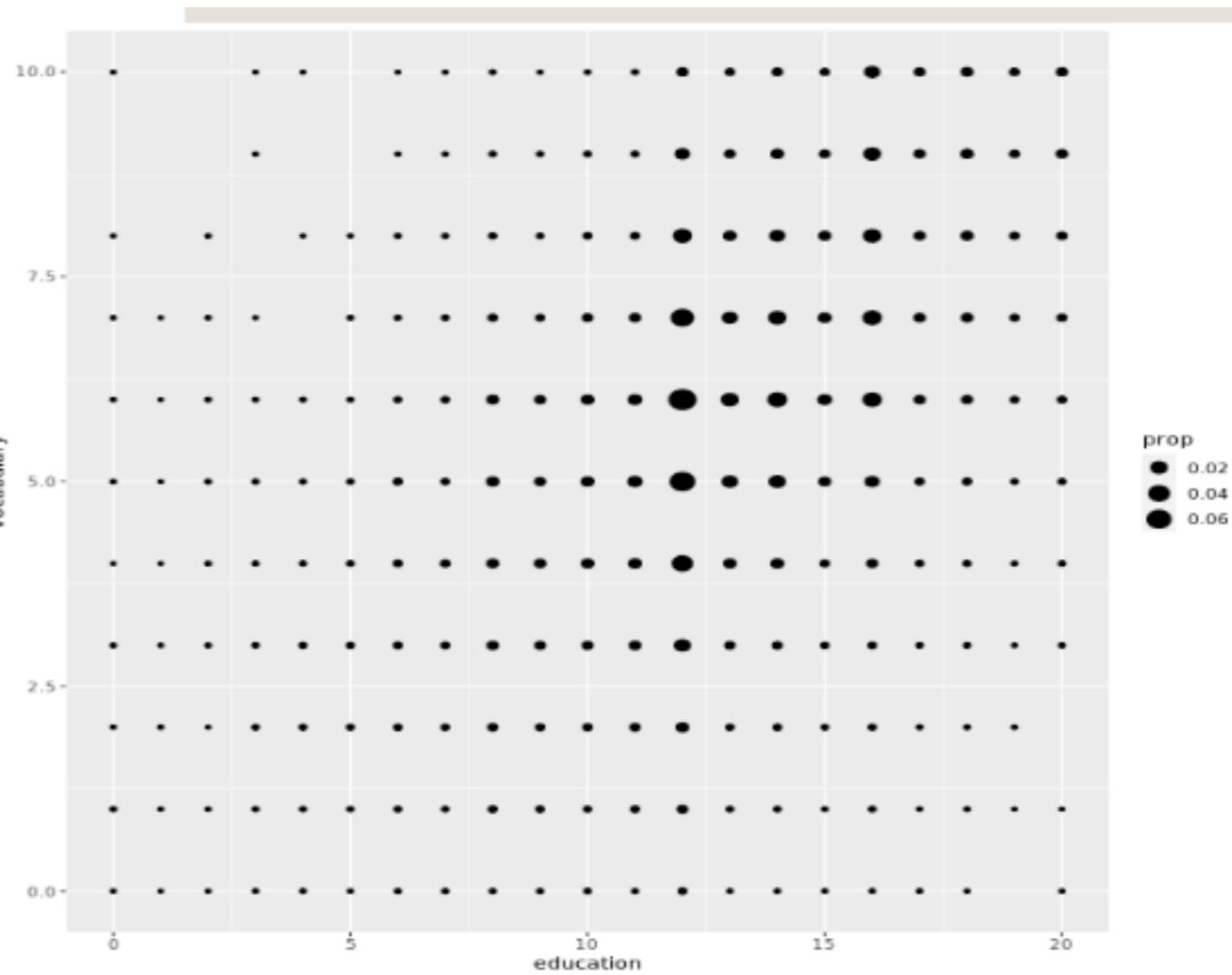
# Dealing with heteroscedasticity

```
library(AER)  
data(Journals)  
  
p <- ggplot(Journals,  
             aes(log(price/citations),  
                  log(subs))) +  
  geom_point(alpha = 0.5) +  
  labs(...)  
  
p +  
  geom_quantile(quantiles =  
    c(0.05, 0.50, 0.95))
```

geom_	stat_
geom_count()	stat_sum()
geom_quantile()	stat_quantile()



```
# Amend the stat to use proportion sizes
ggplot(Vocab, aes(x = education, y =
vocabulary)) +
  stat_sum(aes(size = ..prop..))
```



# Calculating statistics

```
set.seed(123)
xx <- rnorm(100)

# Hmisc
library(Hmisc)
smean.sdl(xx, mult = 1)
```

Mean	Lower	Upper
0.09040591	-0.82240997	1.00322179

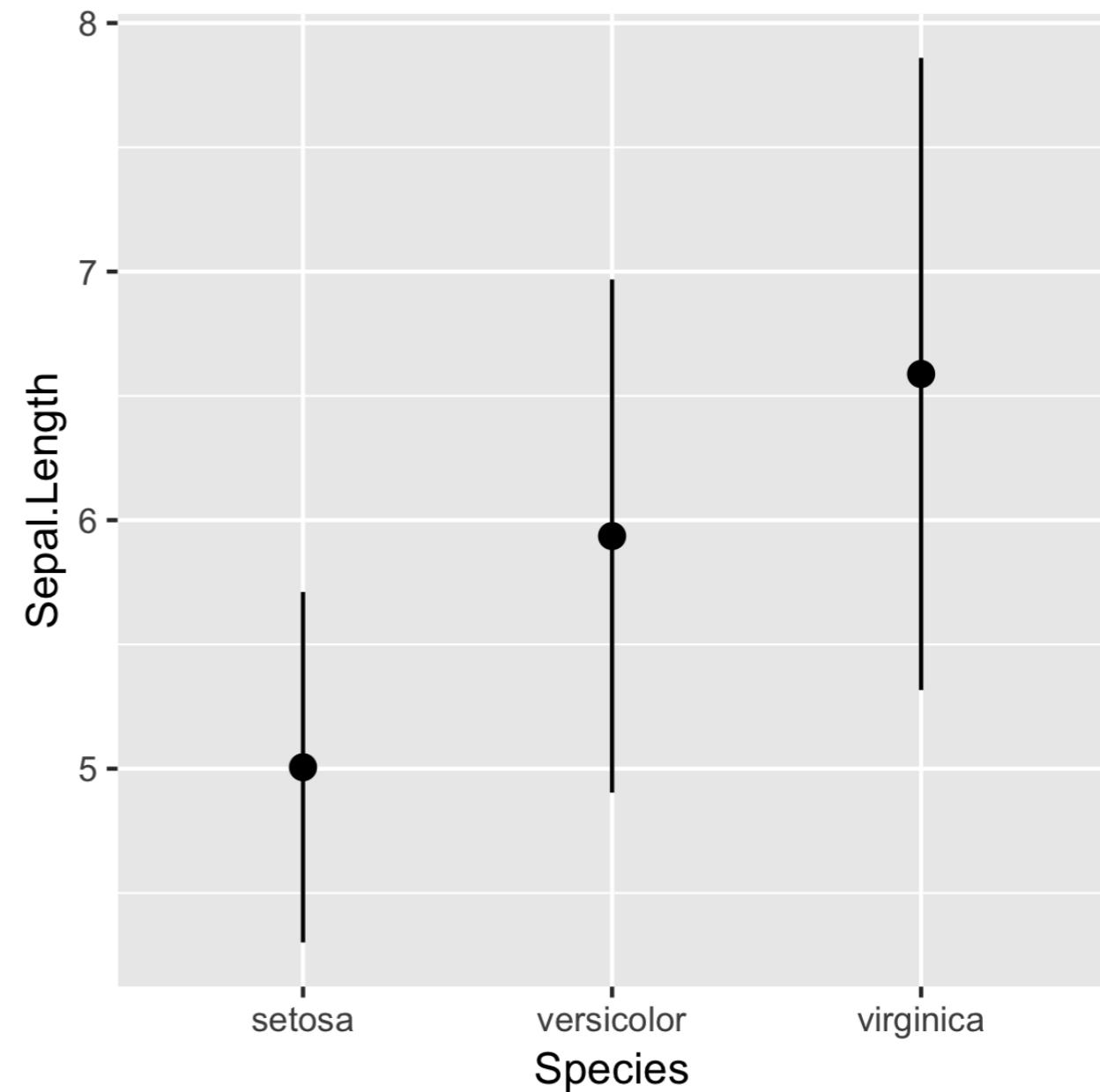
# stat\_summary()

```
# ggplot2  
mean_sdl(xx, mult = 1)
```

```
      y      ymin      ymax A data.frame with this columns  
1 0.09040591 -0.82241 1.003222
```

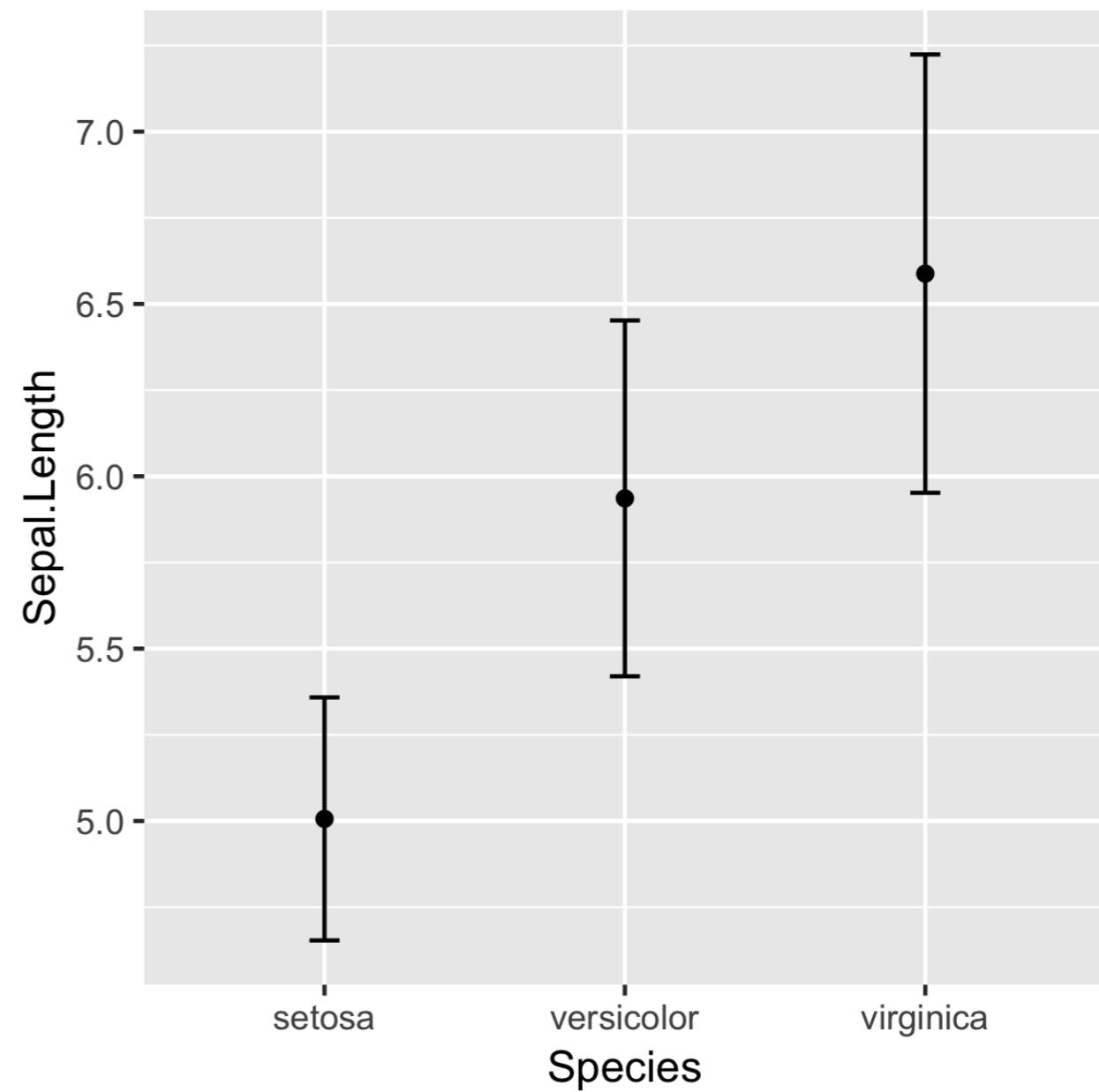
```
ggplot(iris, aes(x = Species,  
                  y = Sepal.Length)) +  
  stat_summary(fun.data = mean_sdl,  
               fun.args = list(mult = 1))
```

- Uses `geom_pointrange()` by default



# stat\_summary()

```
ggplot(iris, aes(x = Species,  
                 y = Sepal.Length)) +  
  stat_summary(fun.y = mean,  
              geom = "point") +  
  stat_summary(fun.data = mean_sdl,  
              fun.args = list(mult = 1),  
              geom = "errorbar",  
              width = 0.1)
```



# 95% confidence interval

```
ERR <- qt(0.975, length(xx) - 1) * (sd(xx) / sqrt(length(xx)))  
mean(xx)
```

```
0.09040591
```

```
mean(xx) + (ERR * c(-1, 1)) # 95% CI
```

```
-0.09071657 0.27152838
```

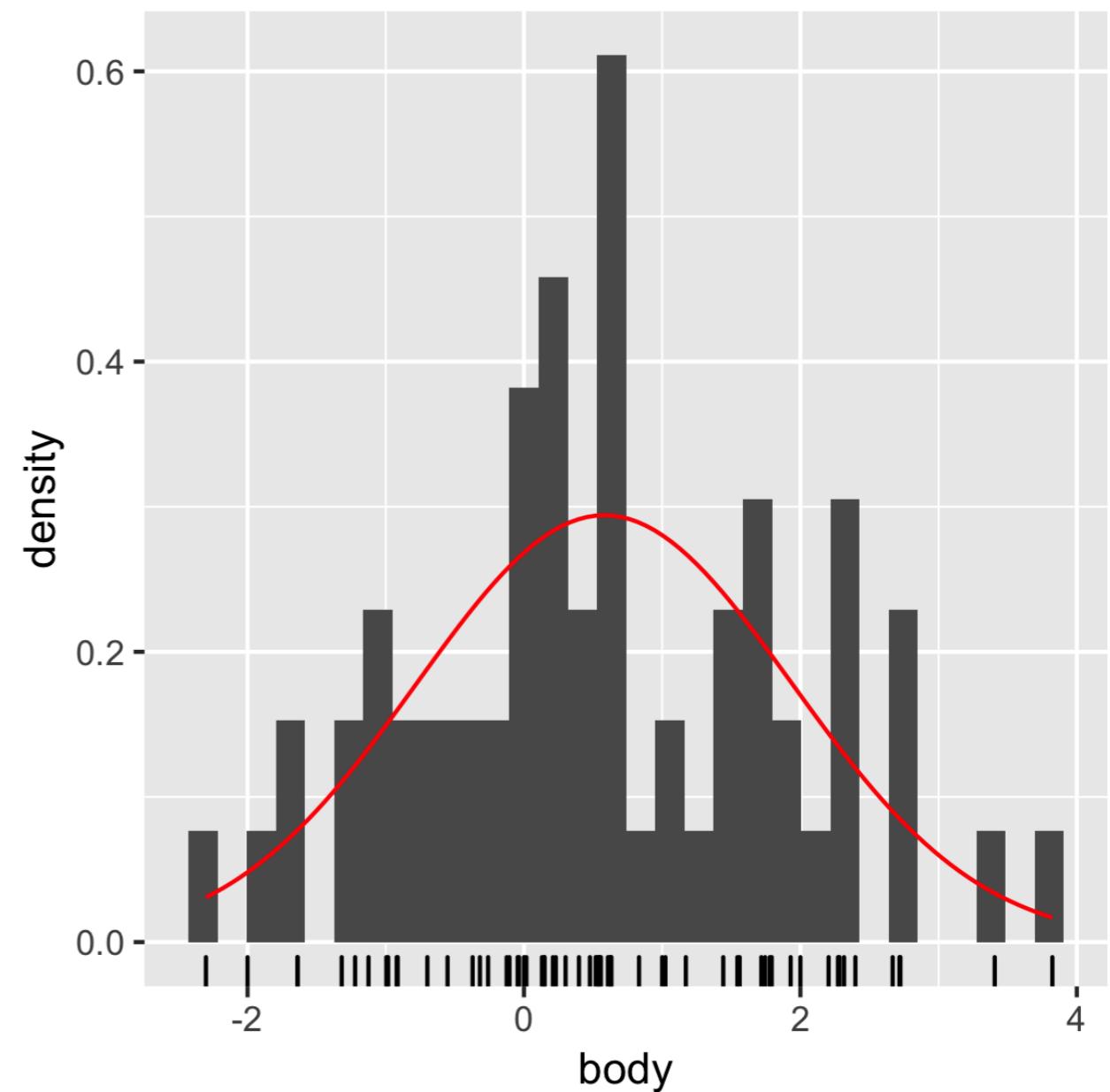
```
mean_cl_normal(xx)
```

y	ymin	ymax
0.09040591	-0.09071657	0.2715284

# Normal distribution

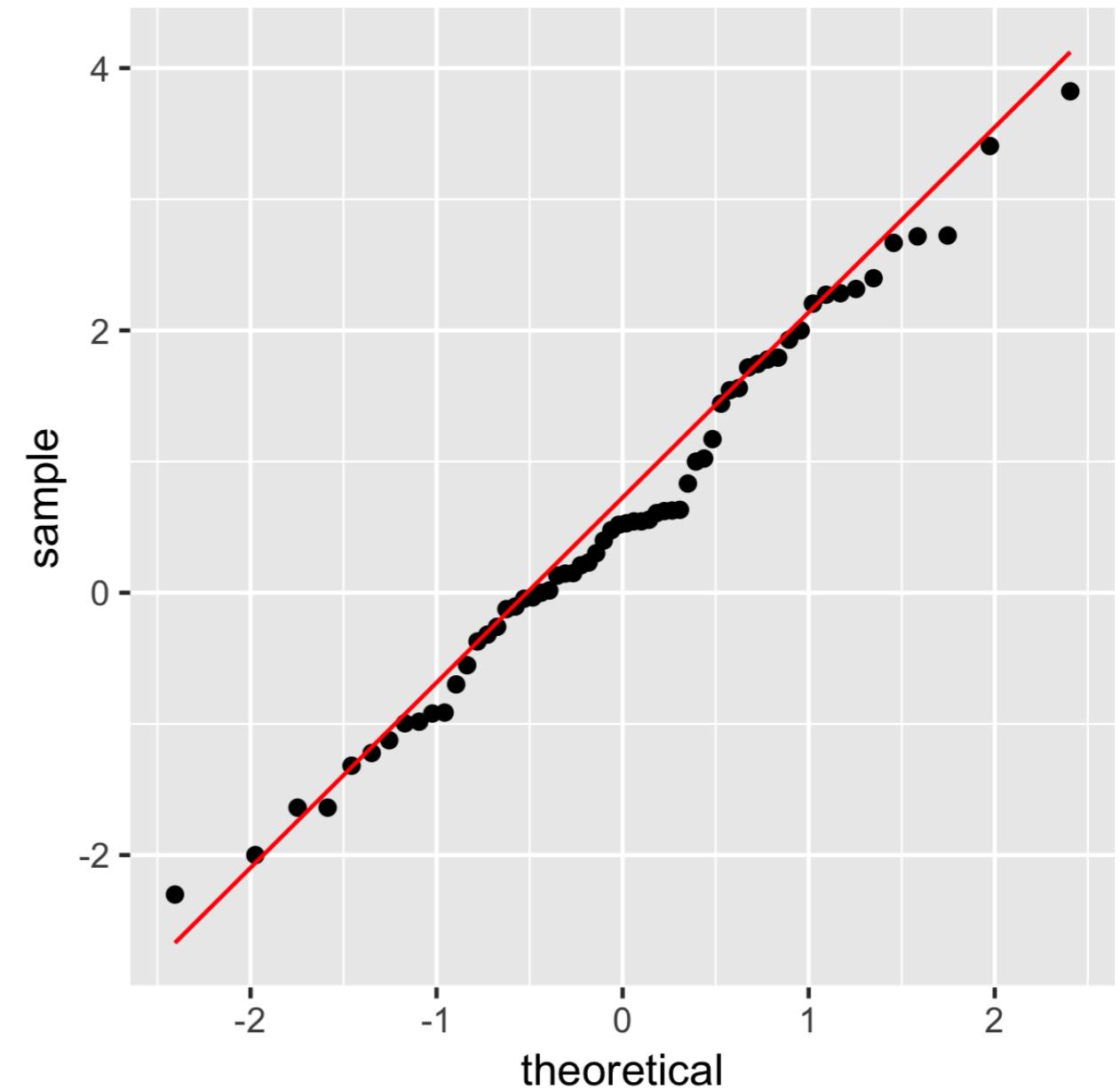
```
mam.new <- data.frame(body = log10(mammals$body))

ggplot(mam.new, aes(x = body)) +
  geom_histogram(aes( y = ..density..)) +
  geom_rug() +
  stat_function(fun = dnorm, color = "red",
                args = list(mean = mean(mam.new$body),
                            sd = sd(mam.new$body)))
```



# QQ plot

```
ggplot(mam.new, aes(sample = body)) +  
  stat_qq() +  
  geom_qq_line(col = "red")
```



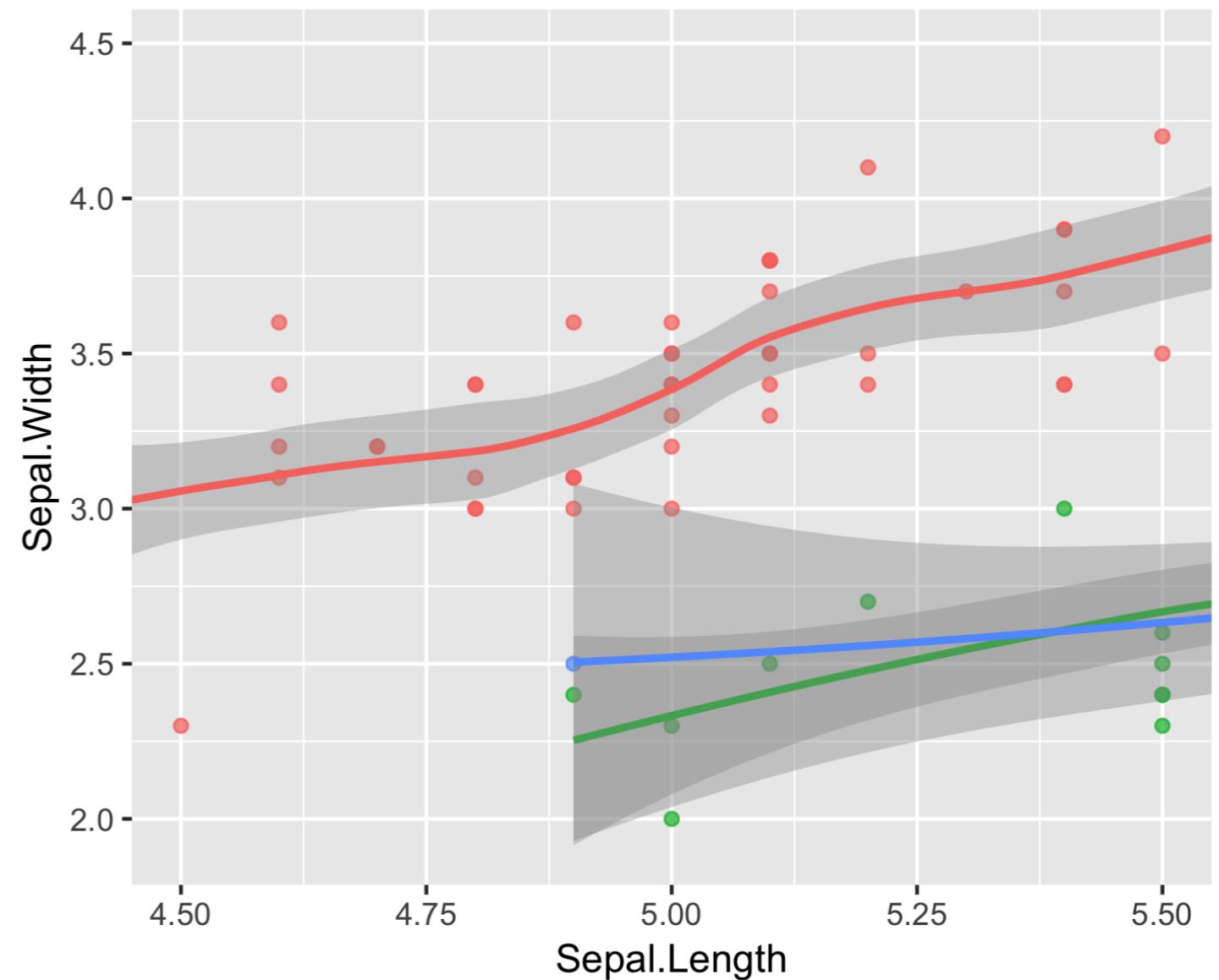
# Plot counts to overcome over-plotting

	Cause of Over-plotting	Solutions	Here...
1.	Large datasets	Alpha-blending, hollow circles, point size	<code>alpha = 0.5, shape = ":"</code> <code>shape = 16</code>
2.	Aligned values on a single axis	As above, plus change position	<code>position_jitter</code> <code>position_jitterdodge</code>
3.	Low-precision data	Position: jitter <code>geom_jitter(alpha = 0.5, width = 0.1)</code>	<code>geom_count()</code>
4.	Integer data	Position: jitter <code>geom_jitter(alpha = 0.2, shape = 1)</code>	<code>geom_count()</code>

geom_	stat_
<code>geom_count()</code>	<code>stat_sum()</code>

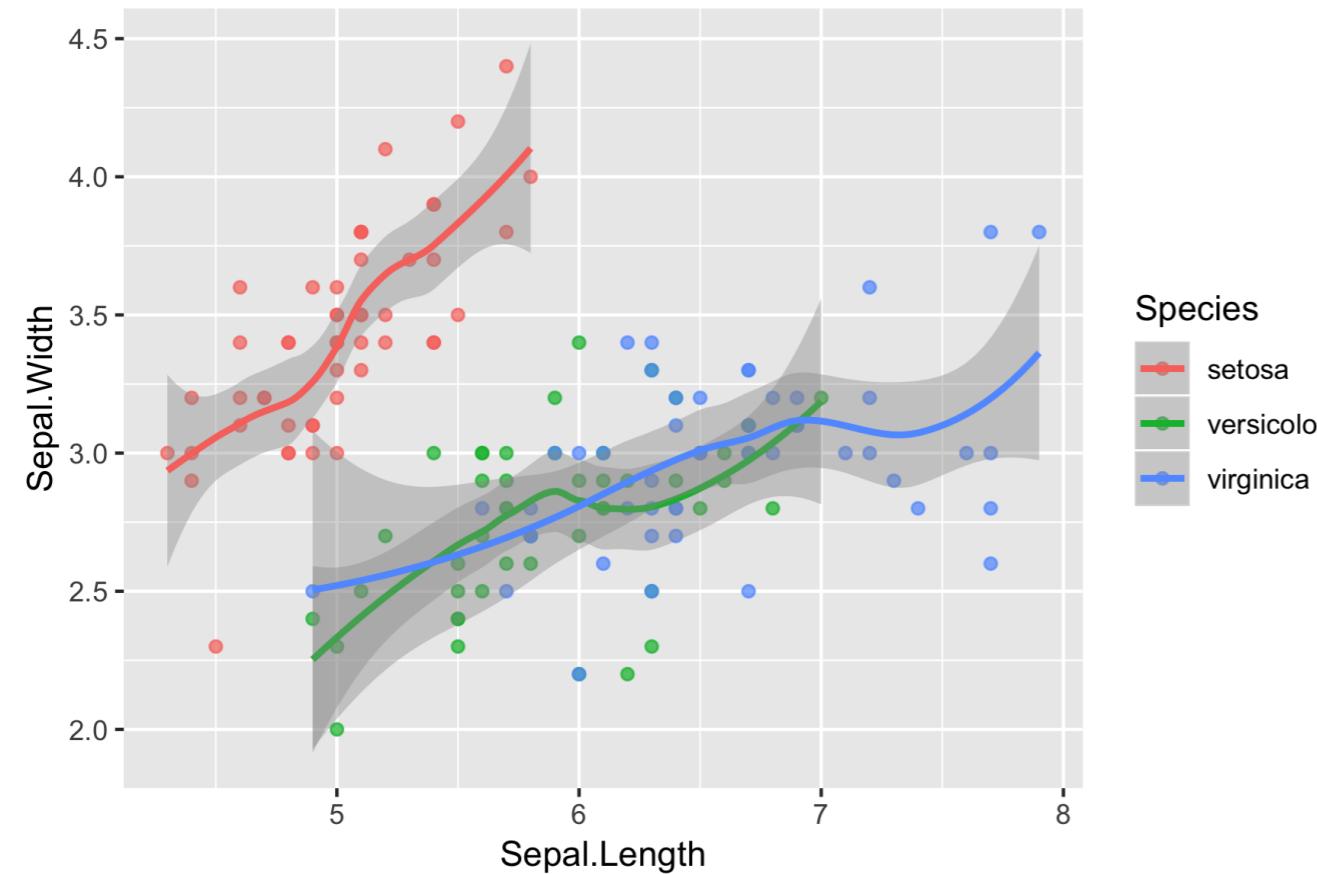
# coord\_cartesian()

```
iris.smooth +  
  coord_cartesian(xlim = c(4.5, 5.5))
```

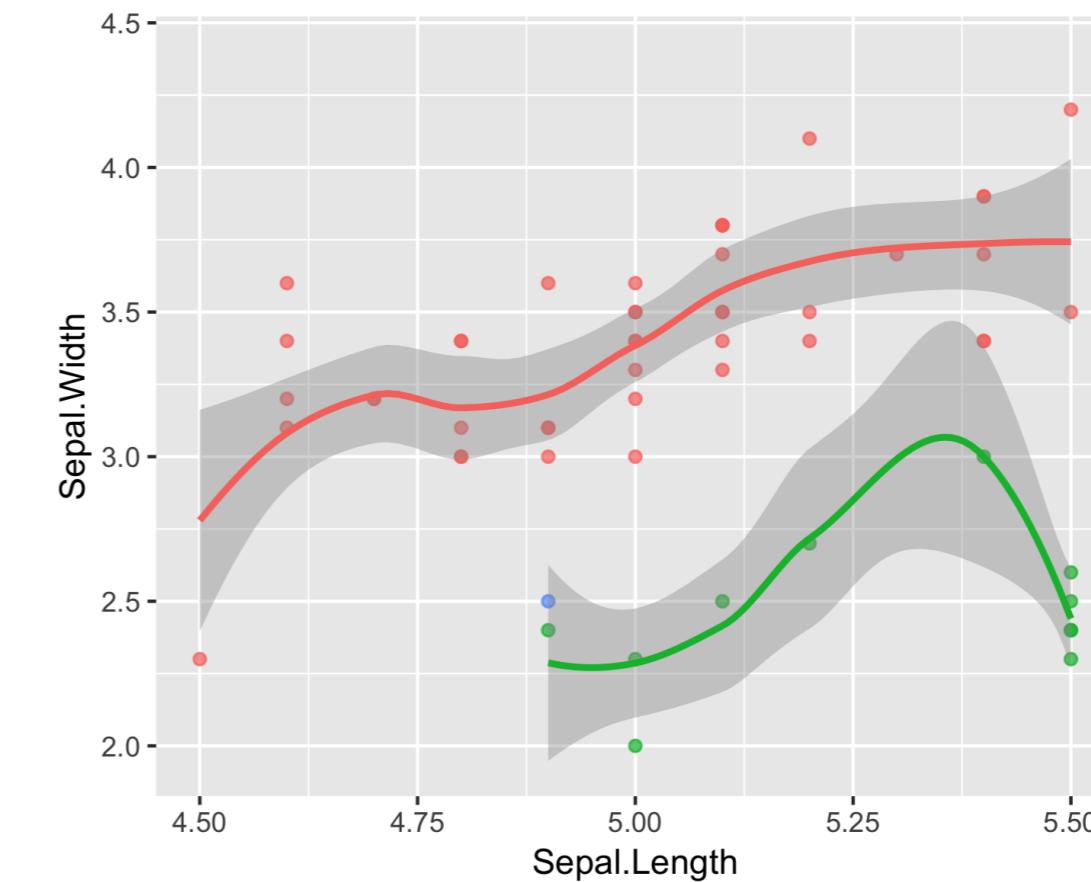


# scale\_x\_continuous()

Original plot



Zoom in with `scale_x_continuous()`

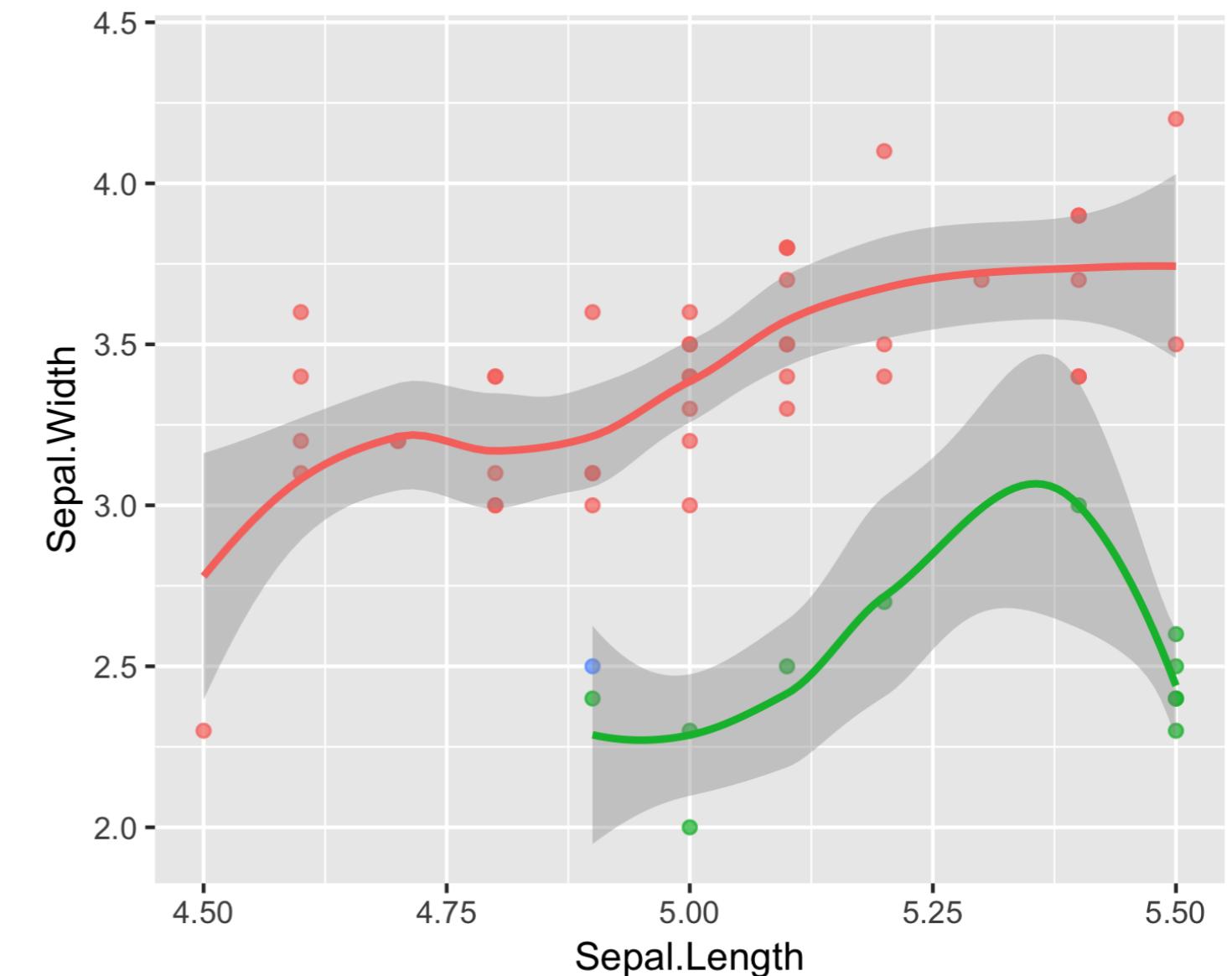


Part of original data is filtered out!

# xlim()

```
iris.smooth +  
  xlim(c(4.5, 5.5))
```

Removed 95 rows containing non-finite values  
(stat\_smooth).  
Removed 95 rows containing missing values  
(geom\_point).

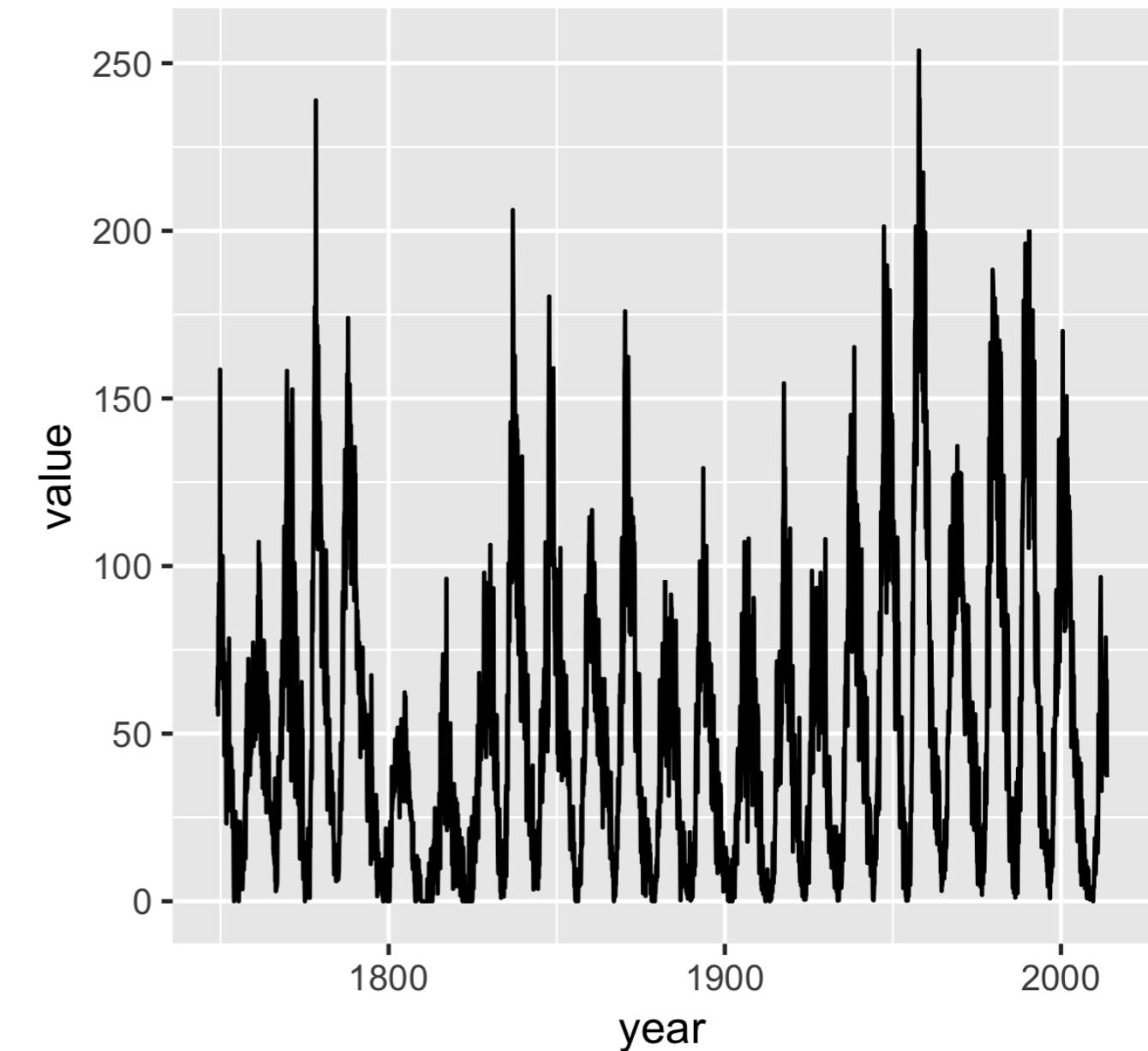


# Sunspots

```
library(zoo)
sunspots.m <- data.frame(
  year = index(sunspot.month),
  value = reshape2::melt(sunspot.month)$value)
)

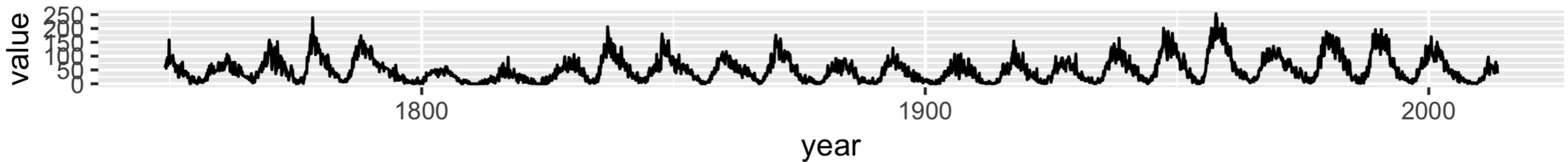
ggplot(sunspots.m, aes(x = year, y = value)) +
  geom_line() +
  coord_fixed() # default to 1:1 aspect ratio
```

- Typically use 1:1 if data is on the same scale



# Sunspots

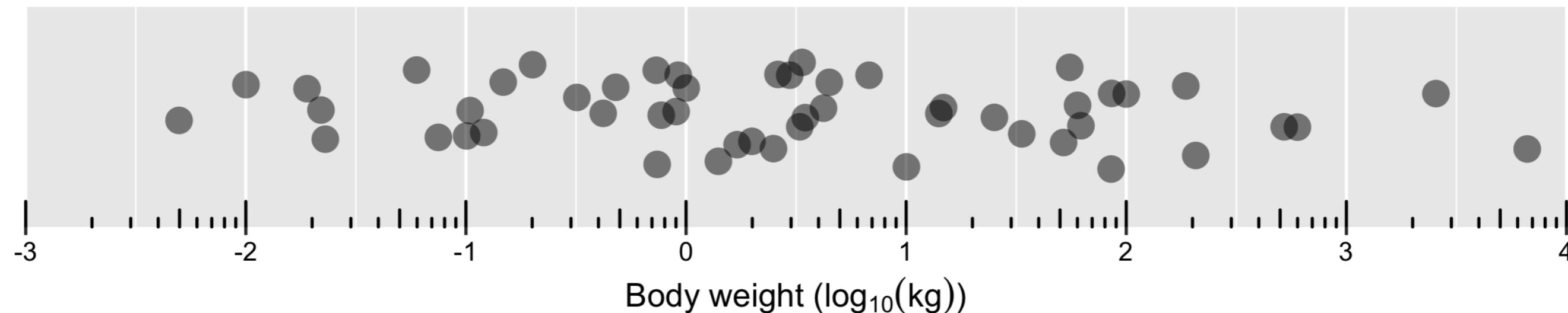
```
ggplot(sunspots.m, aes(x = year, y = value)) +  
  geom_line() +  
  coord_fixed(0.055)
```



# Add logtick annotation

```
ggplot(msleep, aes(log10(bodywt), y = 1)) +  
  geom_jitter() +  
  scale_x_continuous(limits = c(-3, 4),  
                      breaks = -3:4) +  
  annotation_logticks(sides = "b")
```

log10 trans of raw values

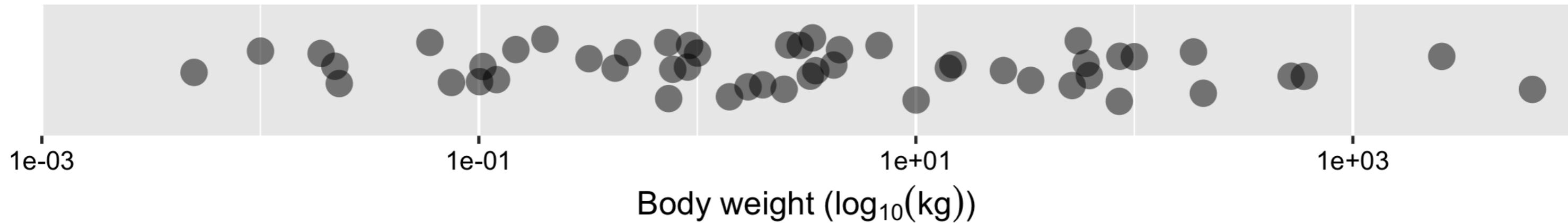


# Use `scale_x_log10()`

```
ggplot(msleep, aes(bodywt, y = 1)) +  
  geom_jitter() +  
  scale_x_log10(limits = c(1e-03, 1e+04))
```

It is equivalent to transforming our actual dataset before getting to ggplot2

log10 trans using `scale_x_log10()`

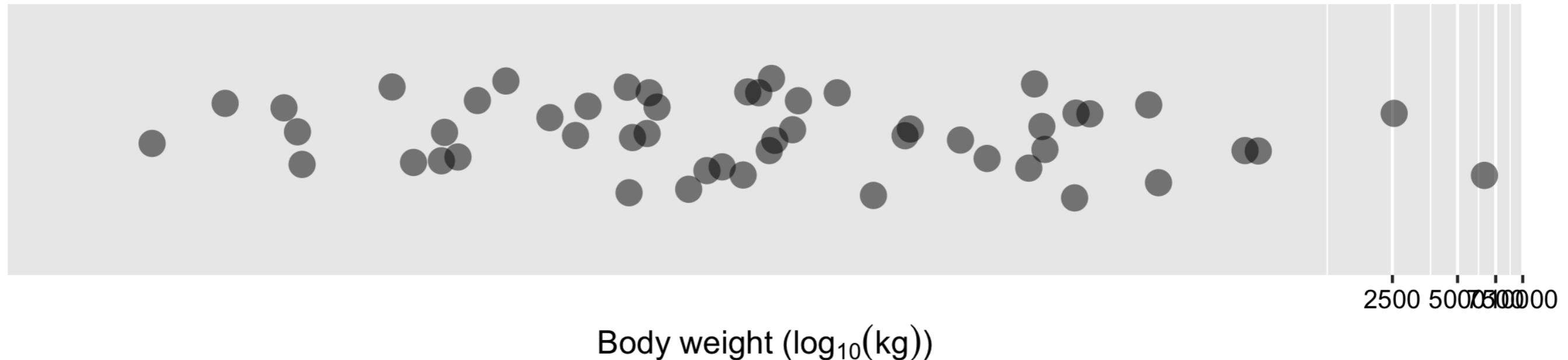


# Use `coord_trans()`

```
ggplot(msleep, aes(bodywt, y = 1)) +  
  geom_jitter() +  
  coord_trans(x = "log10")
```

It transforms the data after statistics have been calculated, a lm doesn't work with this method.

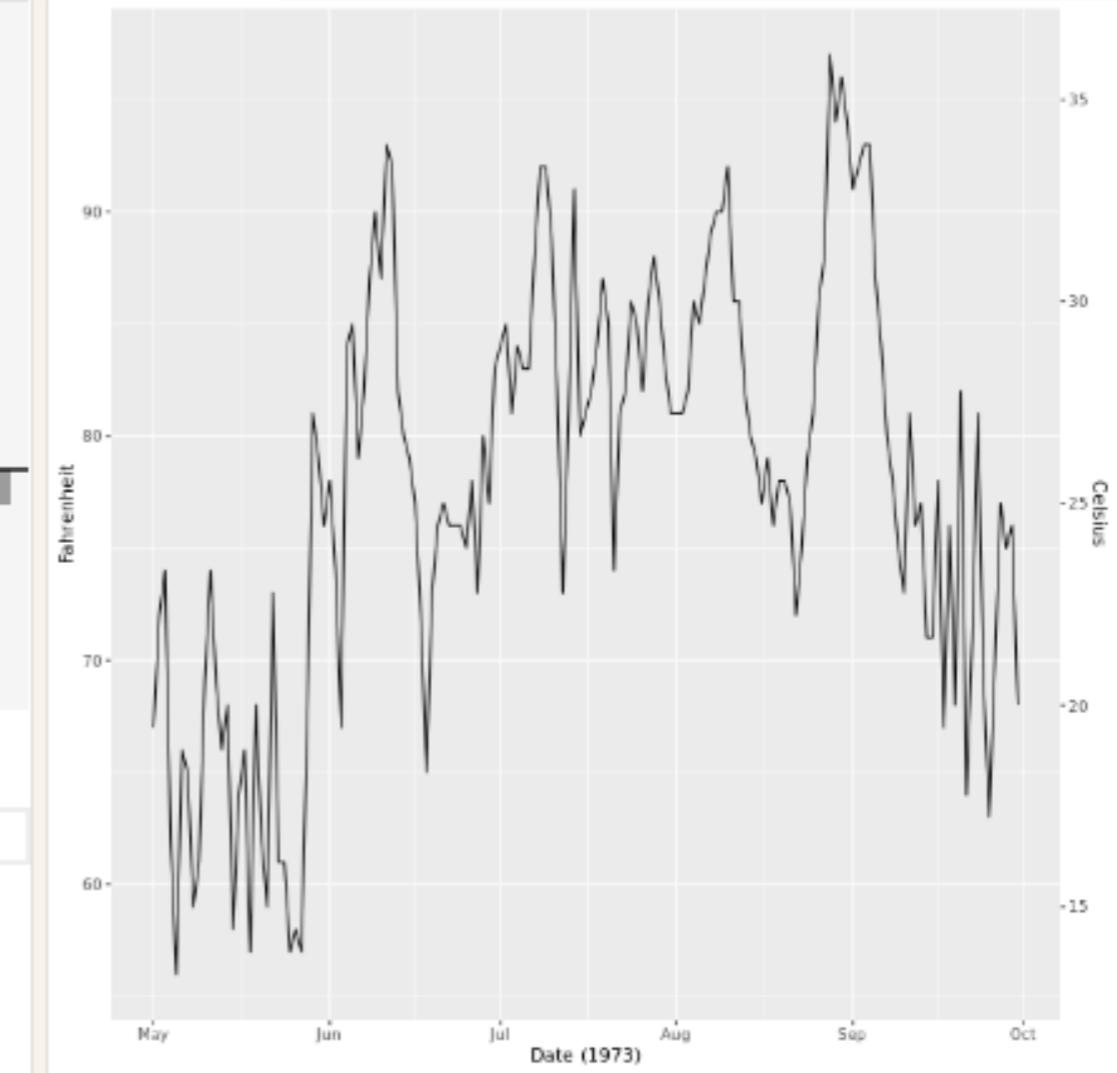
log10 trans using `coord_trans()`



# Double Axis with a vector of values

```
# From previous step
y_breaks <- c(59, 68, 77, 86, 95, 104)
y_labels <- (y_breaks - 32) * 5 / 9
secondary_y_axis <- sec_axis(
  trans = identity,
  name = "Celsius",
  breaks = y_breaks,
  labels = y_labels
)

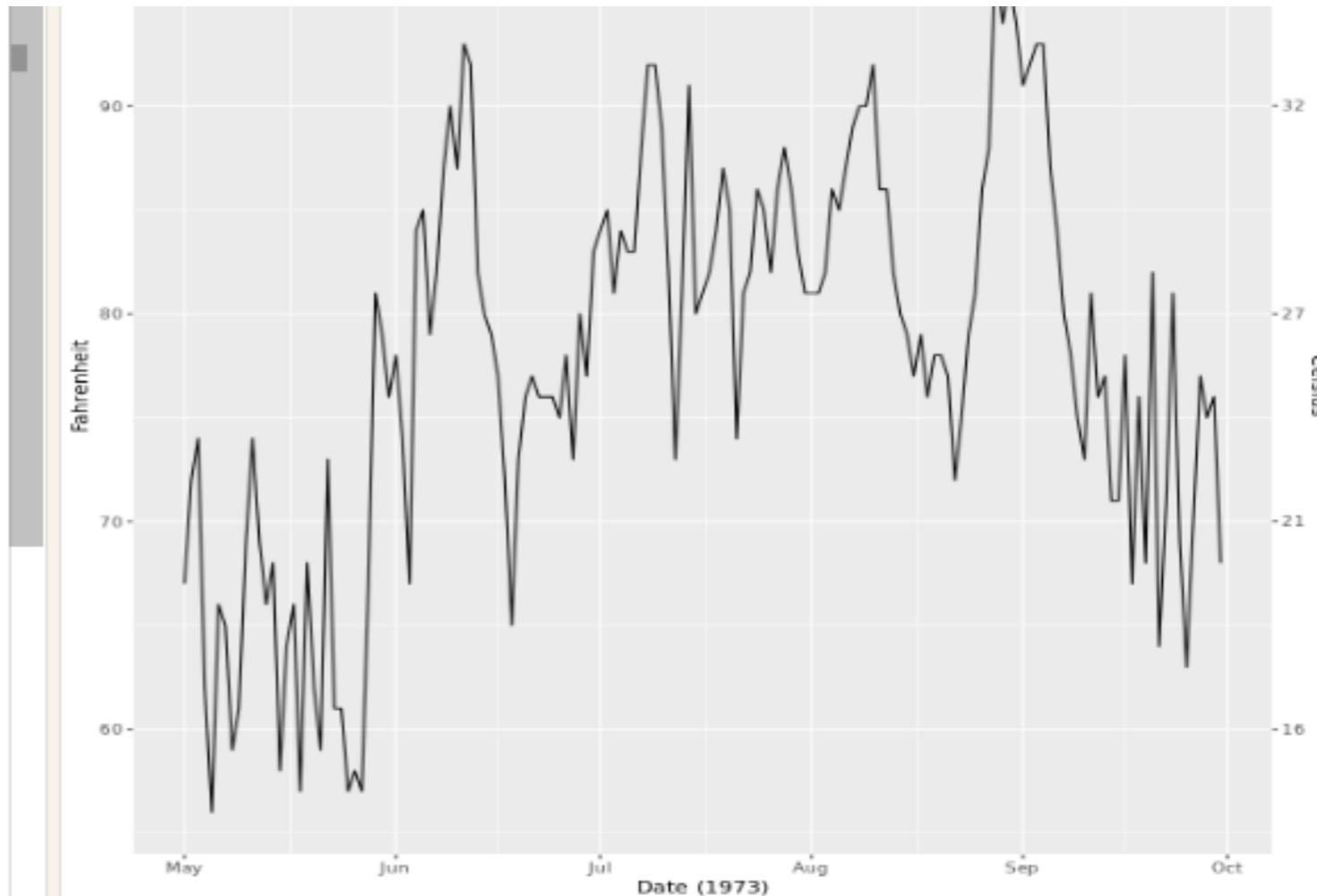
# Update the plot
ggplot(airquality, aes(Date, Temp)) +
  geom_line() +
  # Add the secondary y-axis
  scale_y_continuous(sec.axis = secondary_y_axis) +
  labs(x = "Date (1973)", y = "Fahrenheit")
```



Double axes are most useful when you want to display the same value in two different units. log10 vs exponent values, count vs proportions, physical magnitudes in different systems.

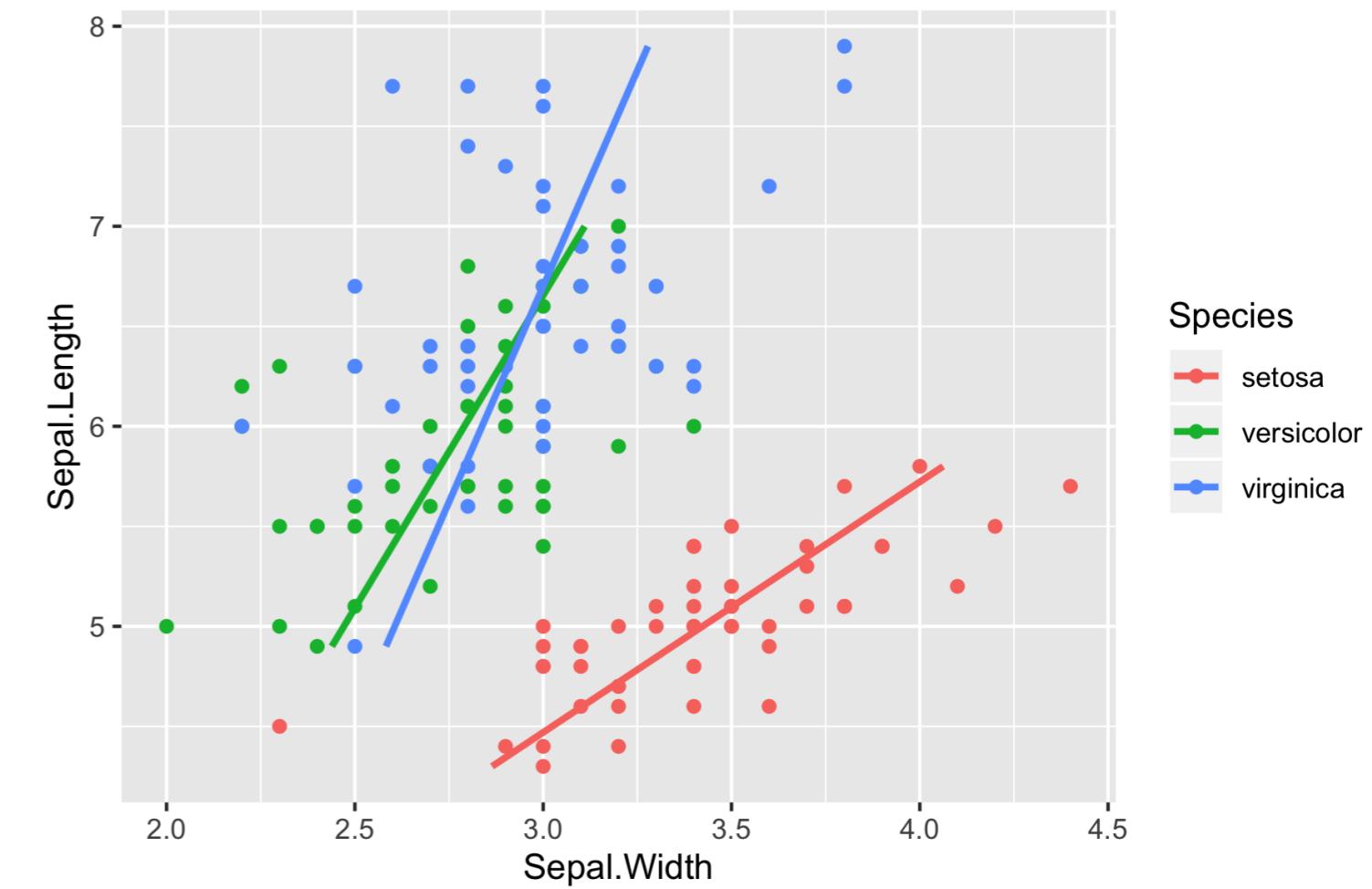
# Double Axis with a function

```
celsius_trans <- function(x){  
  round((x - 32) * 5 / 9,0)  
}  
  
secondary_y_axis <- sec_axis(  
  trans = identity,  
  name = "Celsius",  
  labels = celsius_trans  
)  
  
# Update the plot  
ggplot(airquality, aes(Date, Temp)) +  
  geom_line() +  
  # Add the secondary y-axis  
  scale_y_continuous(sec.axis =  
secondary_y_axis) +  
  labs(x = "Date (1973)", y = "Fahrenheit")
```



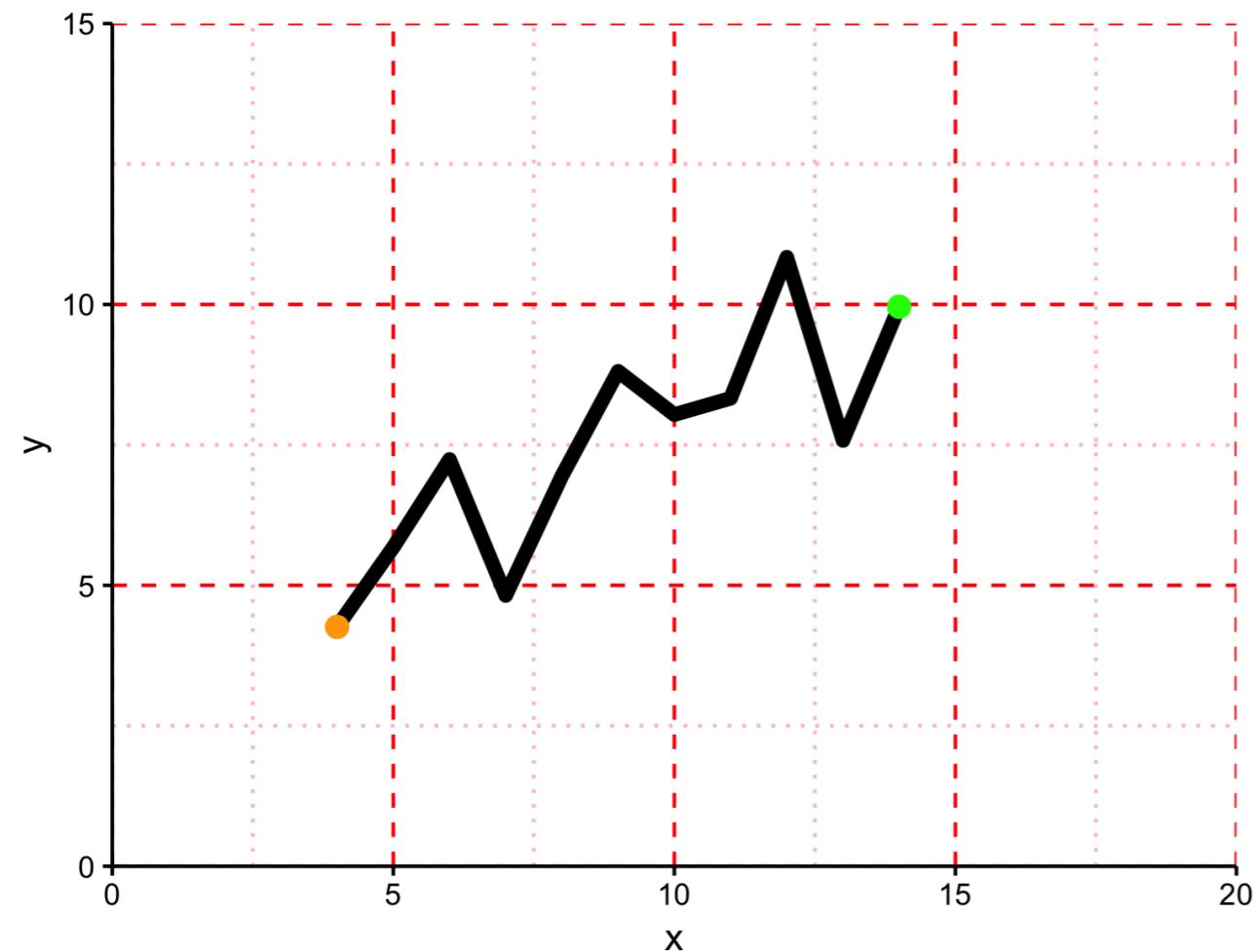
# coord\_flip()

```
ggplot(iris, aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 color = Species)) +  
  geom_point() +  
  geom_smooth(method = "lm",  
              se = FALSE) +  
  coord_flip()
```

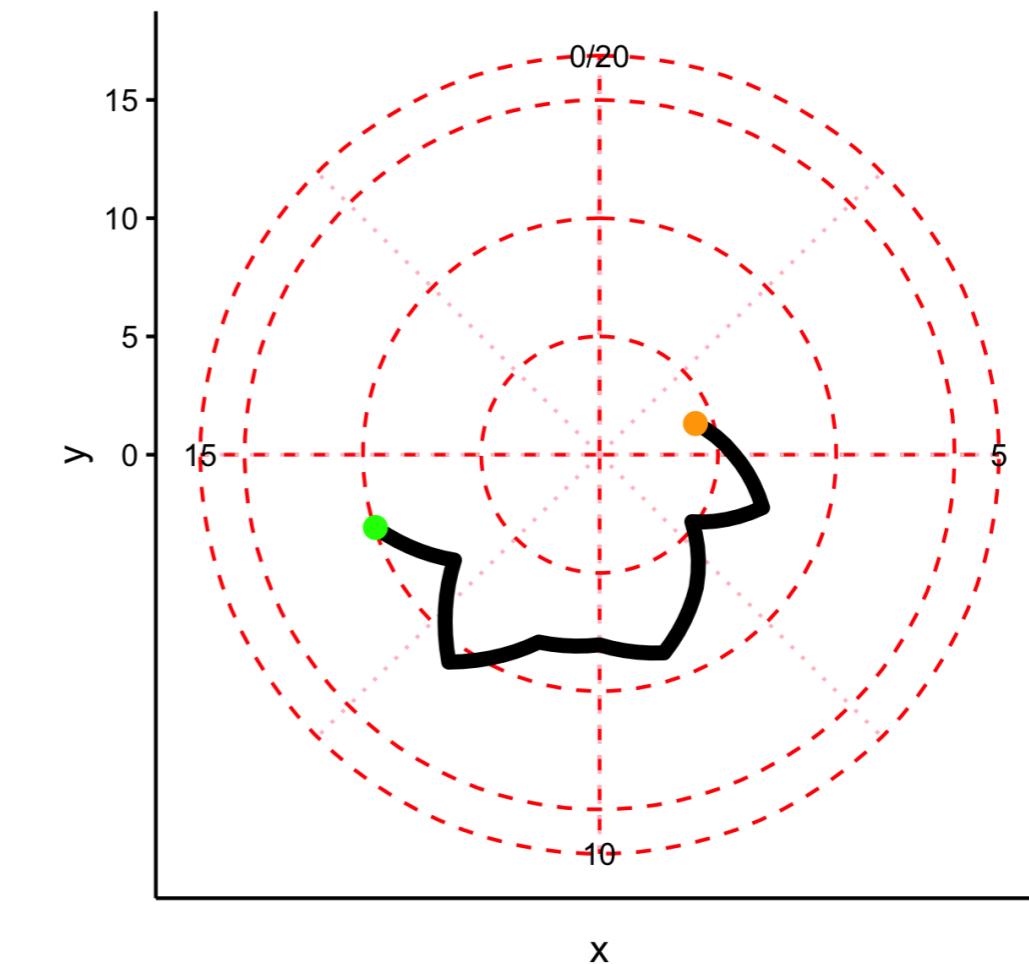


# coord\_polar()

p + coord\_fixed()

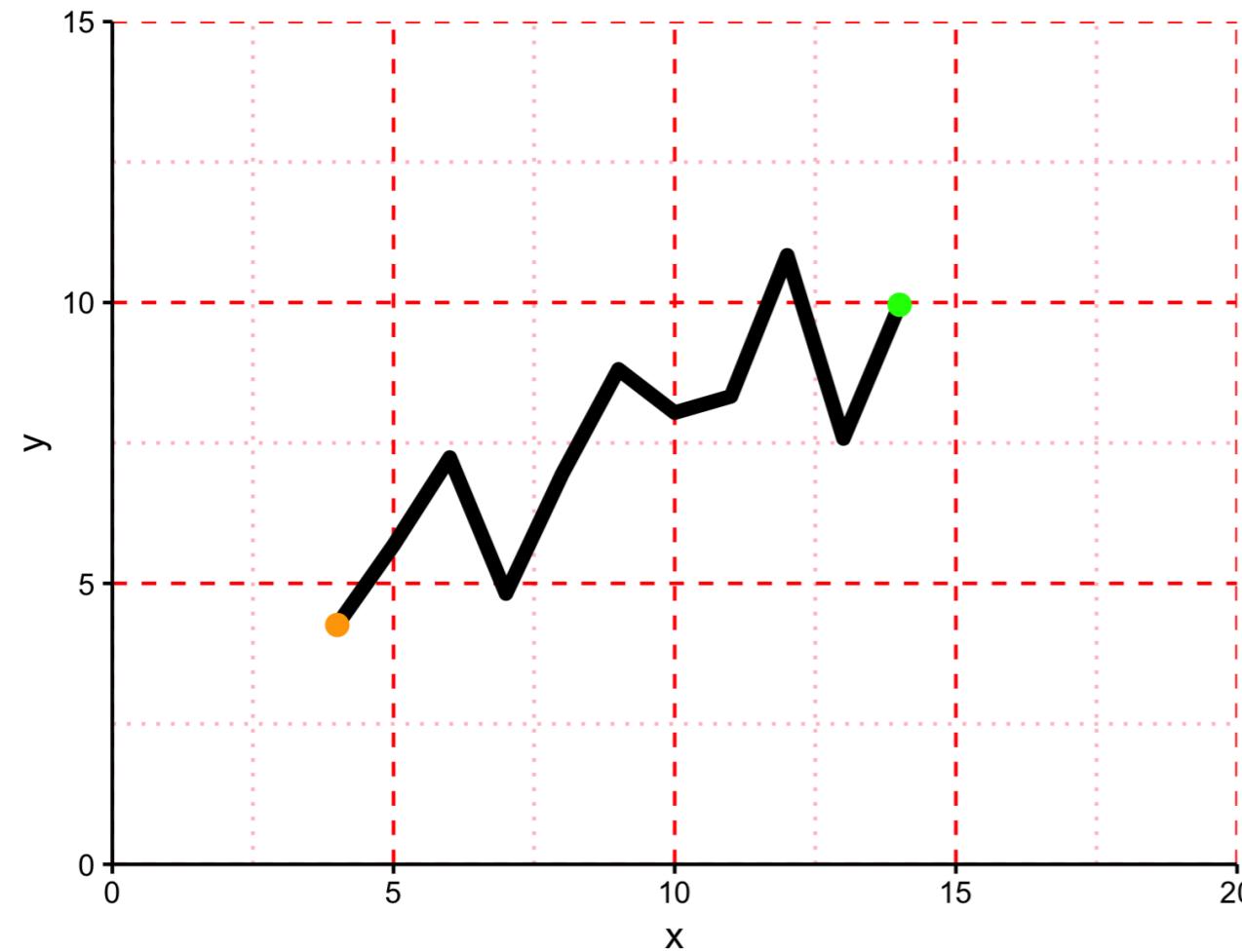


p + coord\_polar()

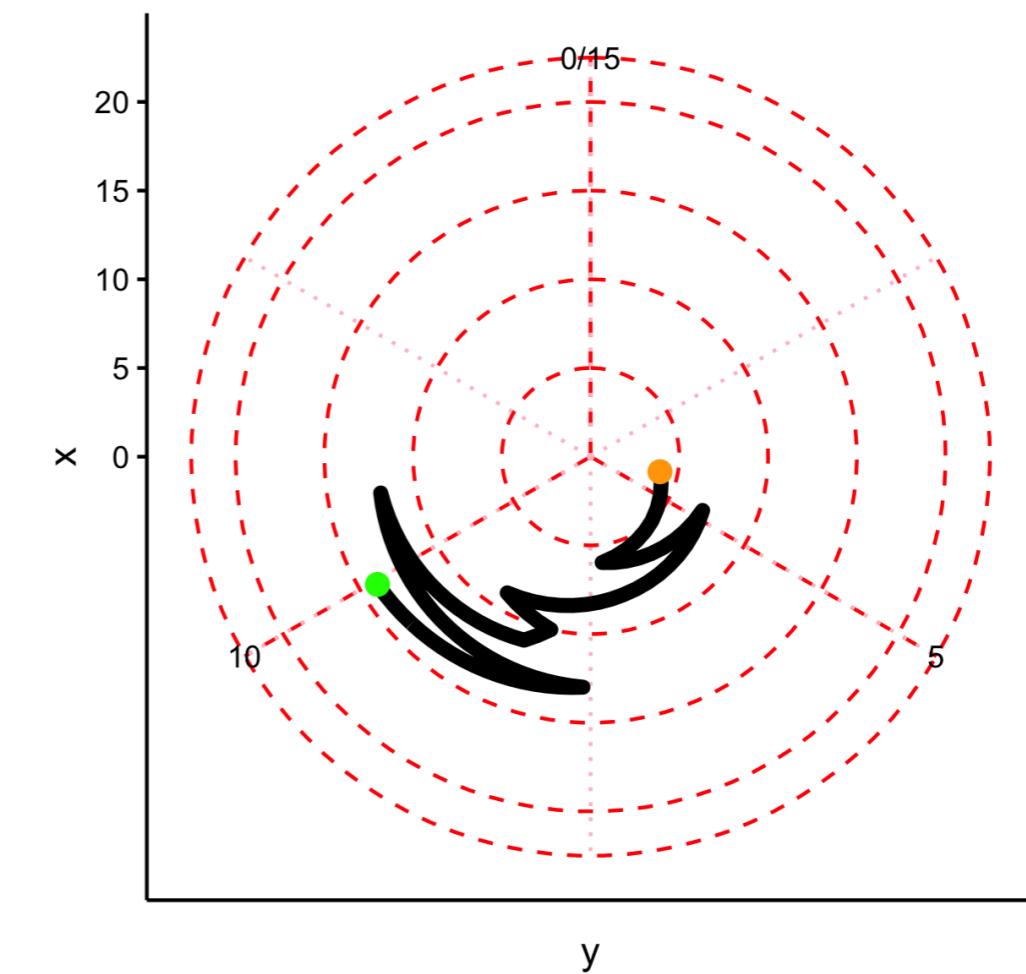


# `coord_polar(theta = "y")`

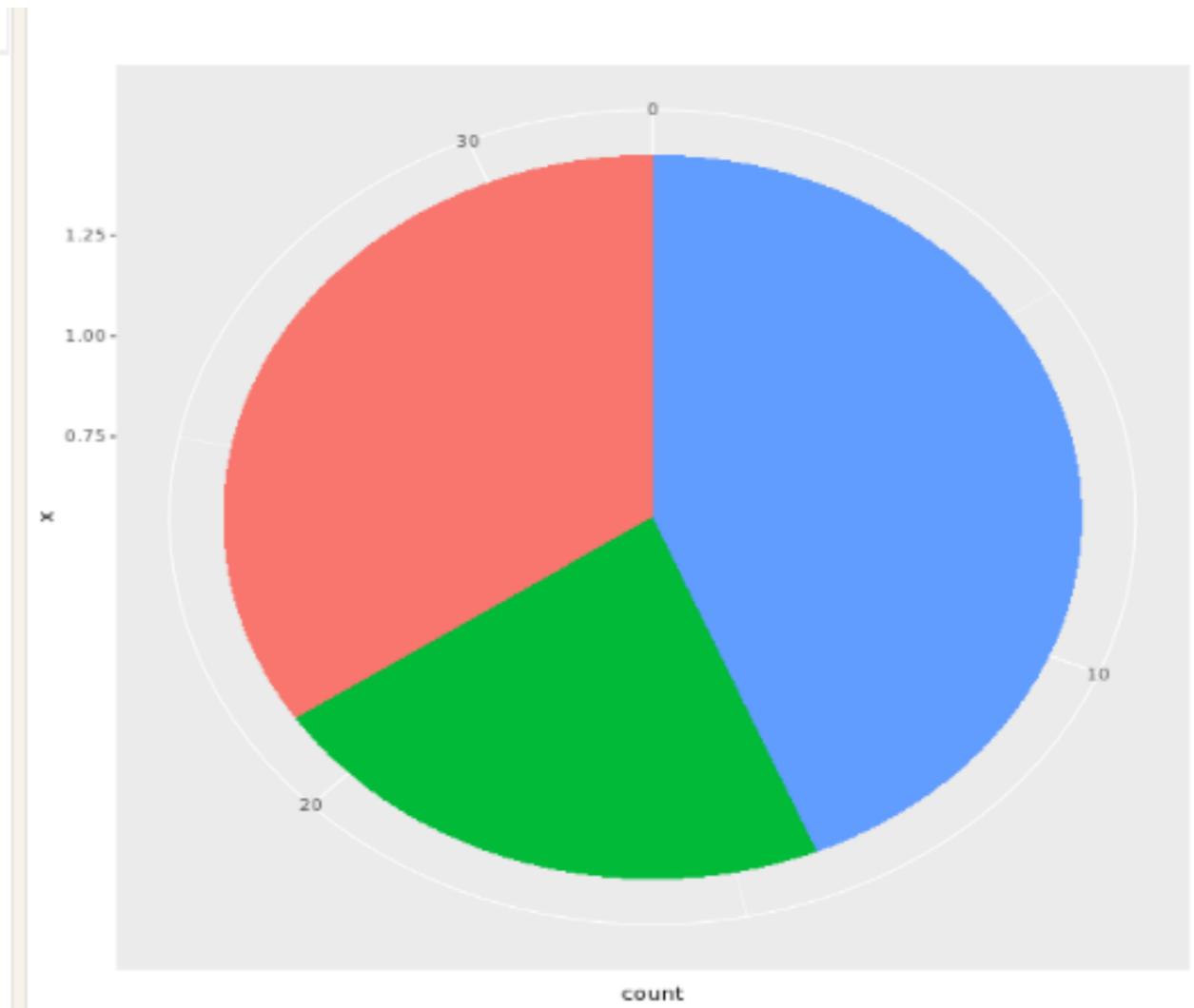
`p + coord_fixed()`



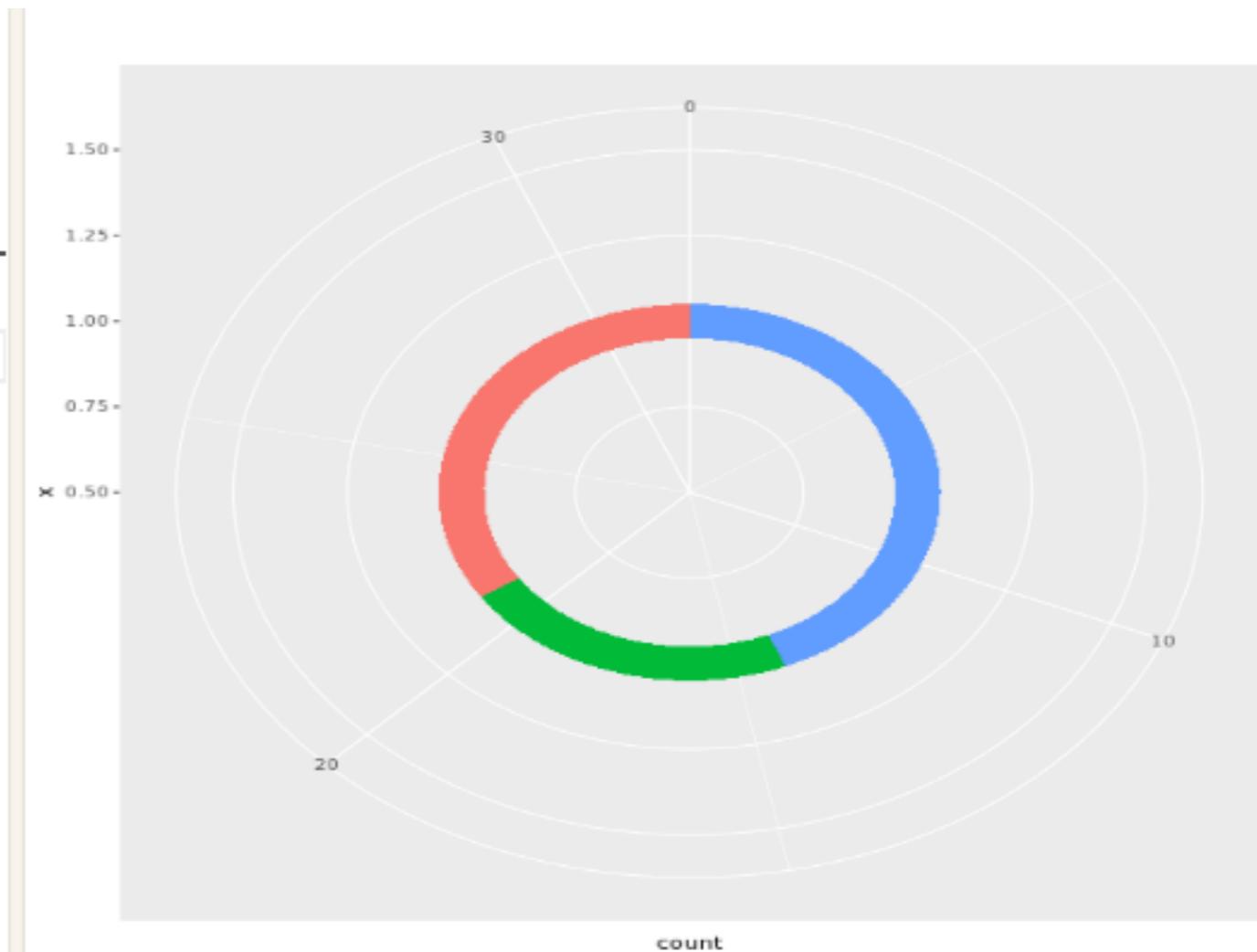
`p + coord_polar(theta = "y")`



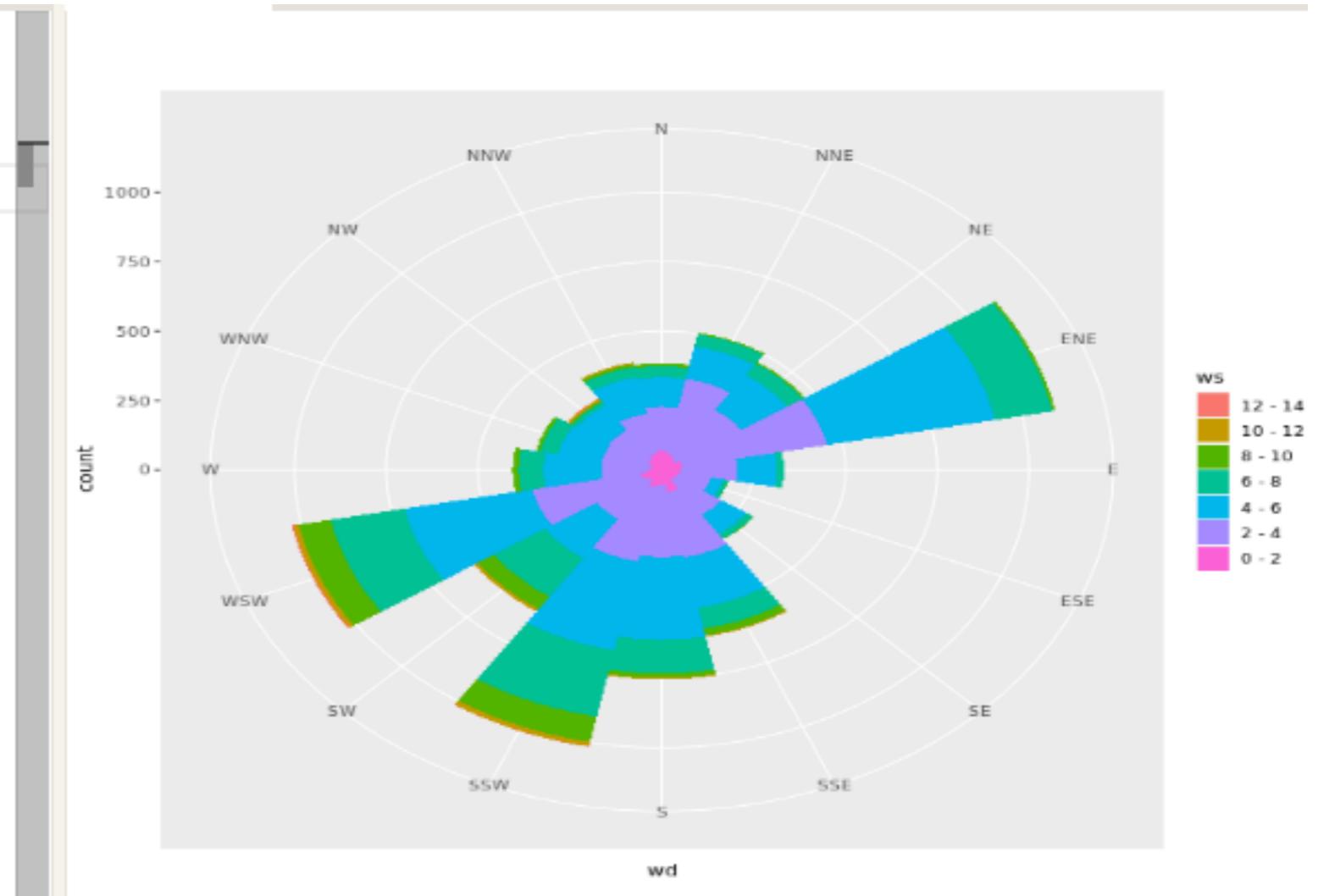
```
# Run the code, view the plot, then update  
it  
ggplot(mtcars, aes(x = 1, fill = fcyl)) +  
  geom_bar() +  
  # Add a polar coordinate system  
  coord_polar(theta = "y")
```



```
ggplot(mtcars, aes(x = 1, fill = fcyl)) +  
  # Reduce the bar width to 0.1  
  geom_bar(width = 0.1) +  
  coord_polar(theta = "y") +  
  # Add a continuous x scale from 0.5 to 1.  
 5  
  scale_x_continuous(limits = c(0.5,1.5))
```

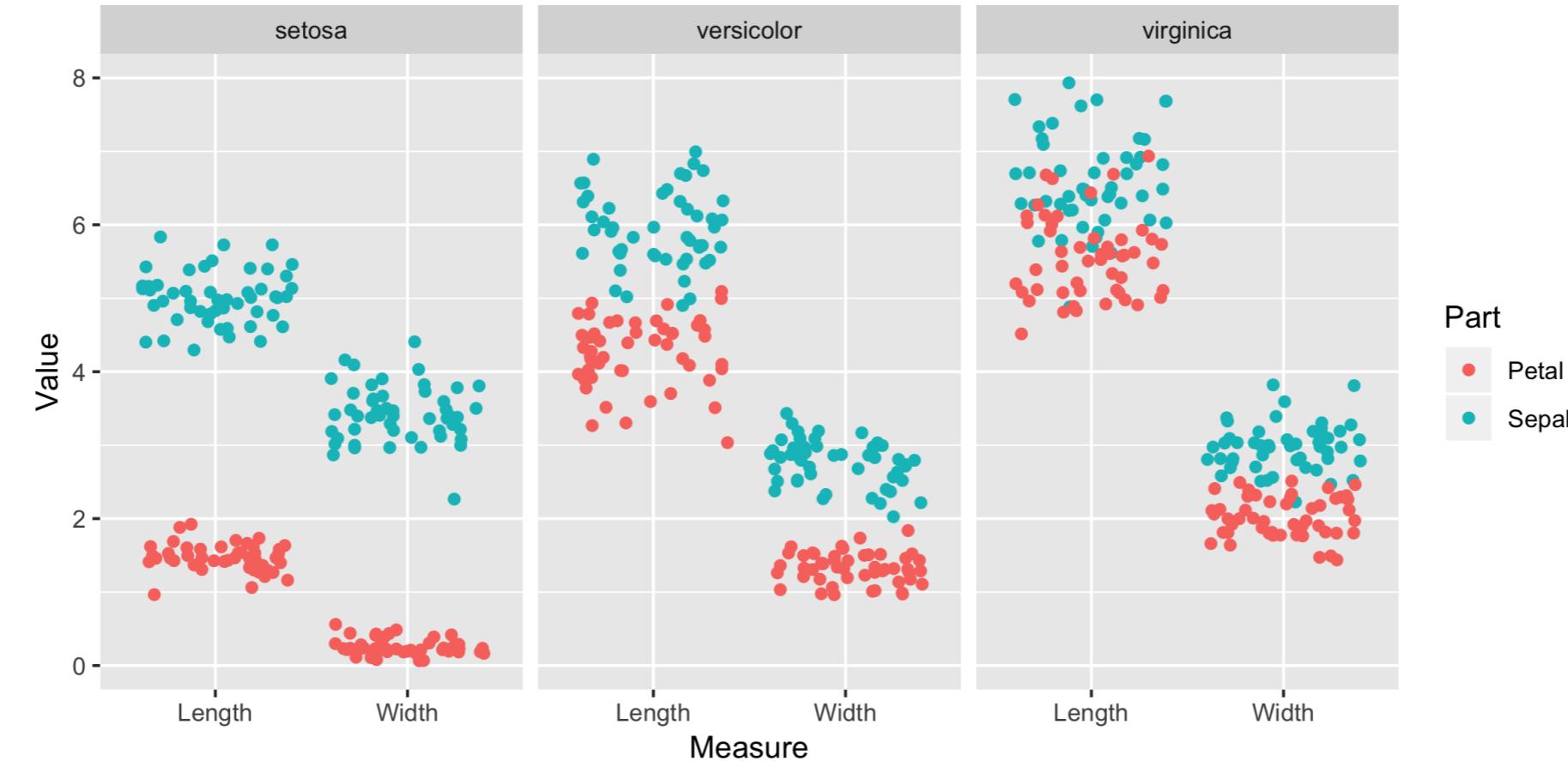


```
# Convert to polar coordinates:  
ggplot(wind, aes(wd, fill = ws)) +  
  geom_bar(width = 1) +  
  coord_polar(start = -pi/16)
```



# iris.tidy

```
ggplot(iris.tidy, aes(x = Measure, y = Value, color = Part)) +  
  geom_jitter() +  
  facet_grid(cols = vars(Species))
```



## iris.tidy faceting done wrong:

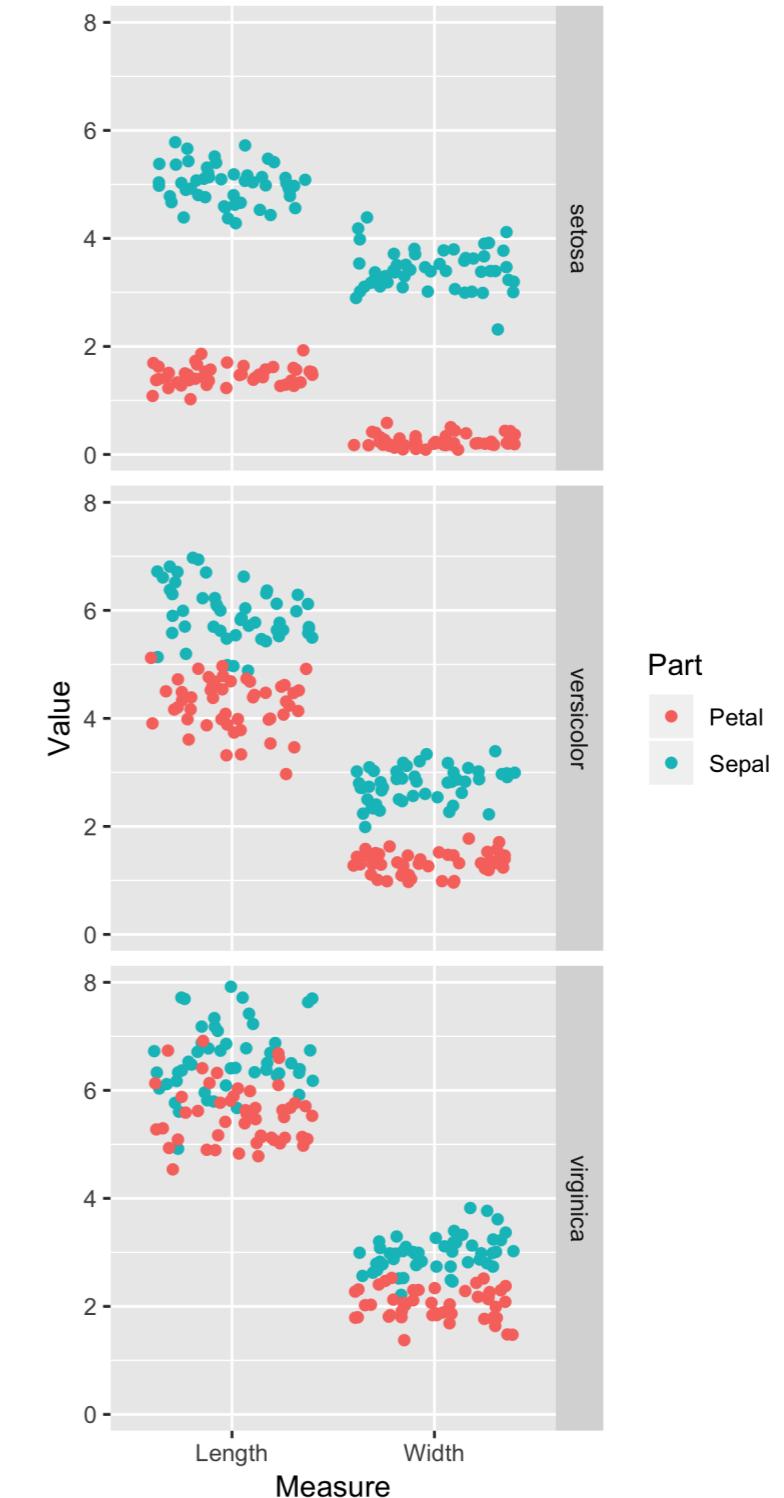
```
ggplot(iris.tidy, aes(x = Measure,  
                      y = Value,  
                      color = Part)) +  
  
  geom_jitter() +  
  
  facet_grid(rows = vars(Species))
```

| Modern notation

```
| facet_grid(rows = vars(A))  
| facet_grid(cols = vars(B))  
| facet_grid(rows = vars(A), cols = vars(B))
```

| Formula notation

```
| facet_grid(A ~ .)  
| facet_grid(. ~ B)  
| facet_grid(A ~ B)
```

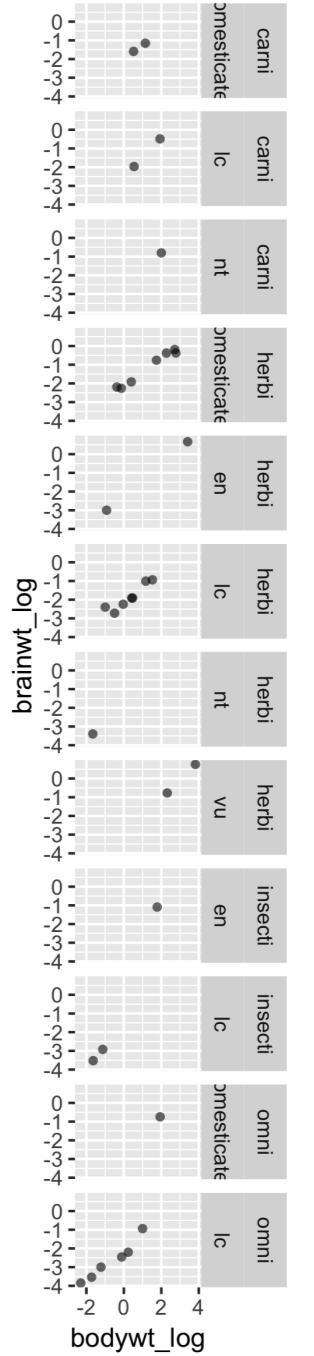
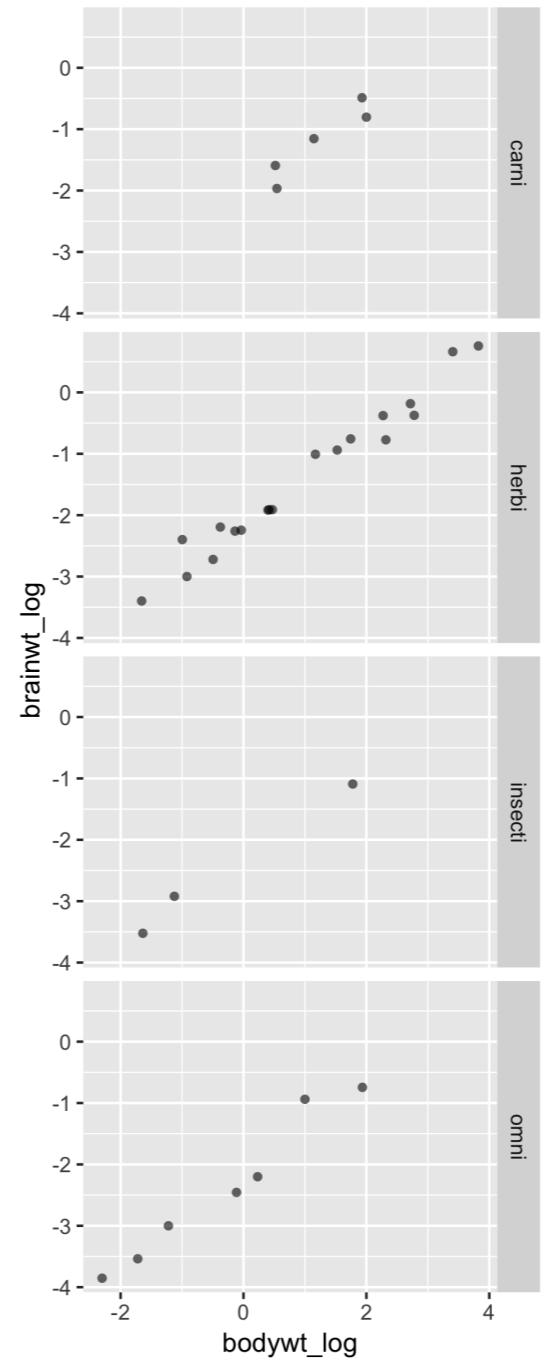


# The labeller argument

```
# Default is to label the value  
p +  
  facet_grid(rows = vars(vore),  
             labeller = label_value)
```

This option just shows the category value

```
p +  
  facet_grid(rows = vars(vore,  
                         conservation))
```

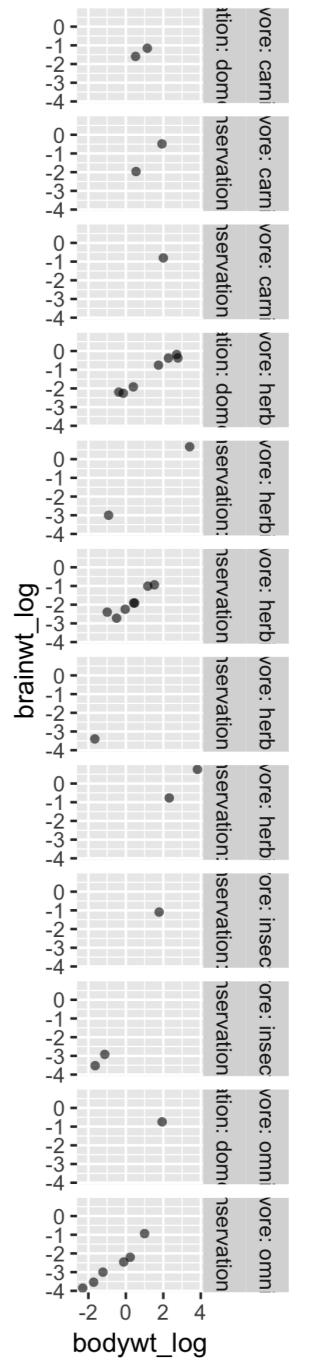
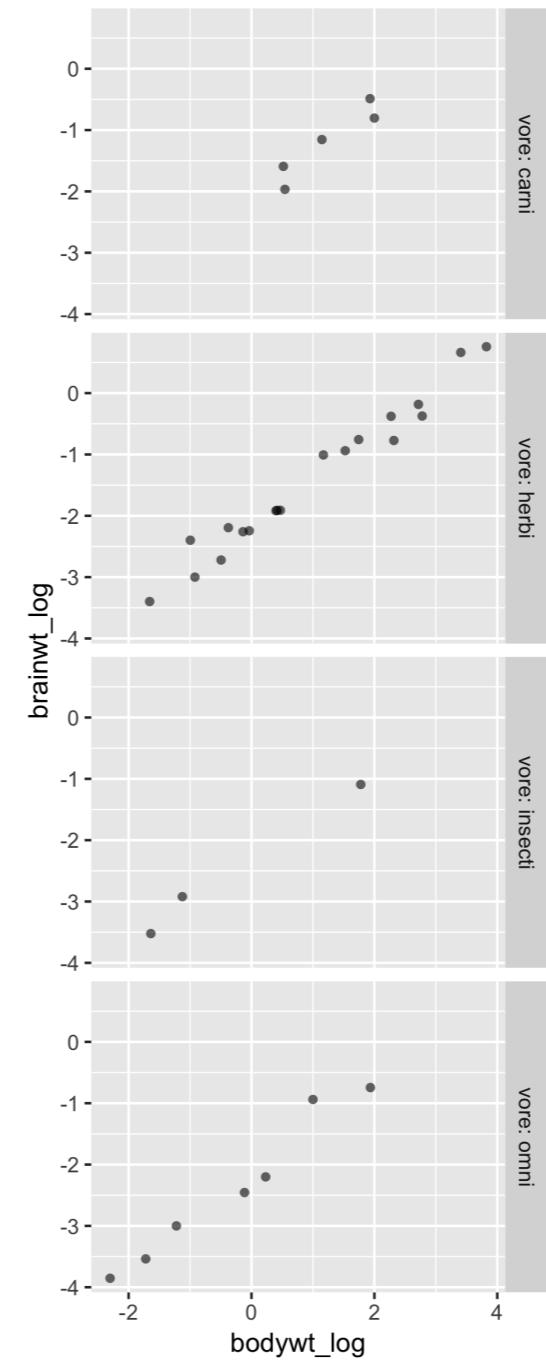


# Using `label_both` adds the variable name

```
# Print variable name also  
p +  
  facet_grid(rows = vars(vore),  
             labeller = label_both)
```

This option just shows the category name and value

```
p +  
  facet_grid(rows = vars(vore,  
                         conservation),  
             labeller = label_context)
```

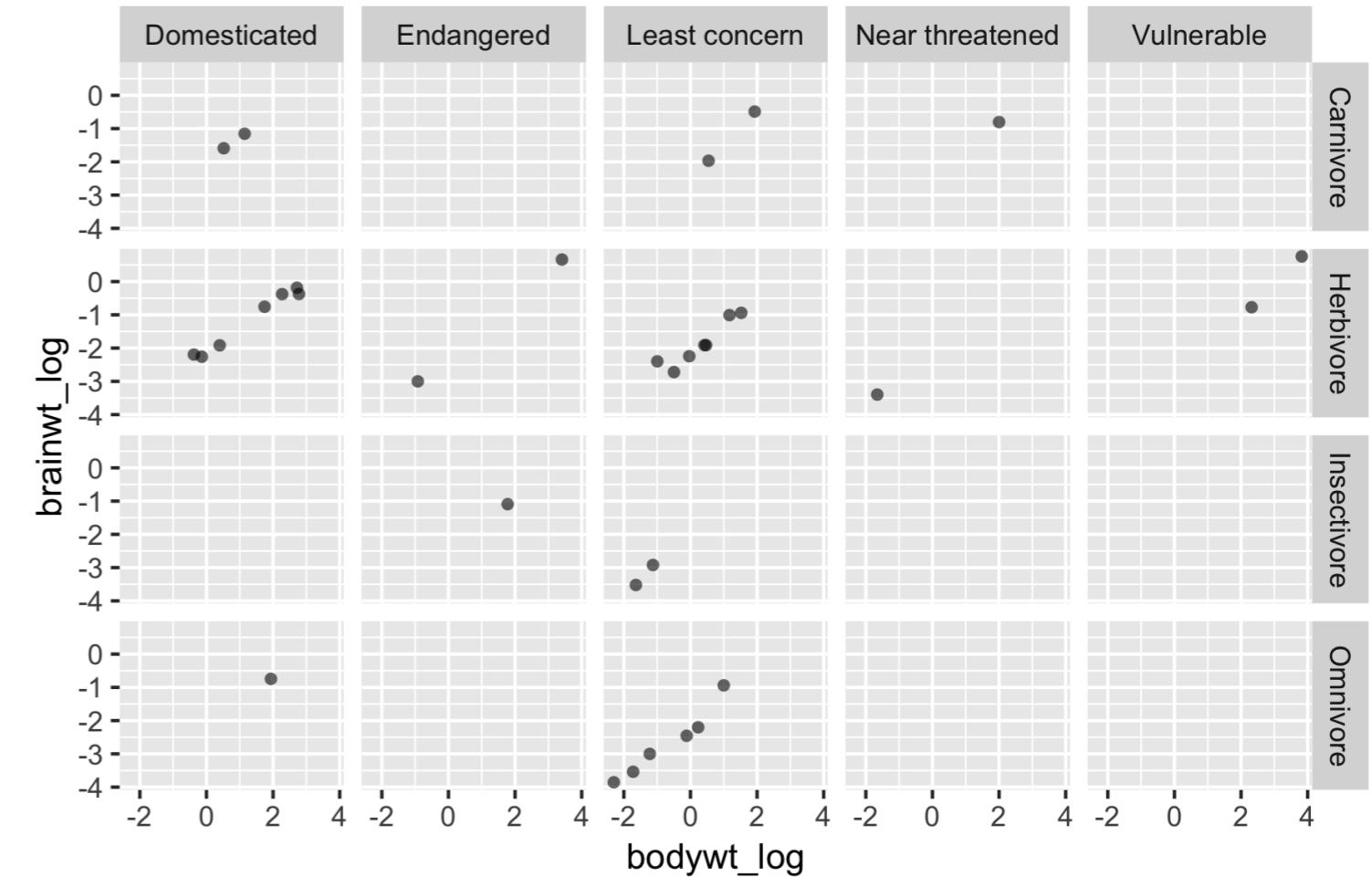


# Relabeling and reordering factors

```
msleep2$conservation <- fct_recode(msleep2$conservation,  
                                     Domesticated = "domesticated",  
                                     `Least concern` = "lc",  
                                     `Near threatened` = "nt",  
                                     Vulnerable = "vu",  
                                     Endangered = "en")  
  
msleep2$vore = fct_recode(msleep2$vore,  
                           Carnivore = "carni",  
                           Herbivore = "herbi",  
                           Insectivore = "insecti",  
                           Omnivore = "omni")
```

# Reinitialize plot with new labels

```
# Plot  
p <- ggplot(msleep2, aes(bodywt_log,  
                           brainwt_log)) +  
  geom_point(alpha = 0.6, shape = 16) +  
  coord_fixed()  
  
p +  
  facet_grid(rows = vars(vore),  
             cols = vars(conservation))
```



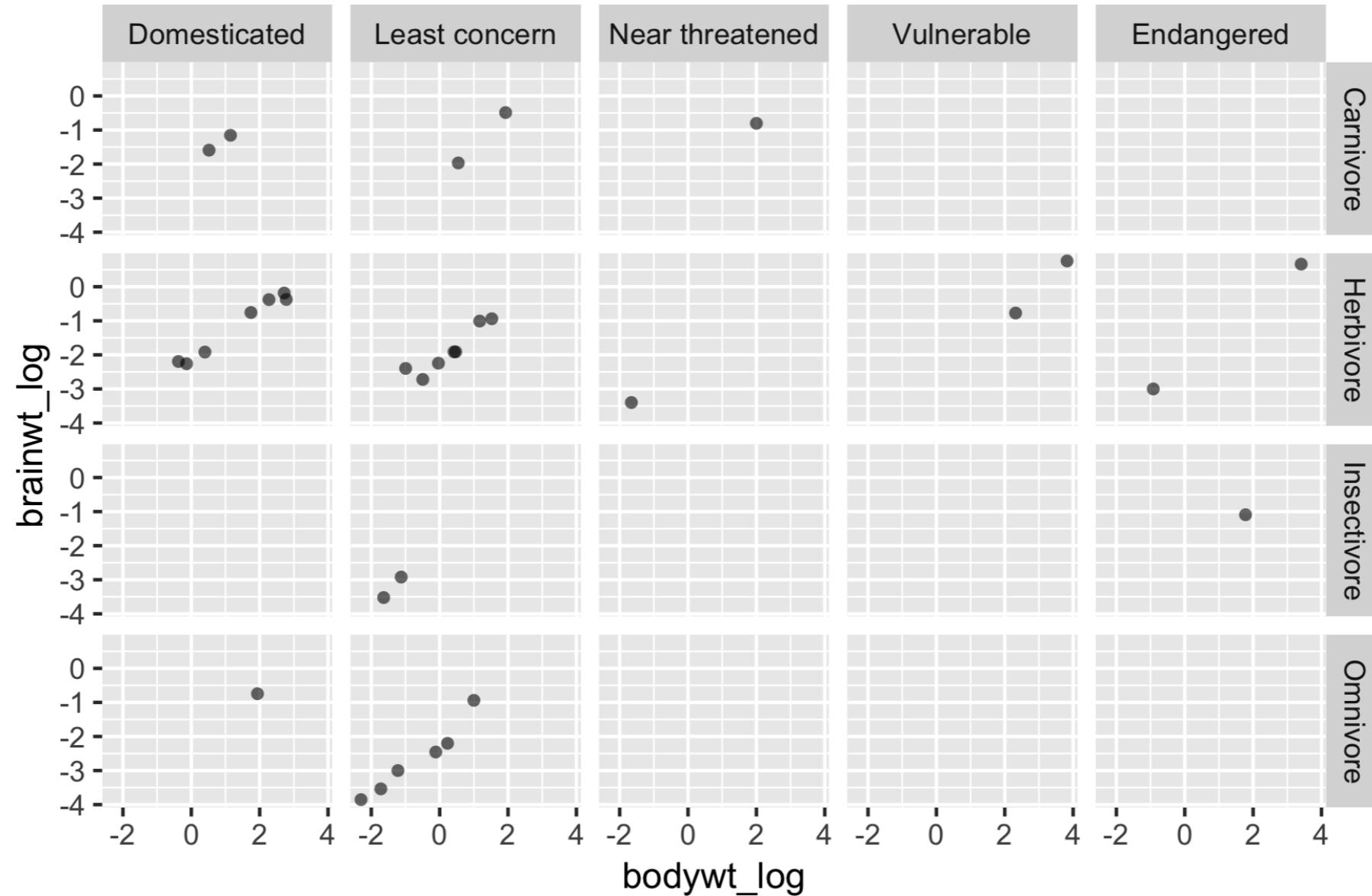
# Changing the order of levels

```
# Change order of levels:
```

```
msleep2$conservation = fct_relevel(msleep2$conservation,  
                                     c("Domesticated",  
                                       "Least concern",  
                                       "Near threatened",  
                                       "Vulnerable",  
                                       "Endangered"))
```

```
msleep2 <- msleep2 %>%  
  # Arrange from lo to hi weight  
  arrange(-bodywt_log) %>%  
  # Redefine factor levels in order  
  mutate(name = as_factor(name))
```

# Reinitialize plot with new order



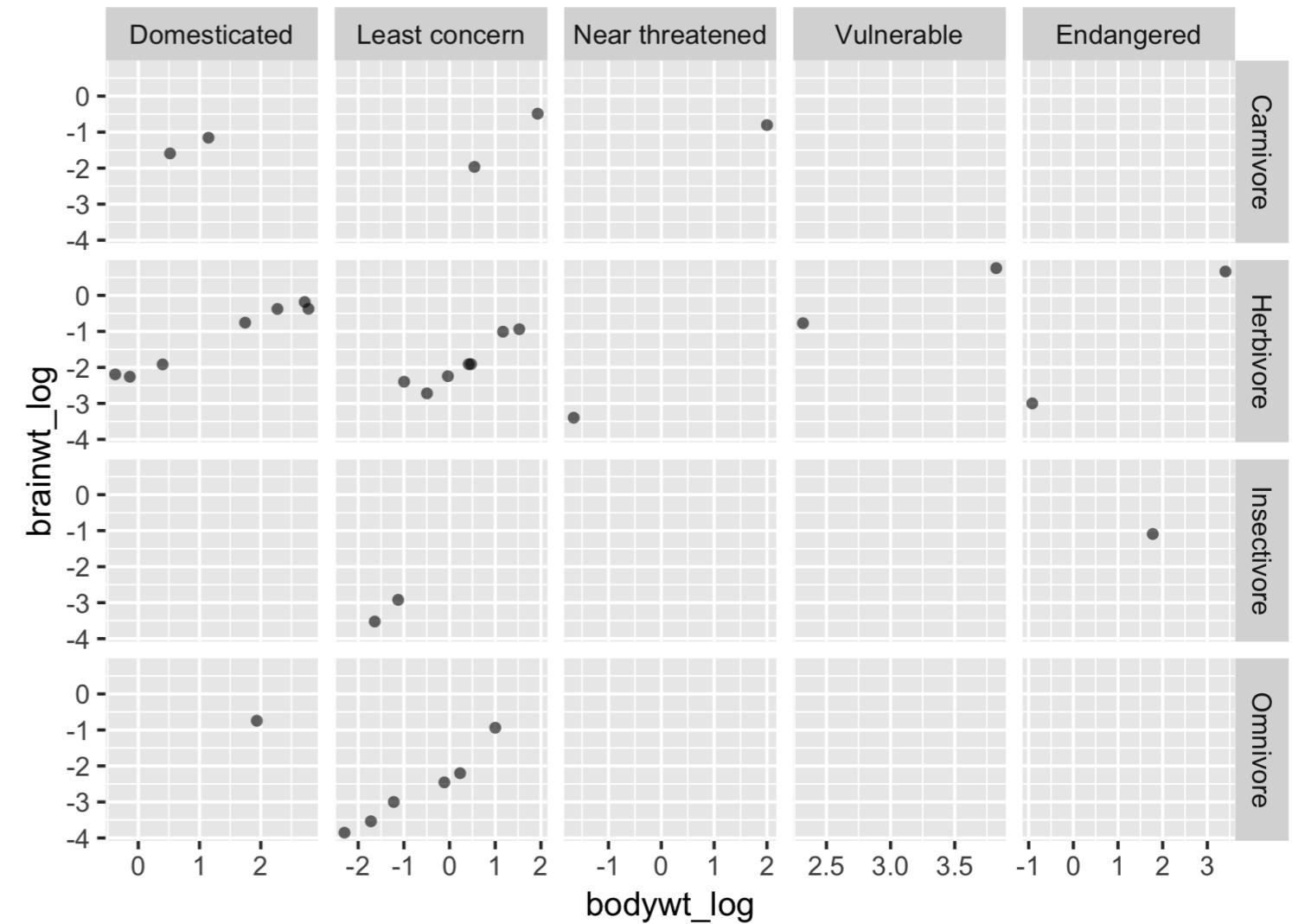
# ... but not with fixed scales

```
ggplot(msleep2, aes(bodywt_log,  
                      brainwt_log)) +  
  geom_point(alpha = 0.6, shape = 16) +  
  coord_fixed() +  
  facet_grid(rows = vars(vore),  
             cols = vars(conservation),  
             scales = "free_x")
```

Error: coord\_fixed doesn't support free scales

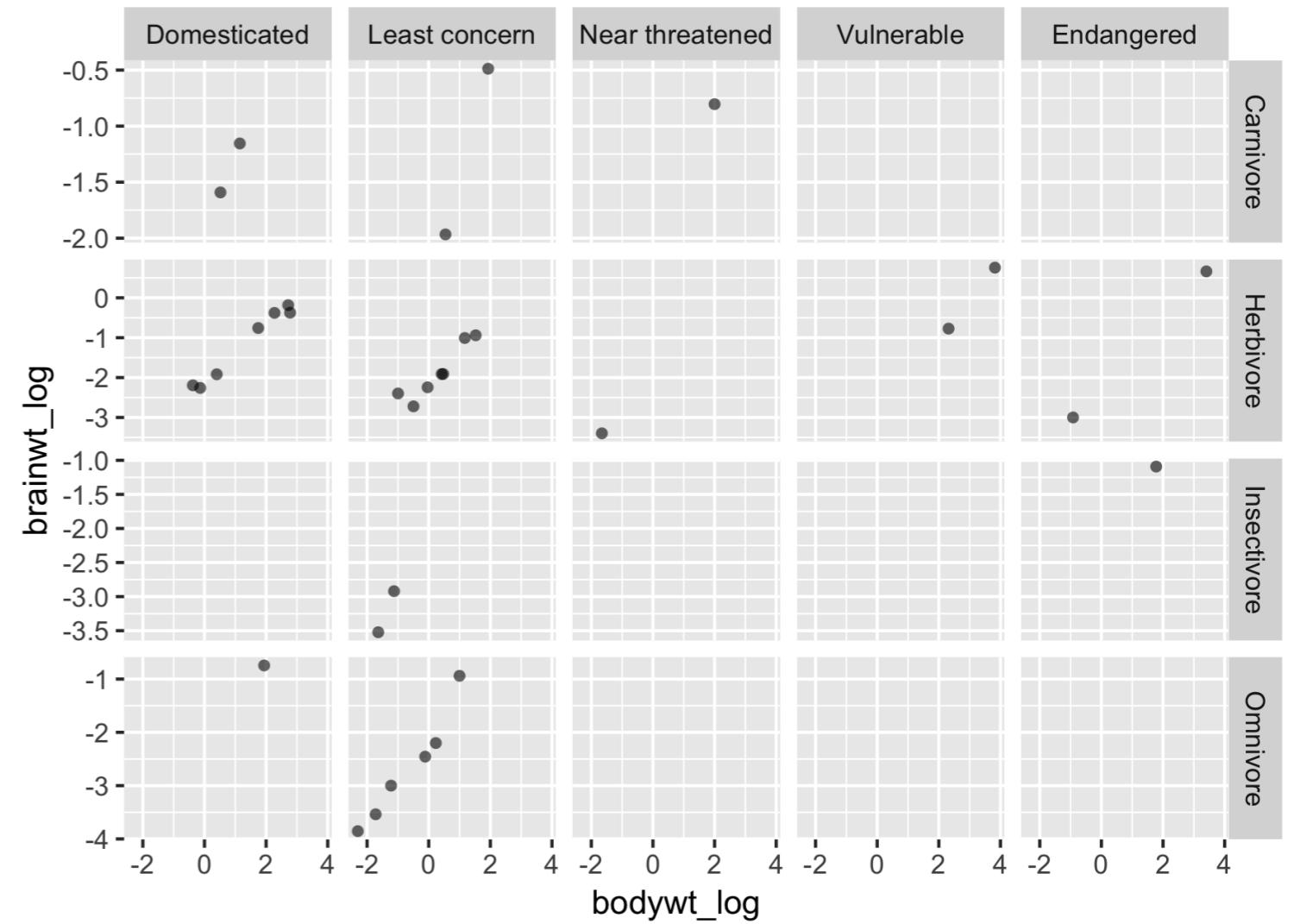
# Adjusting the plotting space

```
ggplot(msleep2, aes(bodywt_log,  
                      brainwt_log)) +  
  geom_point(alpha = 0.6, shape = 16) +  
  facet_grid(rows = vars(vore),  
             cols = vars(conservation),  
             scales = "free_x")
```



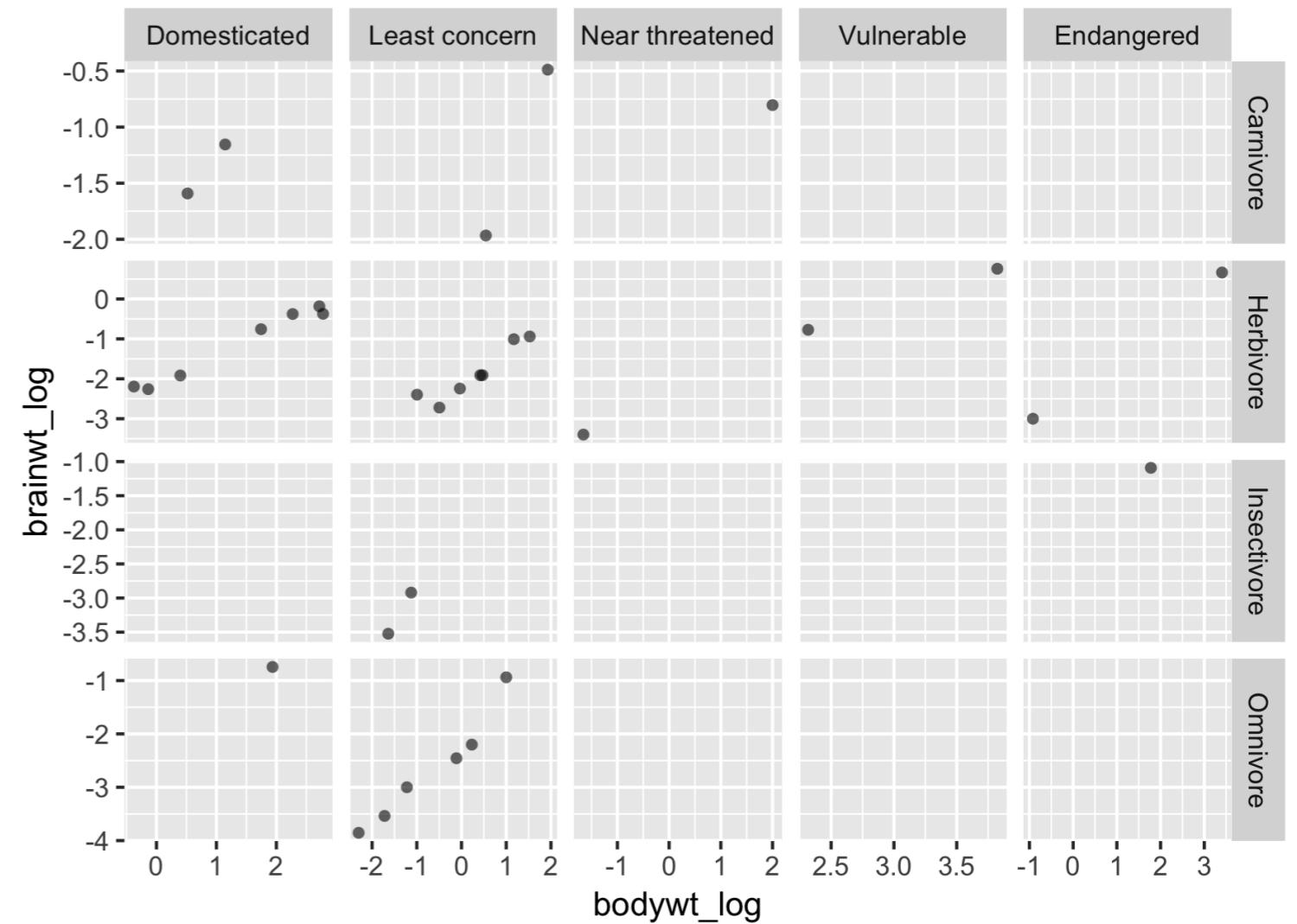
# Adjusting the plotting space

```
ggplot(msleep2, aes(bodywt_log,  
                      brainwt_log)) +  
  geom_point(alpha = 0.6, shape = 16) +  
  facet_grid(rows = vars(vore),  
             cols = vars(conservation),  
             scales = "free_y")
```



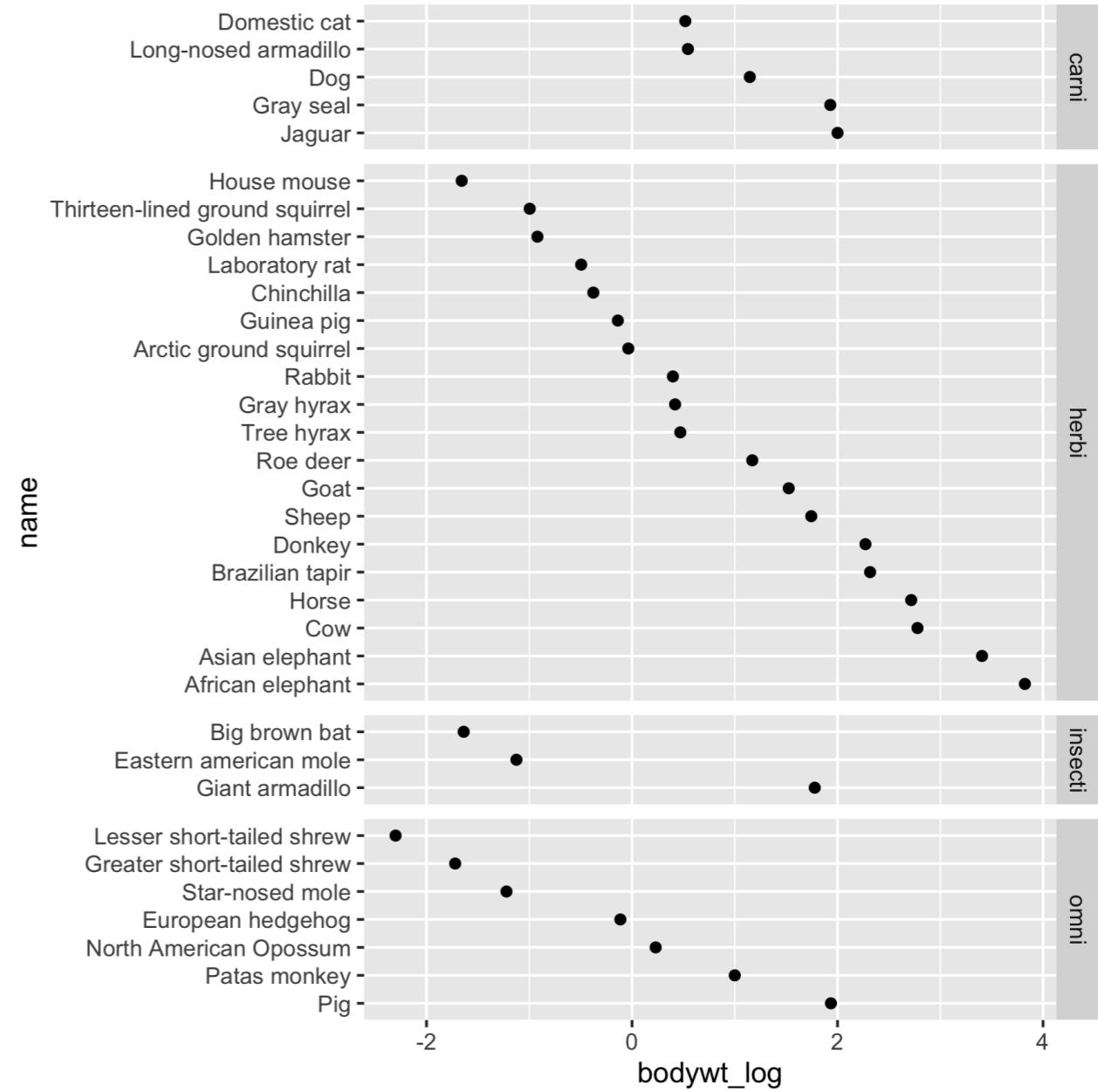
# Adjusting the plotting space

```
ggplot(msleep2, aes(bodywt_log,  
                      brainwt_log)) +  
  geom_point(alpha = 0.6, shape = 16) +  
  facet_grid(rows = vars(vore),  
             cols = vars(conservation),  
             scales = "free")
```



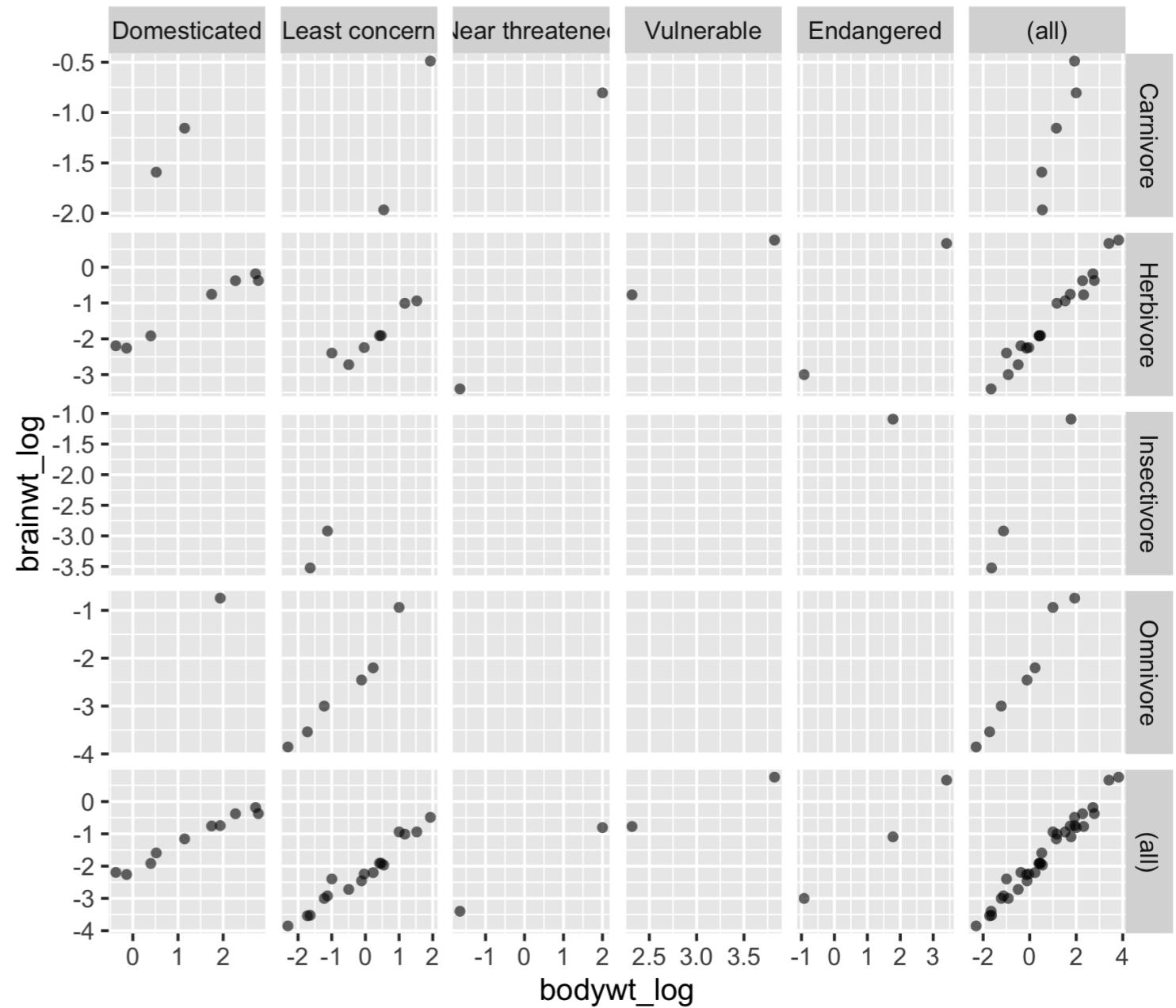
# Final adjustments

```
msleep2 <- msleep2 %>%  
  # Arrange from lo to hi weight  
  arrange(-bodywt_log) %>%  
  # Redefine factor levels in order  
  mutate(name = as_factor(name))  
  
# New order is reflected in y axis  
ggplot(msleep2, aes(x = bodywt_log,  
                     y = name)) +  
  geom_point() +  
  # Free the y scales and space  
  facet_grid(rows = vars(vore),  
             scales = "free_y",  
             space = "free_y")
```



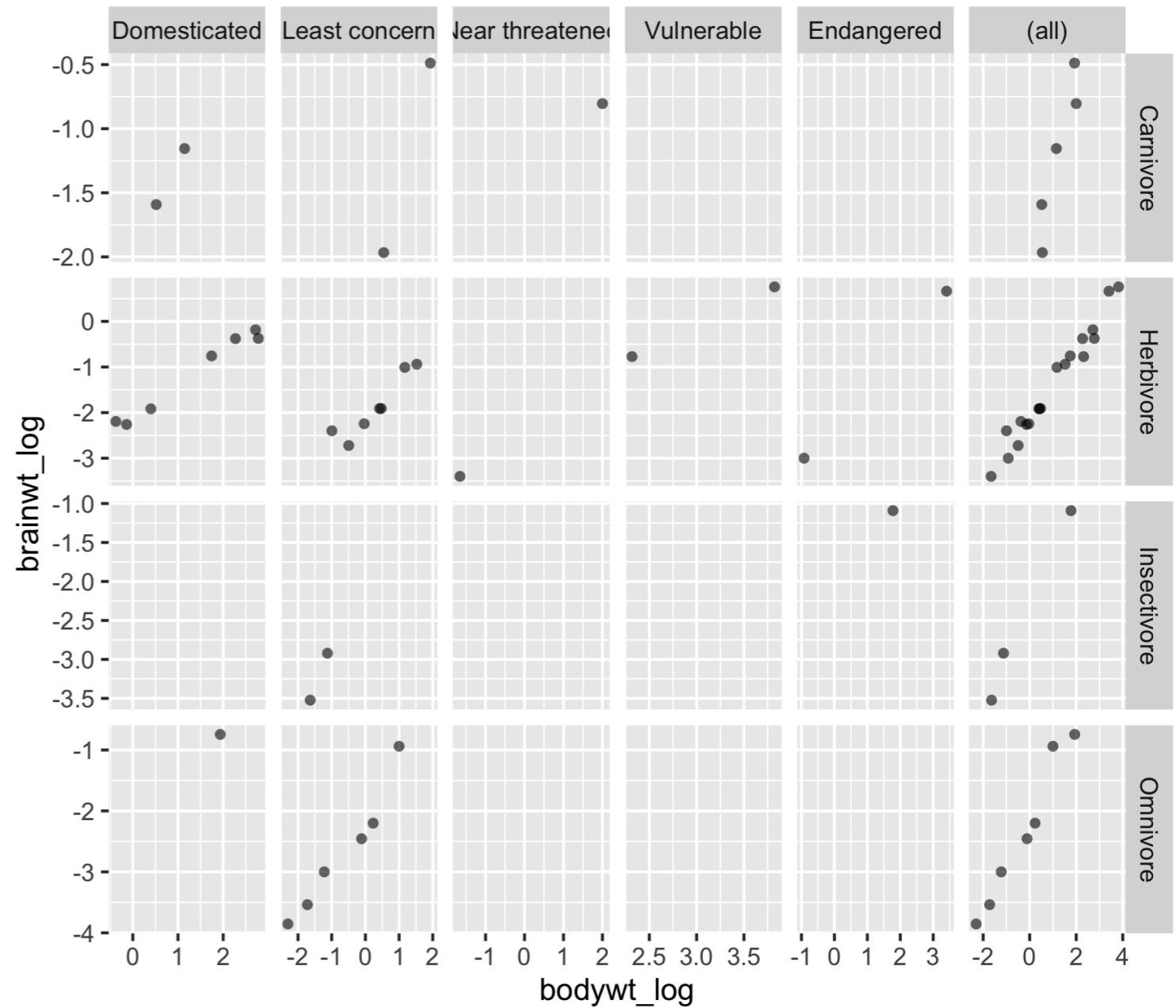
# Using margin plots

```
ggplot(msleep2, aes(bodywt_log,  
                      brainwt_log)) +  
  geom_point(alpha = 0.6, shape = 16) +  
  facet_grid(rows = vars(vore),  
             cols = vars(conservation),  
             scales = "free",  
             margins = TRUE)
```



# Using margin plots

```
ggplot(msleep2, aes(bodywt_log,  
                      brainwt_log)) +  
  geom_point(alpha = 0.6, shape = 16) +  
  facet_grid(rows = vars(vore),  
             cols = vars(conservation),  
             scales = "free",  
             margins = "conservation")
```



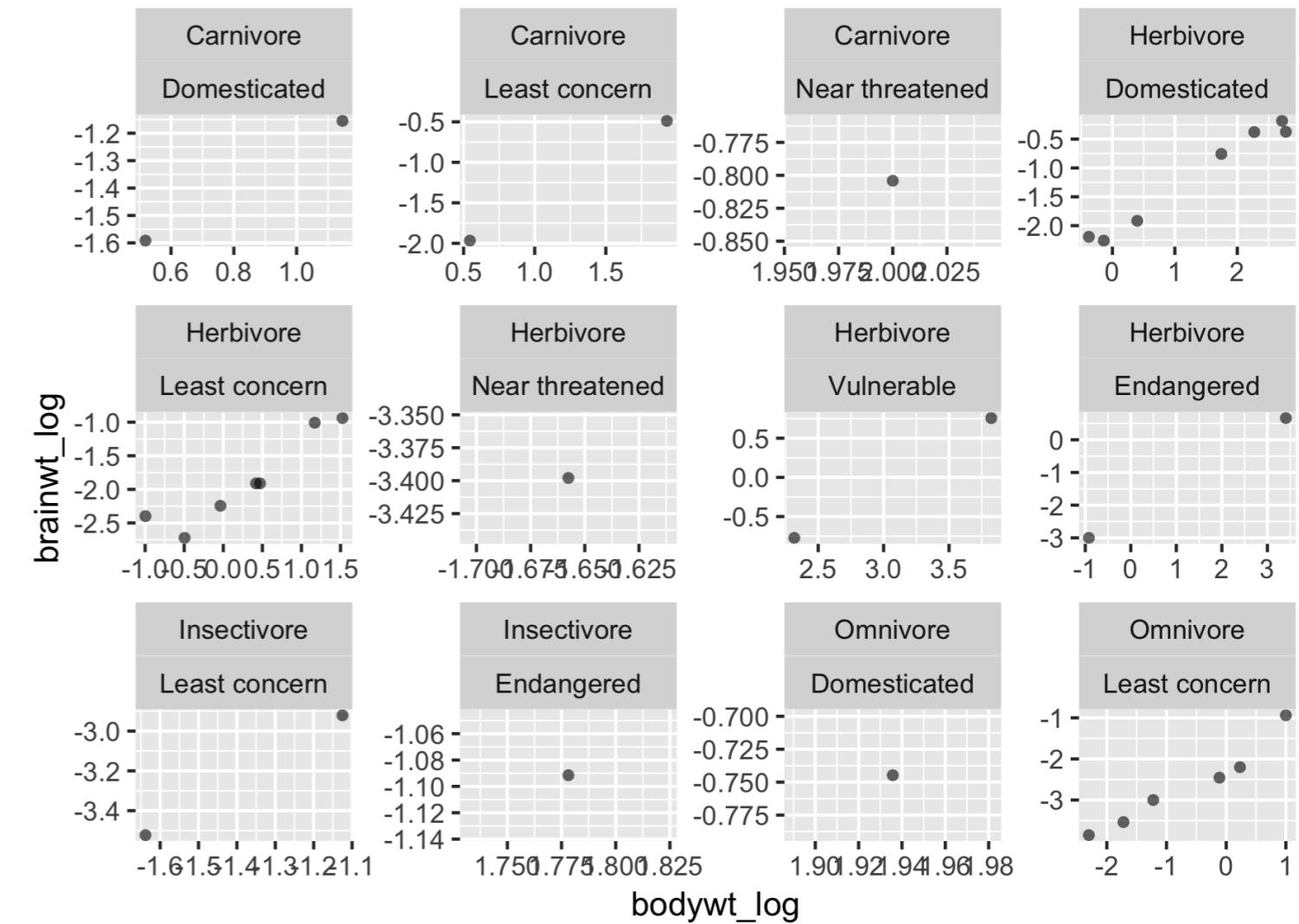
# Using `facet_wrap()`

Use cases:

1. When you want both x and y axes to be free on every individual plot
  - i.e. Not just per row or column as per `facet_grid()`
2. When your categorical (factor) variable has many groups (levels)
  - i.e. too many sub plots for column or row-wise faceting
  - A more typical scenario

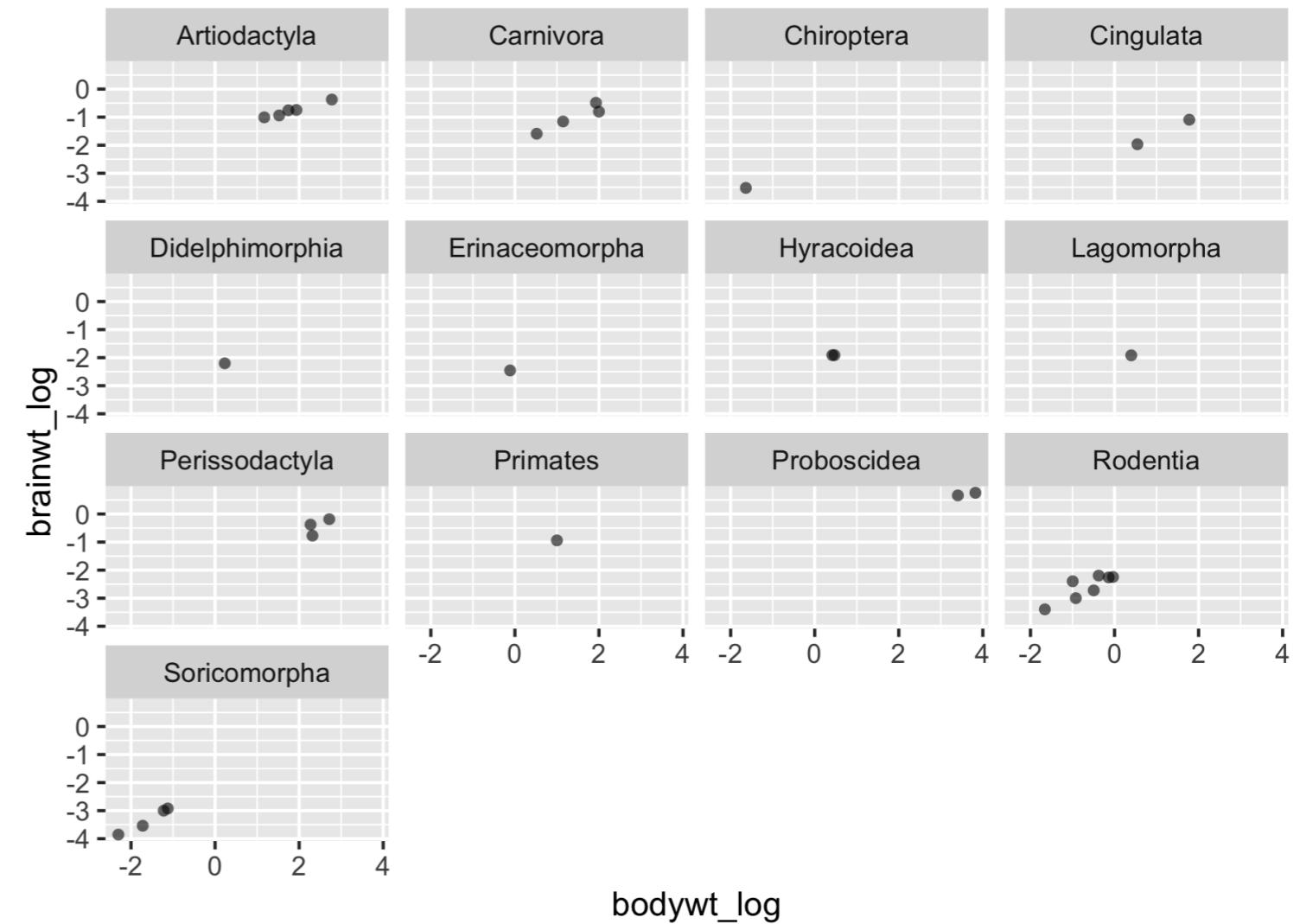
# Using facet\_wrap() - Scenario 1

```
ggplot(msleep2, aes(bodywt_log,  
                      brainwt_log)) +  
  geom_point(alpha = 0.6, shape = 16) +  
  facet_wrap(vars(vore, conservation),  
             scales = "free")
```



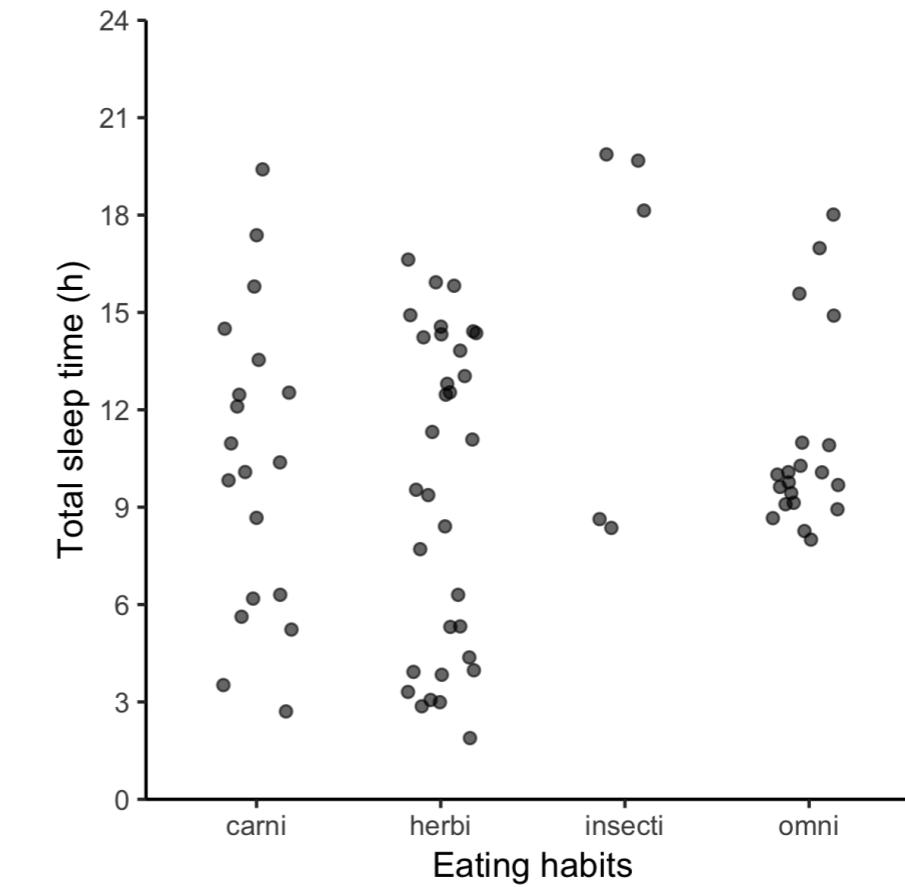
# Using `facet_wrap()` - Scenario 2

```
ggplot(msleep2, aes(bodywt_log,  
                      brainwt_log)) +  
  geom_point(alpha = 0.6, shape = 16) +  
  facet_wrap(vars(order))
```



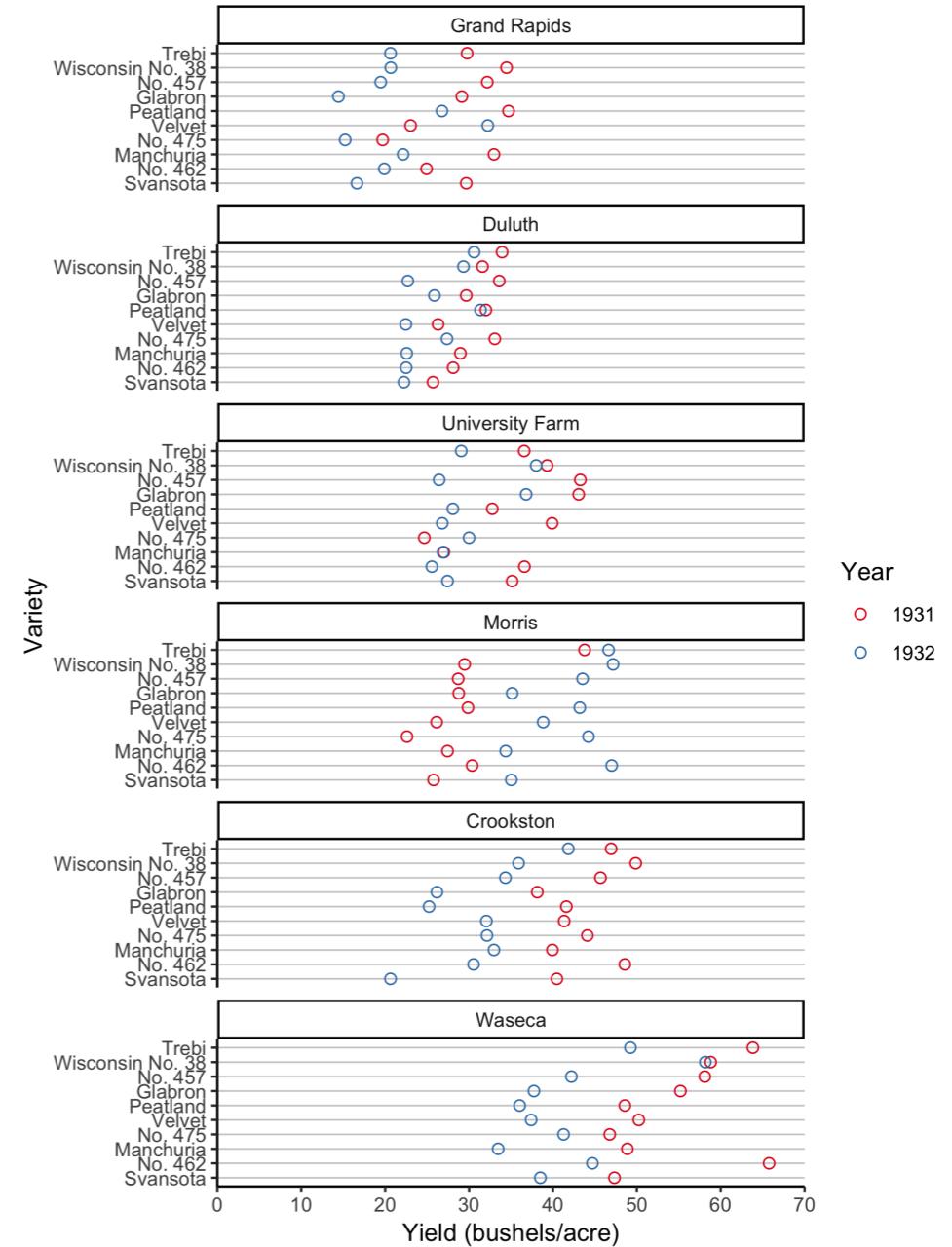
# Individual data points

```
# position  
posn_j <- position_jitter(width = 0.2)  
  
# plot  
d +  
  geom_point(alpha = 0.6,  
             position = posn_j)
```



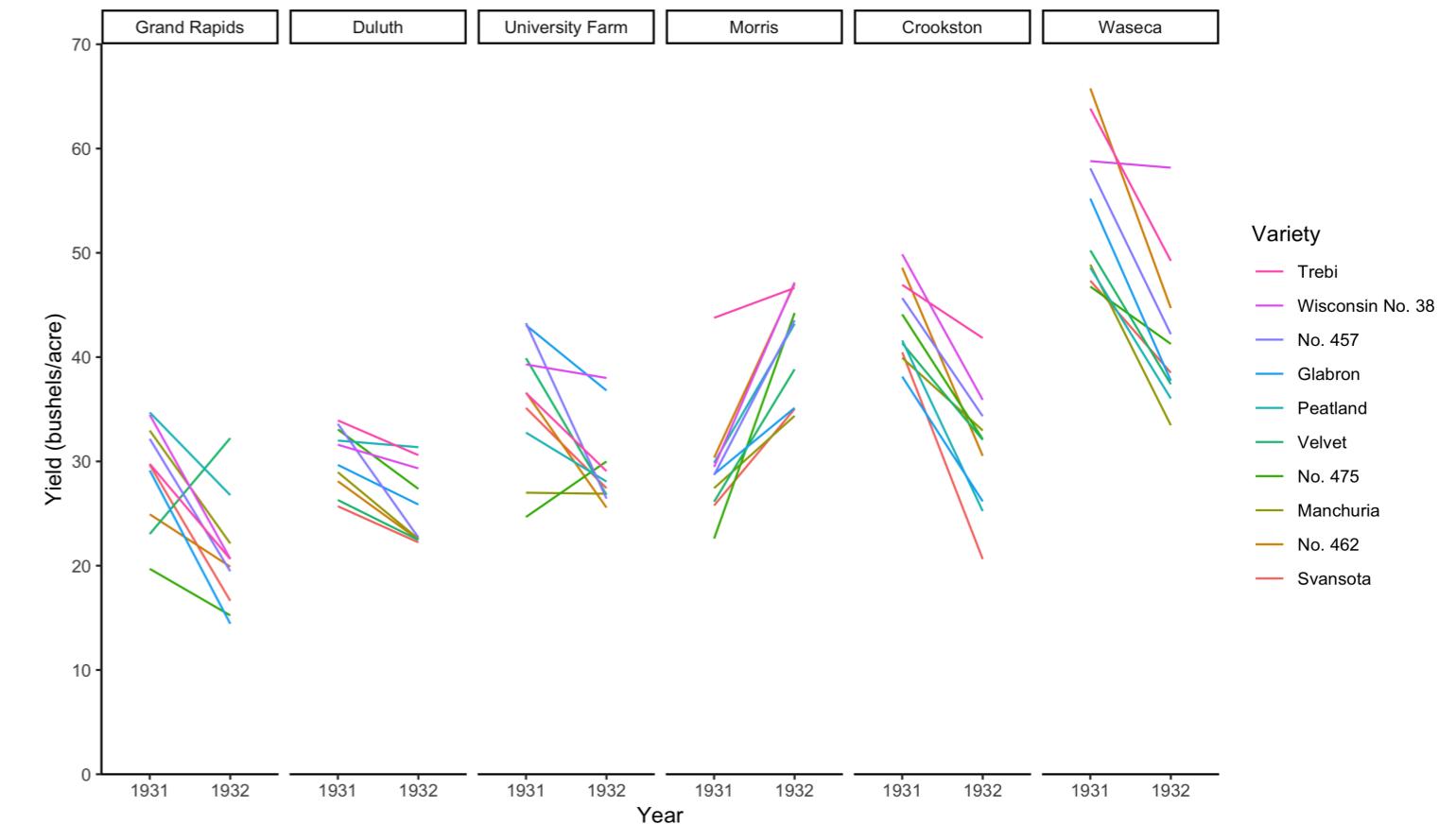
# A dot plot

```
ggplot(barley, aes(yield, variety,  
                    color = year)) +  
  
  geom_point(...) +  
  
  facet_wrap(vars(site), ncol = 1) +  
  
  ...
```



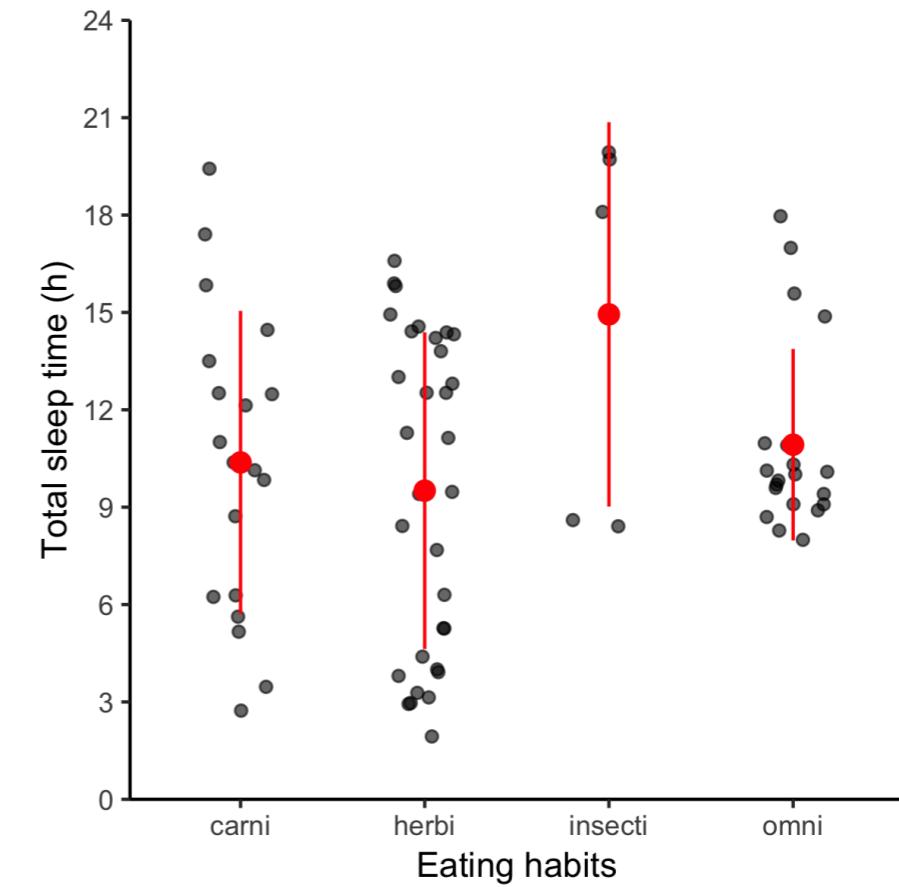
# As a time series

```
ggplot(barley, aes(year, yield,  
                    group = variety,  
                    color = variety)) +  
  
  geom_line() +  
  
  facet_wrap(vars(site), nrow = 1) +  
  
  ...
```



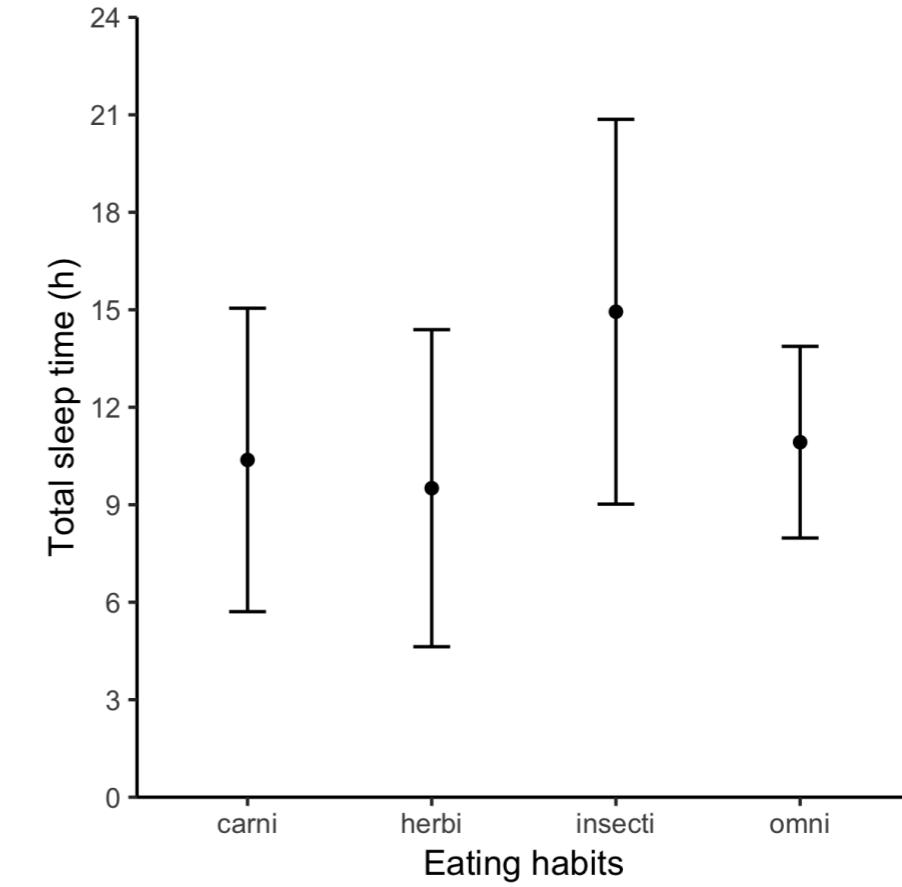
# geom\_pointrange()

```
d +  
  geom_point(...) +  
  stat_summary(fun.data = mean_sdl,  
               mult = 1,  
               width = 0.2,  
               color = "red")
```

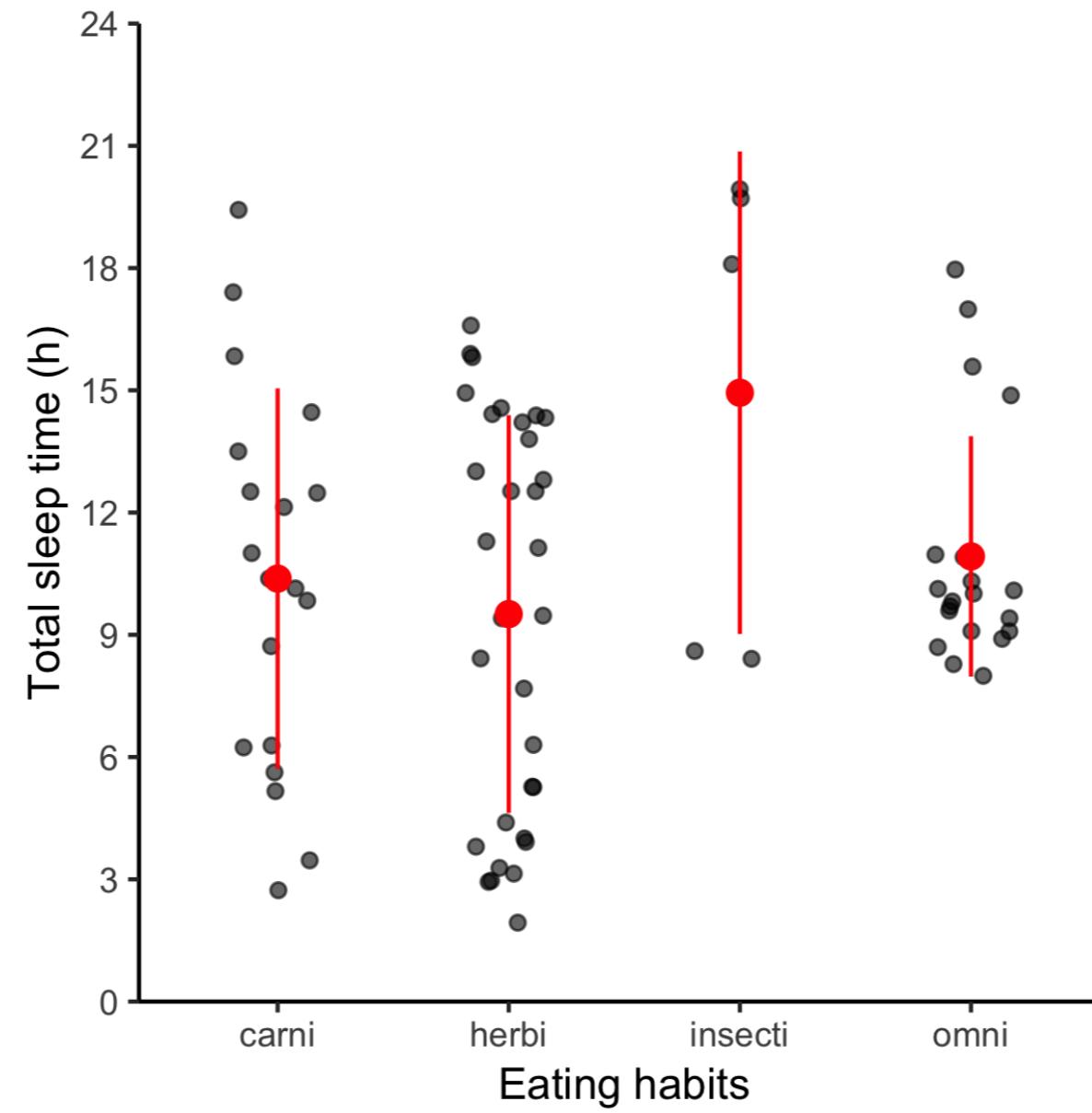


# Without data points

```
d +  
  stat_summary(fun.y = mean,  
               geom = "point") +  
  stat_summary(fun.data = mean_sdl,  
               fun.args = list(mult = 1),  
               geom = "errorbar",  
               width = 0.2)
```

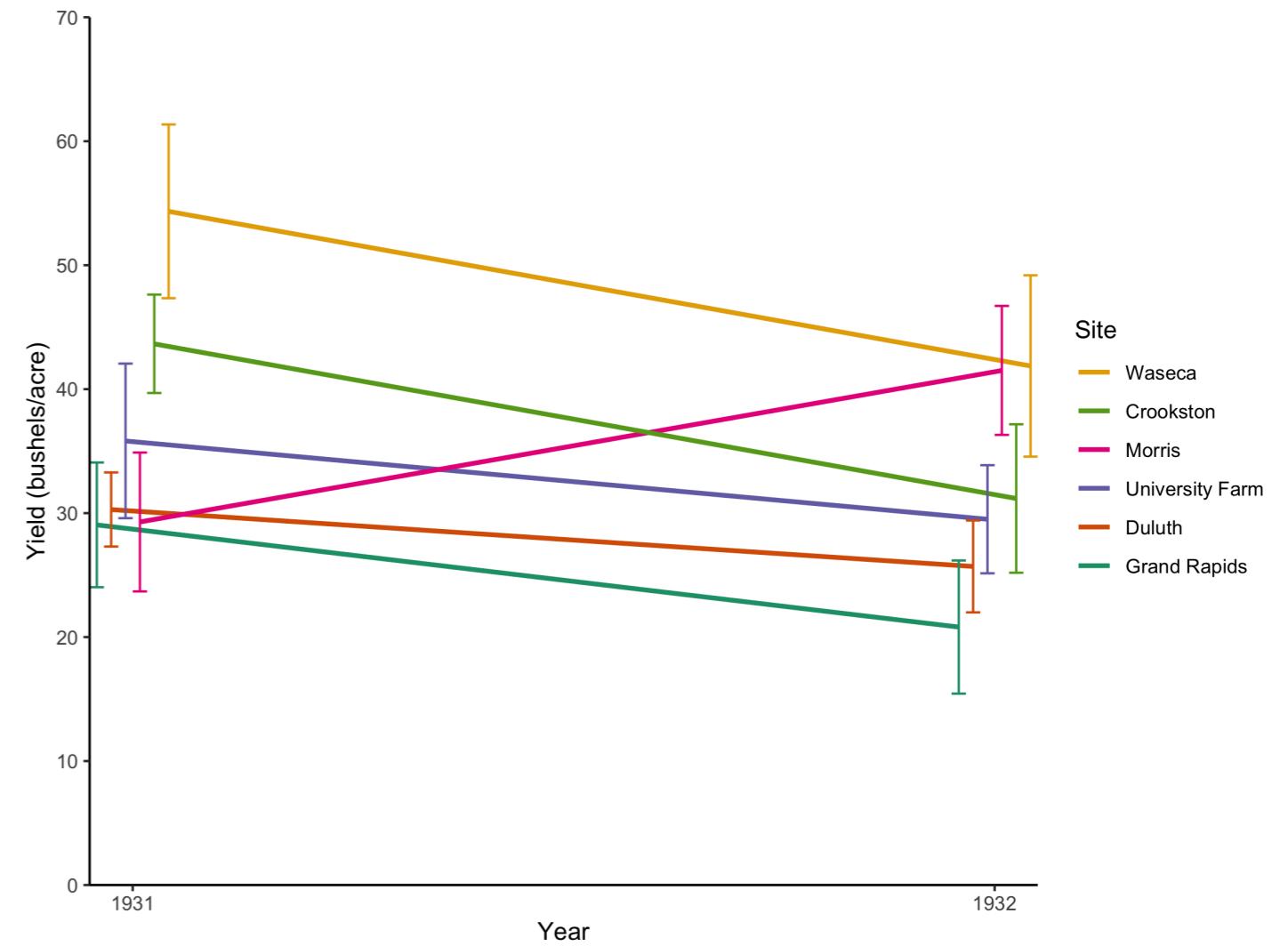


# Bars are not necessary



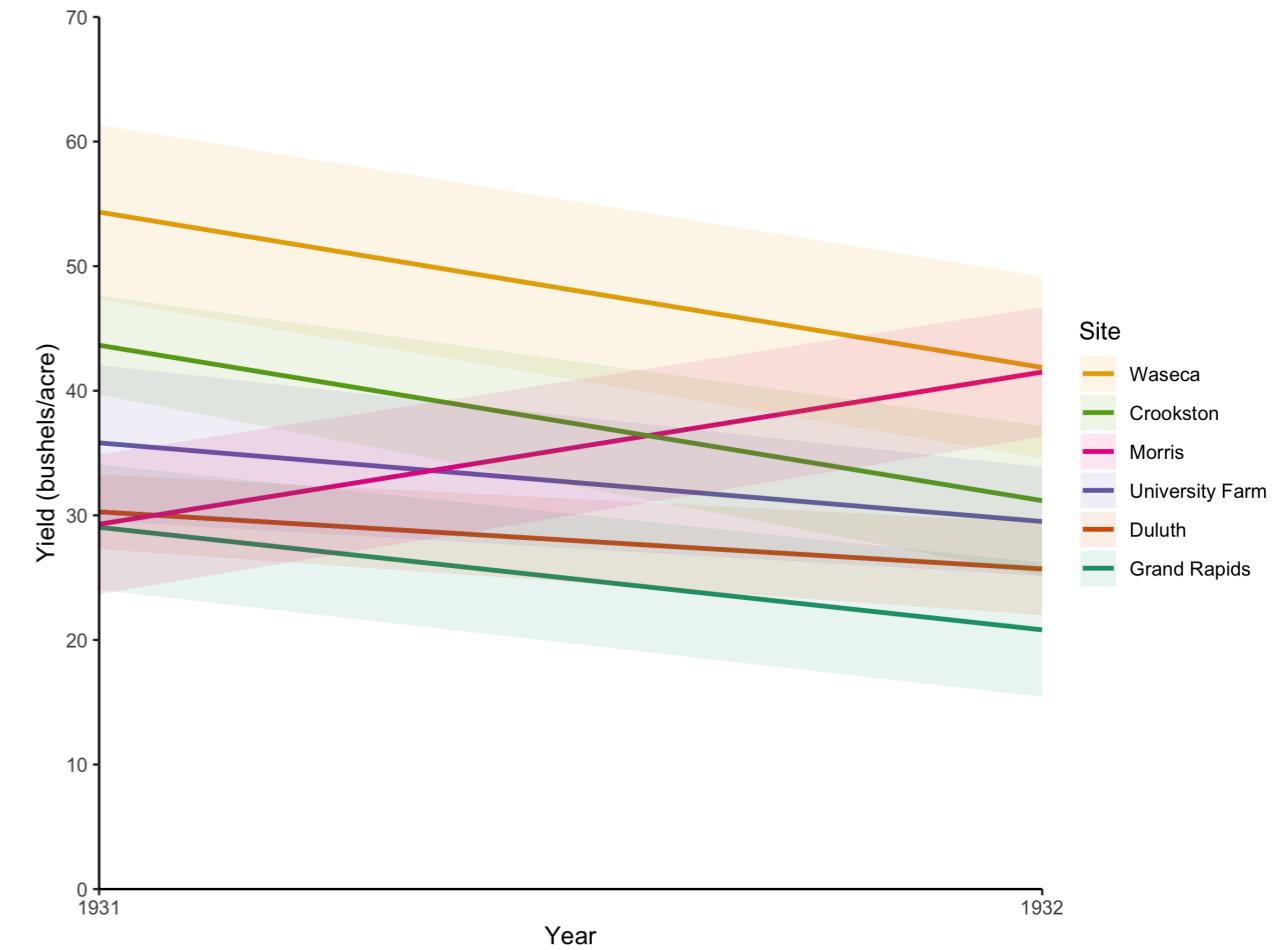
# Using dodged error bars

```
ggplot(barley, aes(x = year, y = yield,  
                    group = site,  
                    color = site)) +  
  
  stat_summary(fun.y = mean,  
               geom = "line", ...) +  
  
  stat_summary(fun.data = mean_sdl,  
               geom = "errorbar", ...) +  
  
  ...
```



# Using ribbons for error

```
# Using barely, plot yield vs. year, colored, grouped, and
filled by site
ggplot(barley, aes(x = year, y = yield,
| | | color = site, group = site, fill = site)) +
  # Add a line summary stat aggregated by mean
  stat_summary(fun.y = mean, geom = "line") +
  # Add a ribbon summary stat with 10% opacity, no color
  stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1),
| | | | | | | geom = "ribbon", alpha = 0.1, color = NA)
```



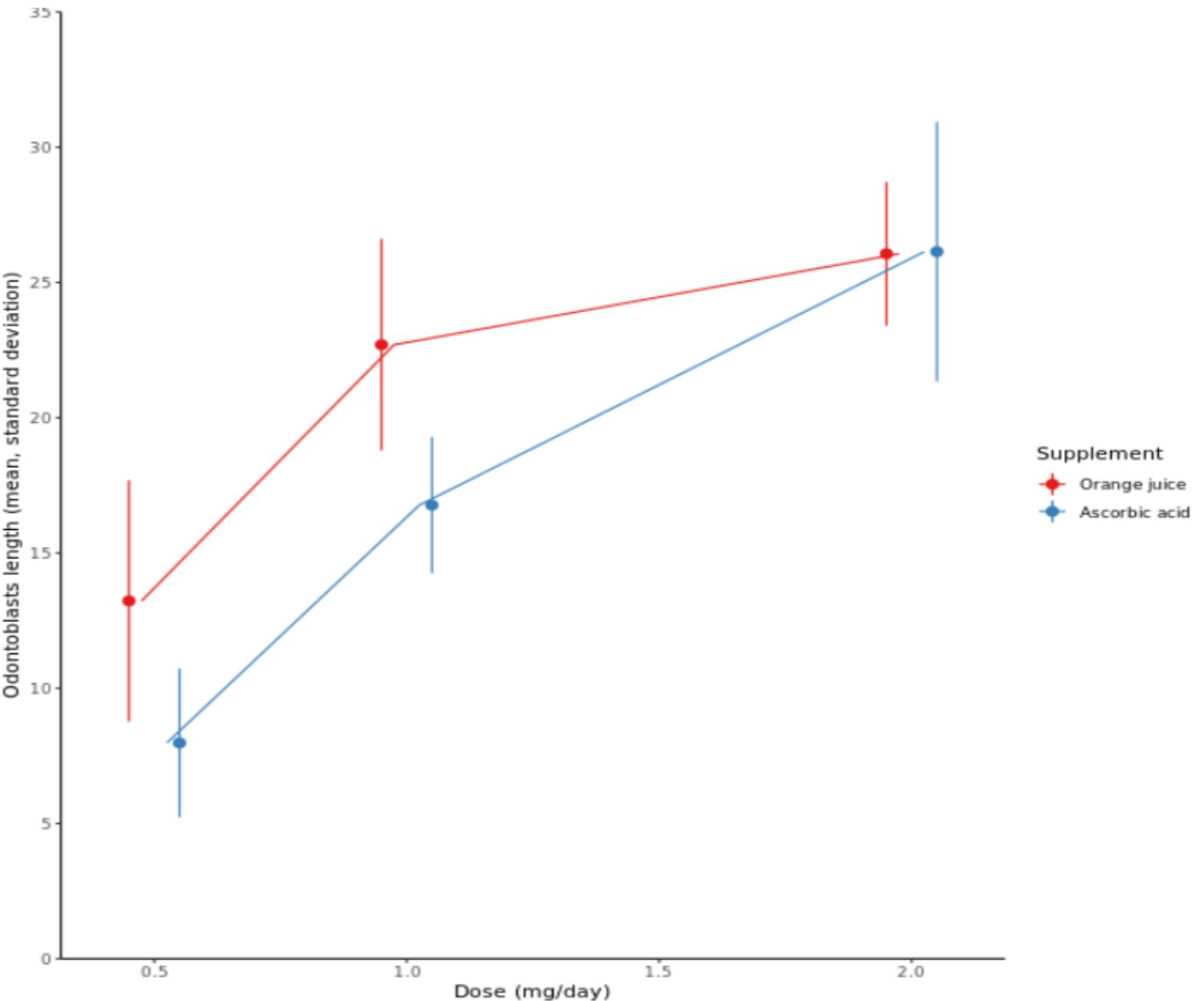
```

# Change type
TG$dose <- as.numeric(as.character(TG$dose))

# Plot
growth_by_dose <- ggplot(TG, aes(dose, len, color = supp)) +
  stat_summary(fun.data = mean_sdl,
  fun.args = list(mult = 1),
  position = position_dodge(0.2)) +
  stat_summary(fun.y = mean,
  geom = "line",
  position = position_dodge(0.1)) +
  theme_classic() +
  # Adjust labels and colors:
  labs(x = "Dose (mg/day)",
  y = "Odontoblasts length (mean, standard deviation)",
  color = "Supplement") +
  scale_color_brewer(palette = "Set1",
  labels = c("Orange juice", "Ascorbic acid")) +
  scale_y_continuous(limits = c(0,35),
  breaks = seq(0, 35, 5),
  expand = c(0,0))

# View plot
growth_by_dose

```



# A basic heat map

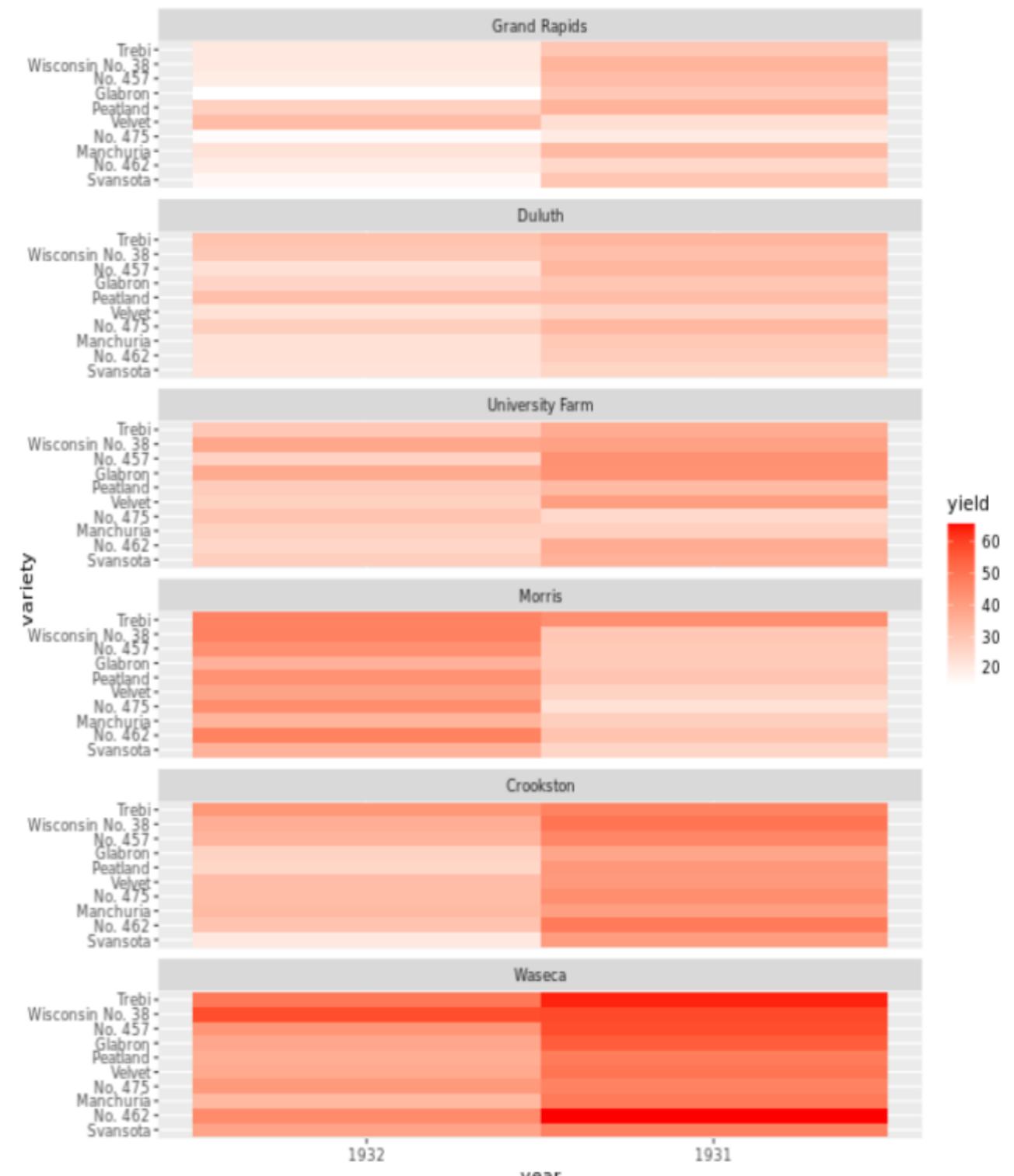
```
# Previously defined
```

```
ggplot(barley, aes(x = year, y = variety, fill = yield)) +  
  geom_tile() +  
  # Facet, wrapping by site, with 1 column  
  facet_wrap(facets = vars(site), ncol = 1) +  
  # Add a fill scale using an 2-color gradient  
  scale_fill_gradient(low = "white", high = "red")
```

Heat maps are most useful if you have:

- \* a small number of boxes
- \* a clear pattern that allows

There are few groups with large differences.



# A basic heat map

```
# A palette of 9 reds
red_brewer_palette <- brewer.pal(9, "Reds")

# Update the plot
ggplot(barley, aes(x = year, y = variety, fill = yield)) +
  geom_tile() +
  facet_wrap(facets = vars(site), ncol = 1) +
  # Update scale to use n-colors from red_brewer_palette
  scale_fill_gradientn(colors = red_brewer_palette)
```

Heat maps are most useful if you have:

- \* a small number of boxes
- \* a clear pattern that allows

There are few groups with large differences.

