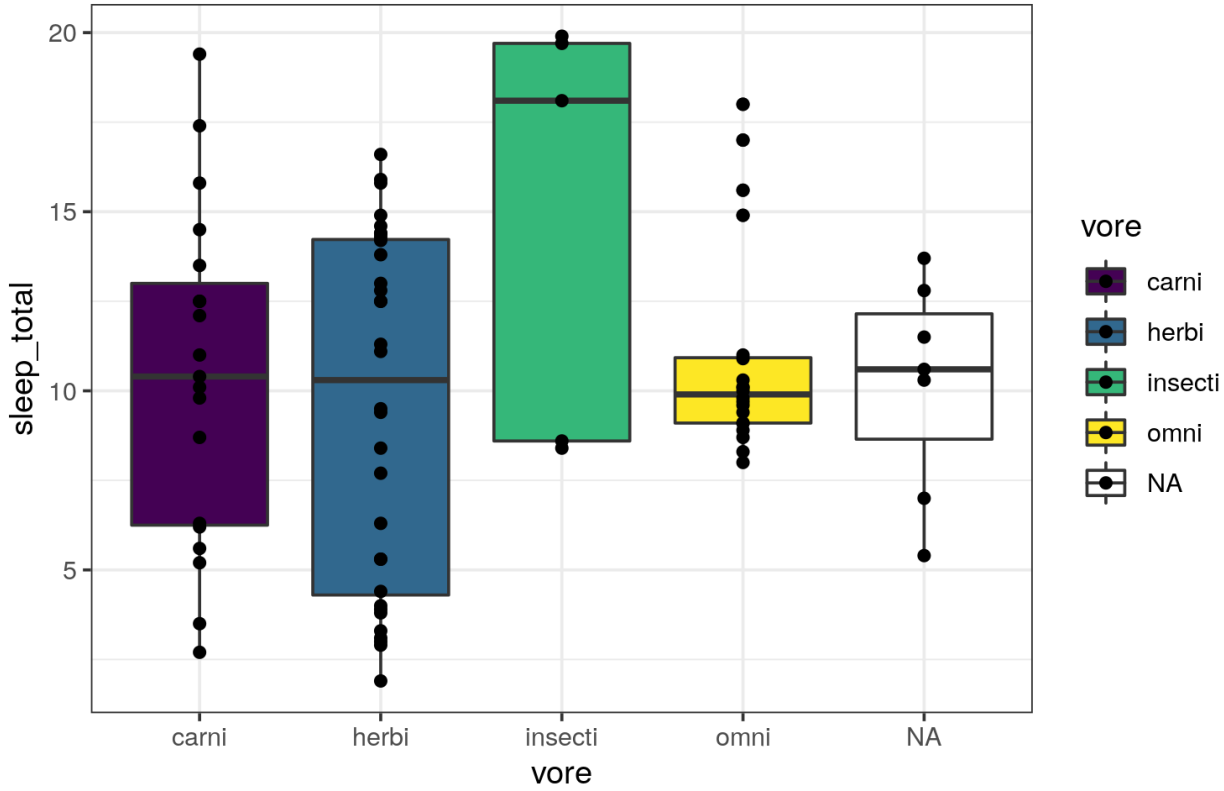
TWS R Workshop

Creating Figures as an Intro to R

Using the **`ggplot2`** package



Presentation available here: https://git.io/Je699

Introductions

Dr. Steffi LaZerte

- Background in Biology (Animal Behaviour)
- Working with R since 2007
- Professional R programmer/consultant since 2017

What about you?

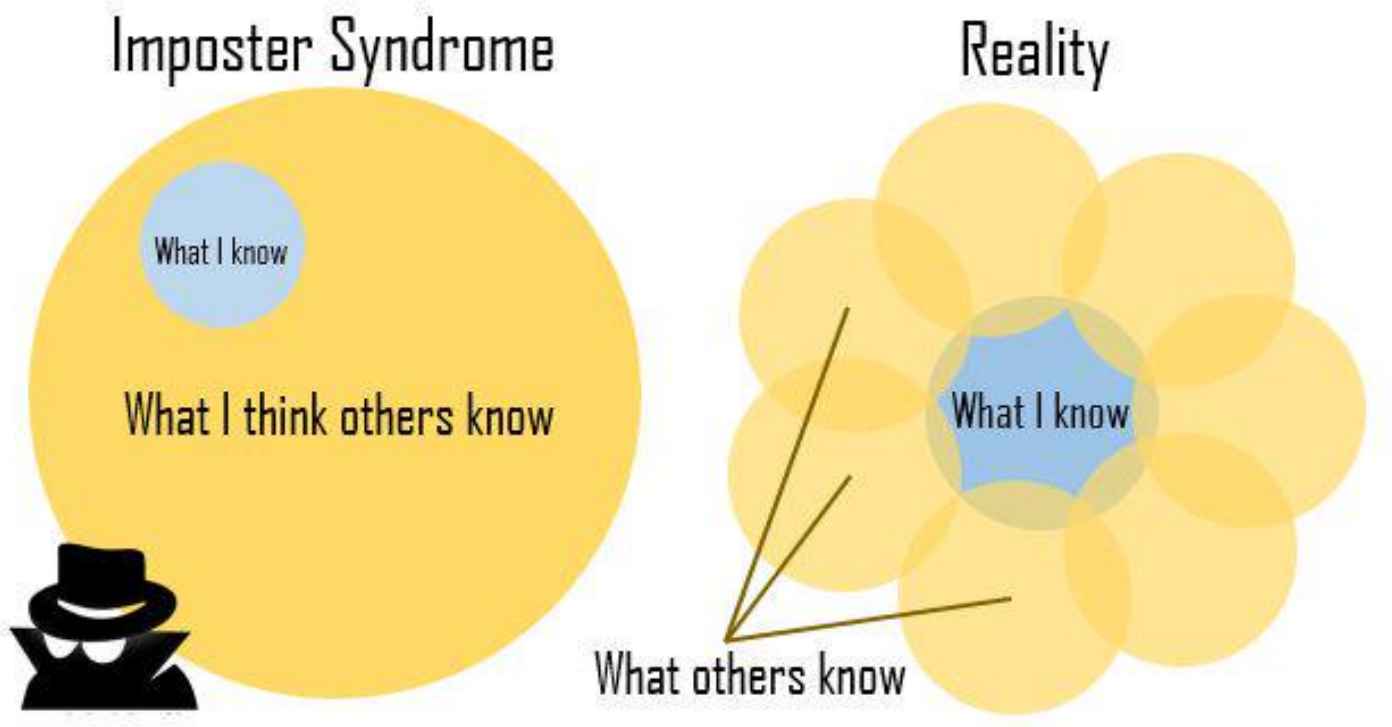
- Name
- Background (Area of study, etc.)
- Familiarity with Computer Programming (C+, Java, HTML, PHP, python, SAS)
- Familiarity with R
  - I've heard of R
  - I've installed R (before this class)
  - I've used R
  - I've used R a lot
  - I use R all the time

Outline

1. A little about R
2. Creating figures with **`ggplot2`**
   - Basic plot
   - Common plot types
   - Plotting by categories
   - Adding statistics
   - Customizing plots
   - Annotating plots
3. Saving figures
4. (EXTRA) Combining figures

ImpostR Syndrome

ImpostR Syndrome



David Whittaker

Impost R Syndrome

Moral of the story?

Make friends, code in groups, learn together and don't beat yourself up

@allison_horst

Why R?

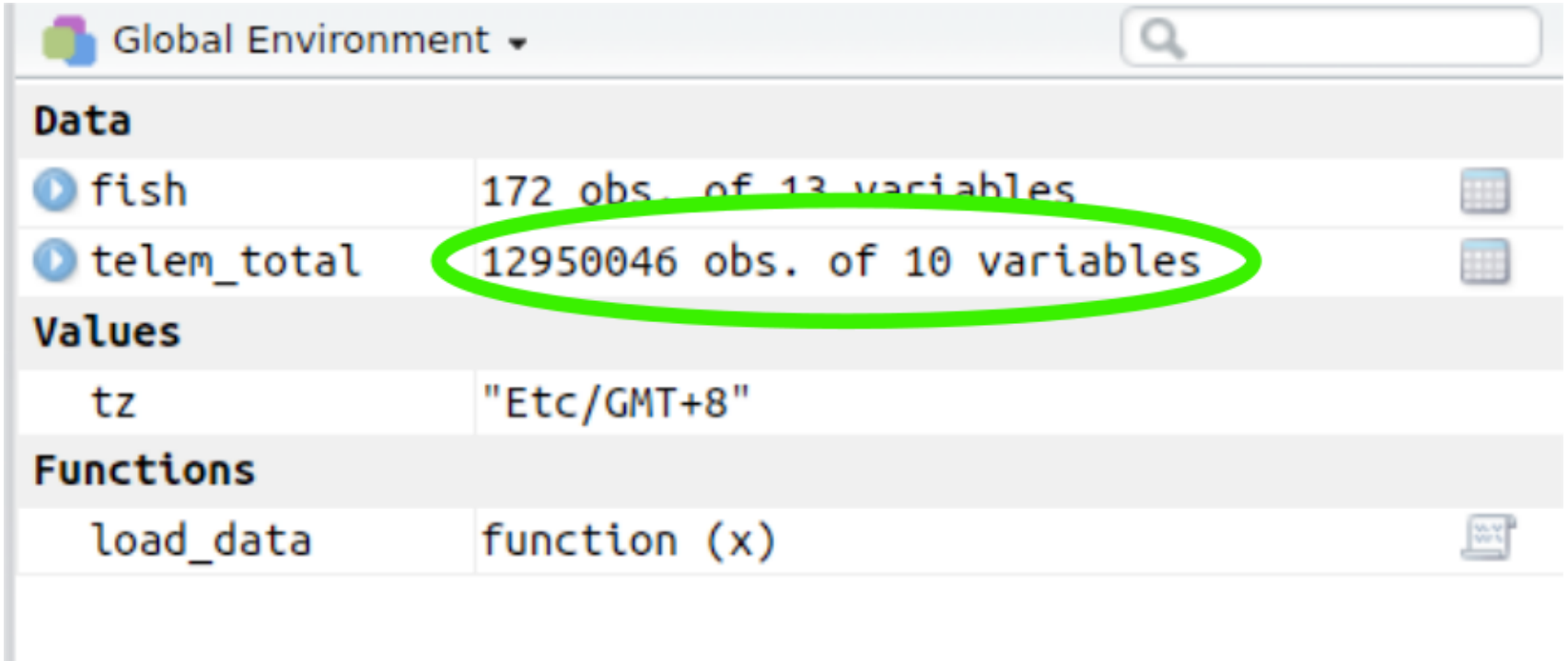
Why R?

R is hard

```
# Get in circle around city
circle <- data.frame()
cutoff <- 10
for(i in unique(gps$region)) {
  n <- nrow(gps[gps$region == i,])  ##number of IDs
  if(i == "wil") tmp <- geocode("Williams Lake, Canada")
  if(i == "kam") tmp <- geocode("Kamloops, Canada")
  if(i == "kel") tmp <- geocode("Kelowna, Canada")
  temp <- data.frame()
  for(a in 1:n){
    if(a <= cutoff) temp <- rbind(temp, gcDestination(lon = tmp$lon,
                                                      lat = tmp$lat,
                                                      bearing = (a*(360/(cutoff))-360/(cutoff)),
                                                      dist = 20,
                                                      dist.units = "km",
                                                      model = "WGS84"))
    if(a > cutoff) temp <- rbind(temp, gcDestination(lon = tmp$lon,
                                                     lat = tmp$lat,
                                                     bearing = ((a-cutoff)*(360/(max(table(gps$region
))-10))-360/(max(table(gps$region))-cutoff)),
                                                     dist = 35,
                                                     dist.units = "km",
                                                     model = "WGS84"))
  }
  circle <- rbind(circle, cbind(temp,
                                region = i,
                                hab = gps$hab[gps$region == i],
                                spl = gps$spl.orig[gps$region == i],
```

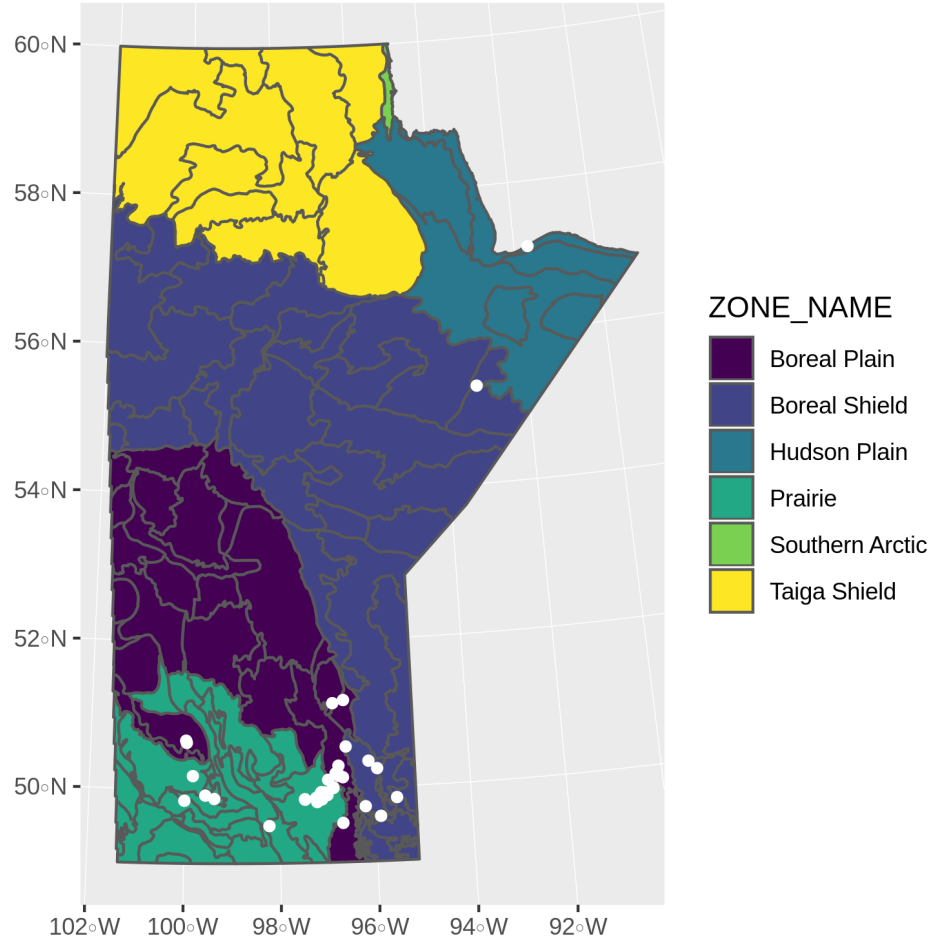
Why R?

But R is powerful (and reproducible)!



Why R?

R is also beautiful



Why R?

R is affordable (i.e. free!)

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

What is R?

R is Programming language

A programming **language** is a way to give instructions in order to get a computer to do something

- You need to know the language (i.e. the code)
- Computers don't know what you mean, only what you type (unfortunately)
- Spelling, punctuation, and capitalization all matter!

For example

R, what is 56 times 5.8?

```
56 * 5.8
```

```
## [1] 324.8
```

Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

R, save this value for later

```
steffis_mean <- mean(c(1, 2, 3, 4))
```

R, multiply this value by 6

```
steffis_mean * 6
```

```
## [1] 15
```

Code, Output, Scripts

Code

- The actual commands

Output

- The result of running code or a script

Script

- A text file full of code that you want to run
- You should **always** keep your code in a script

For example:

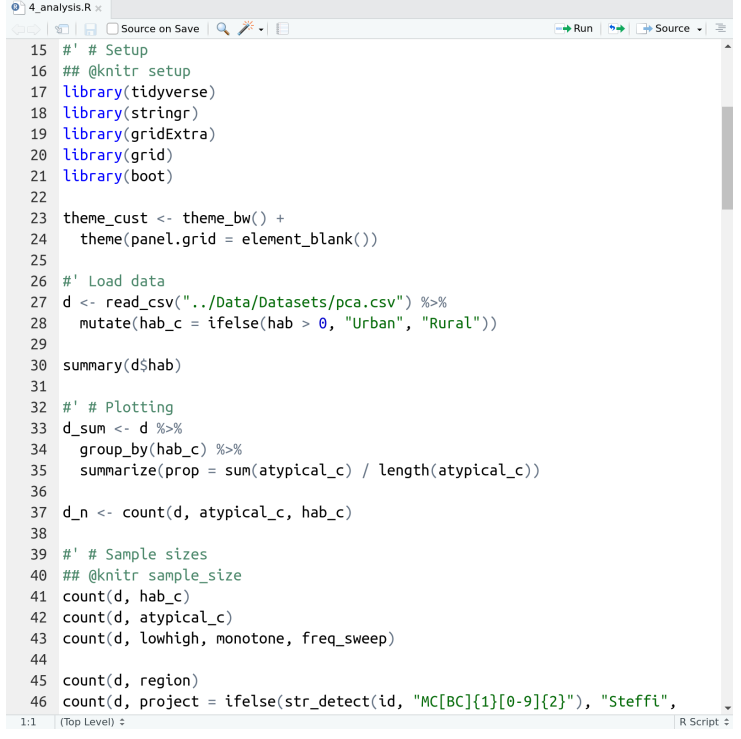
```
mean(c(1, 2, 3, 4))
```

Code

```
## [1] 2.5
```

Output

Script



RStudio vs. R

RStudio

R

- **RStudio** is not **R**
- RStudio is a User Interface or IDE (integrated development environment)
  - (i.e., Makes coding simpler)

R Basics: Objects

Objects are *things* in the environment

(Check out the **Environment** pane in RStudio)

`functions()` - Do things, Return things

`mean()`, `read_csv()`, `ggplot()`, `c()`, etc.

- Always have `()`
- Can take **arguments** (think 'options')
  - `mean(x = c(2, 10, 45))`,
  - `mean(x = c(NA, 10, 2, 65), na.rm = TRUE)`
- Arguments defined by **name** or by **position**
- With correct position, do not need to specify by name

By name:

```
mean(x = c(1, 5, 10))
```

```
## [1] 5.333333
```

By position:

```
mean(c(1, 5, 10))
```

```
## [1] 5.333333
```

R documentation

```
?mean
```

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
...	further arguments passed to or from other methods.

Data

Generally kept in **`vectors`** or **`data.frames`**

- These are objects with names (like functions)
- We can use **`<-`** to assign values to objects (assignment)

Vector (1 dimension)

```
my_data <- c("a", 100, "c")
my_data
```

```
## [1] "a"   "100" "c"
```

Data frame (2 dimensions)

rows x columns

```
my_data <- data.frame(site = c("s1", "s2", "s3"),
                      count = c(101, 102, 103),
                      treatment = c("a", "b", "c"))
my_data
```

```
##   site count treatment
## 1   s1   101         a
## 2   s2   102         b
## 3   s3   103         c
```

R Basics: Code

Your first code

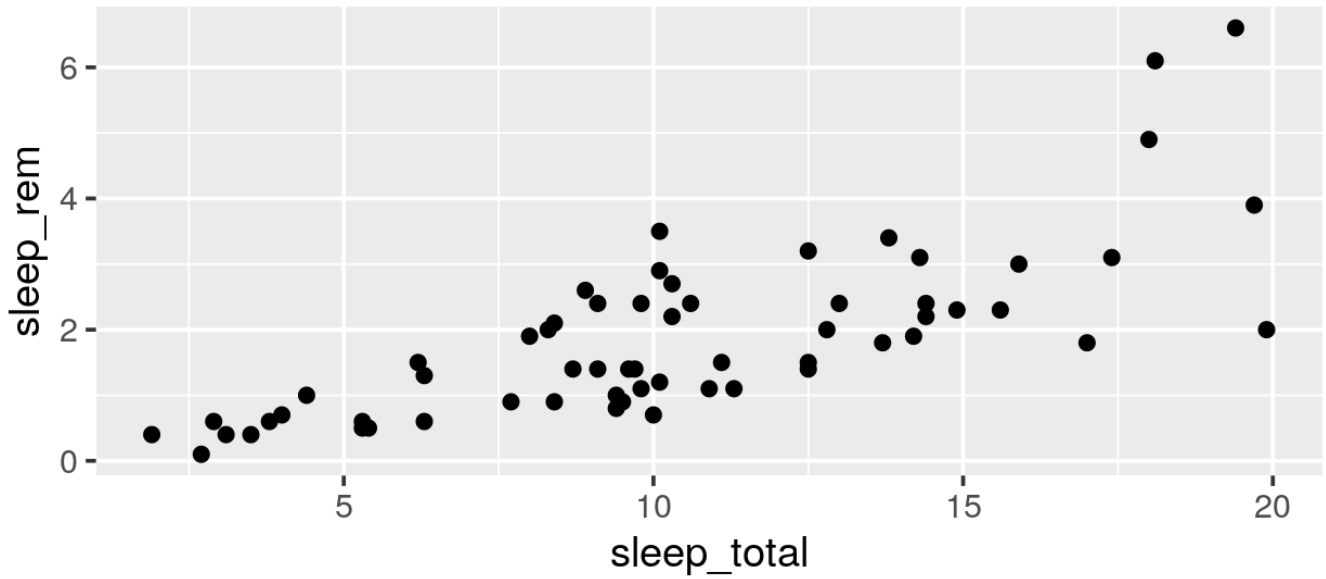
```
# First load the package and data
library(tidyverse)
sleep <- read_csv("https://git.io/Je6DF")

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

- Copy/paste or type this into the script window in RStudio
- Click anywhere on the first line of code
- Use the 'Run' button to run this code, **or** use the short-cut **`Ctrl-Enter`**
    - Repeat until all the code has run

First Code

```
# First load the package and data
library(tidyverse)
sleep <- read_csv("ht

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

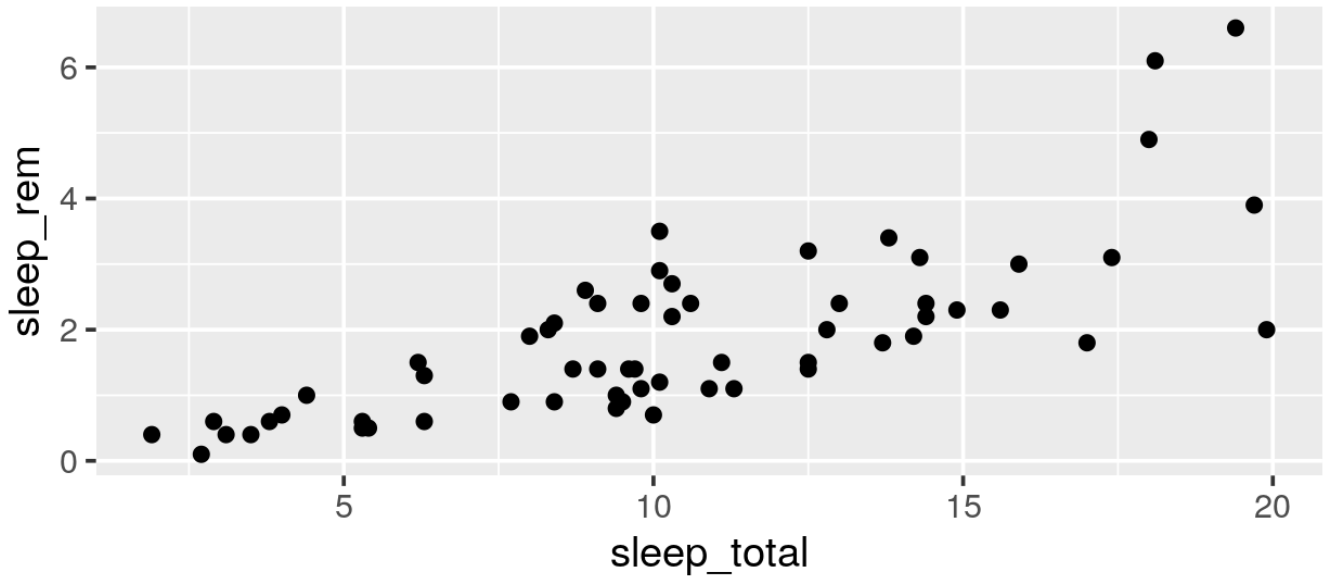
Package
tidyverse



First Code

```
# First load the package and data
library(tidyverse)
sleep <- read_csv("ht

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

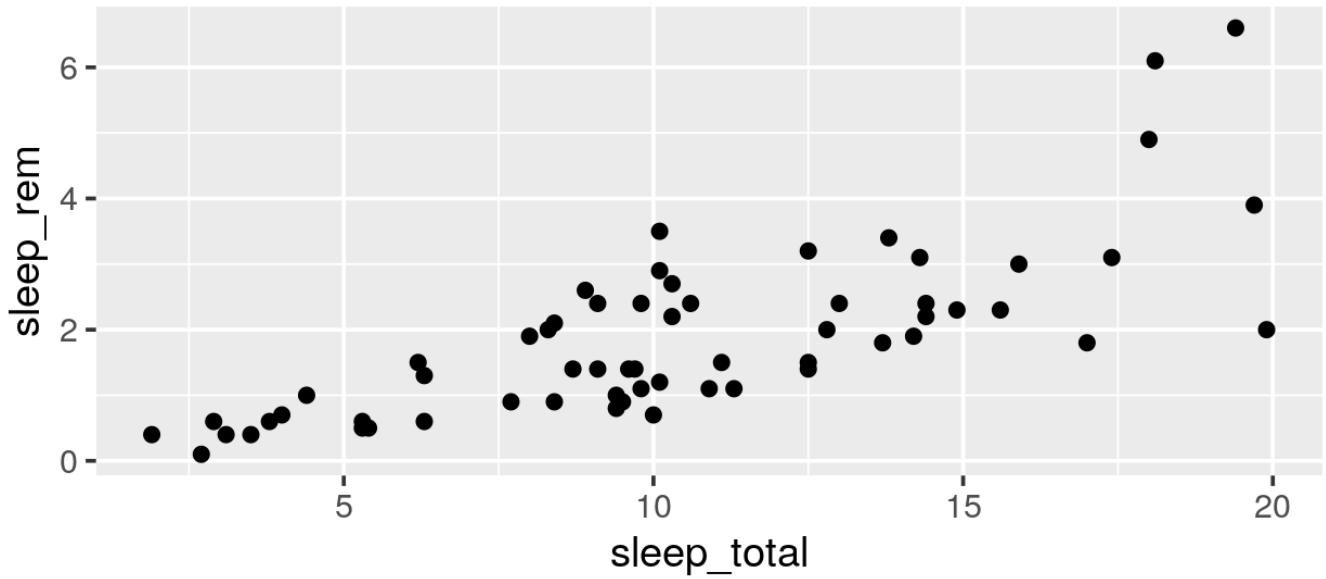
Function
library()



First Code

```
# First load the package and data
library(tidyverse)
sleep <- read_csv("https://git.io/Je6DF")

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

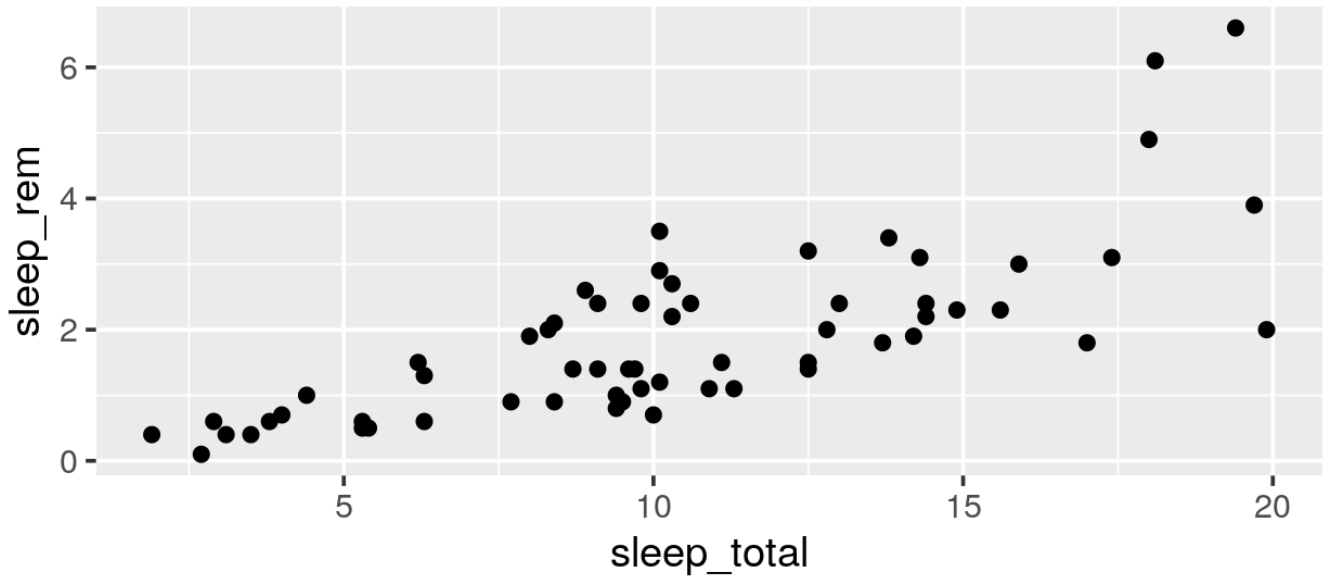
Function
read_csv()



First Code

```
# First load the package and data
library(tidyverse)
sleep <- read_csv("https://git.io/Je6DF")

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

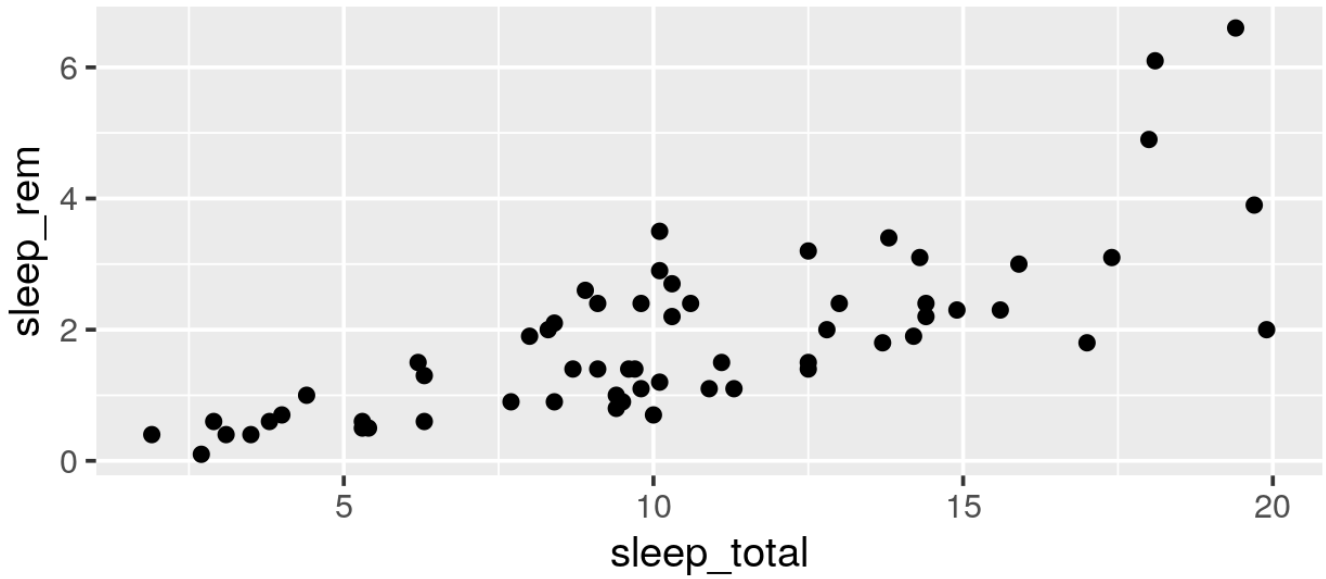
Function
ggplot()



First Code

```
# First load the package and data
library(tidyverse)
sleep <- read_csv("https://git.io/Je6DF")

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

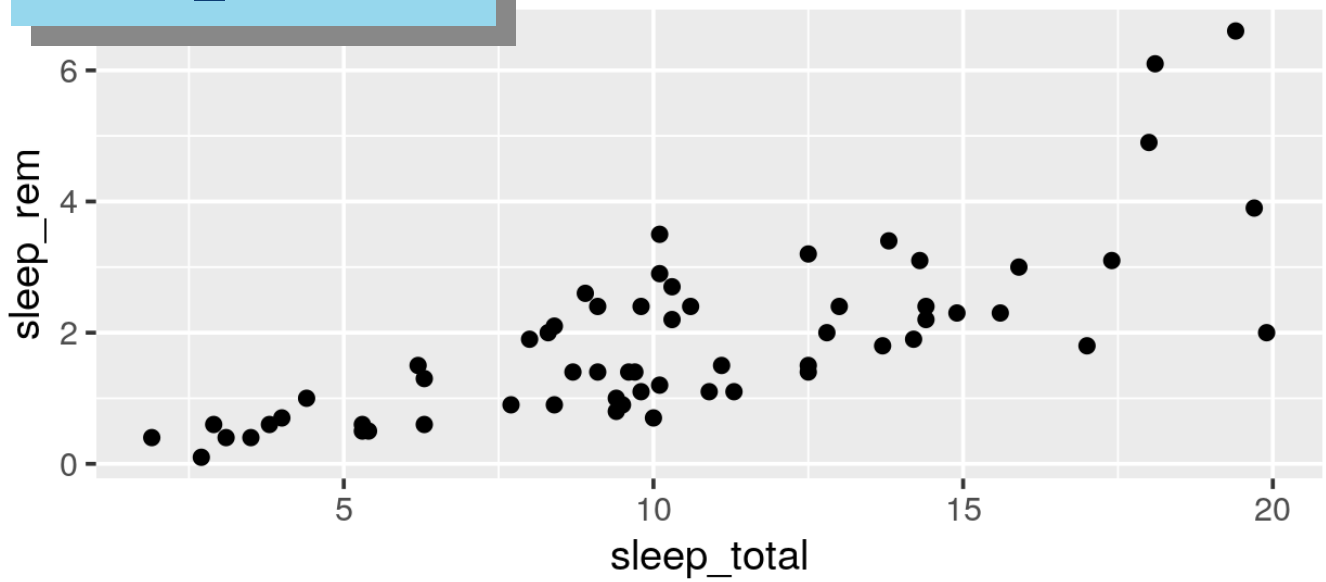
Function
aes()



First Code

```
# First load the package and data
library(tidyverse)
sleep <- read_csv("https://git.io/Je6DF")

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

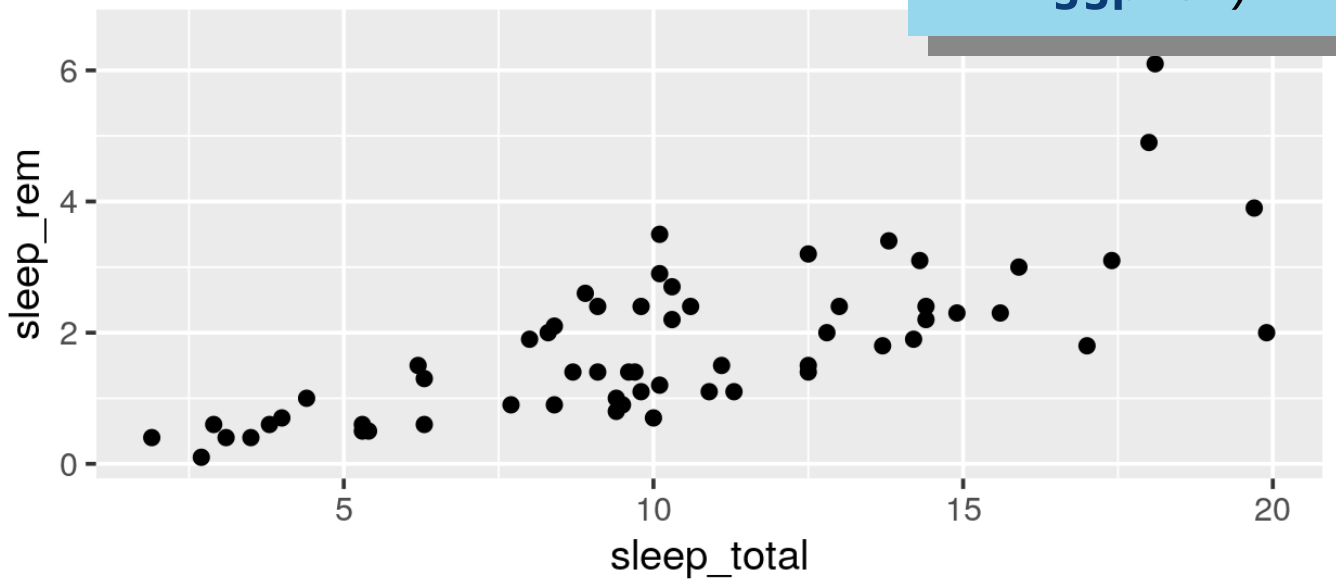
Function
geom_point()



First Code

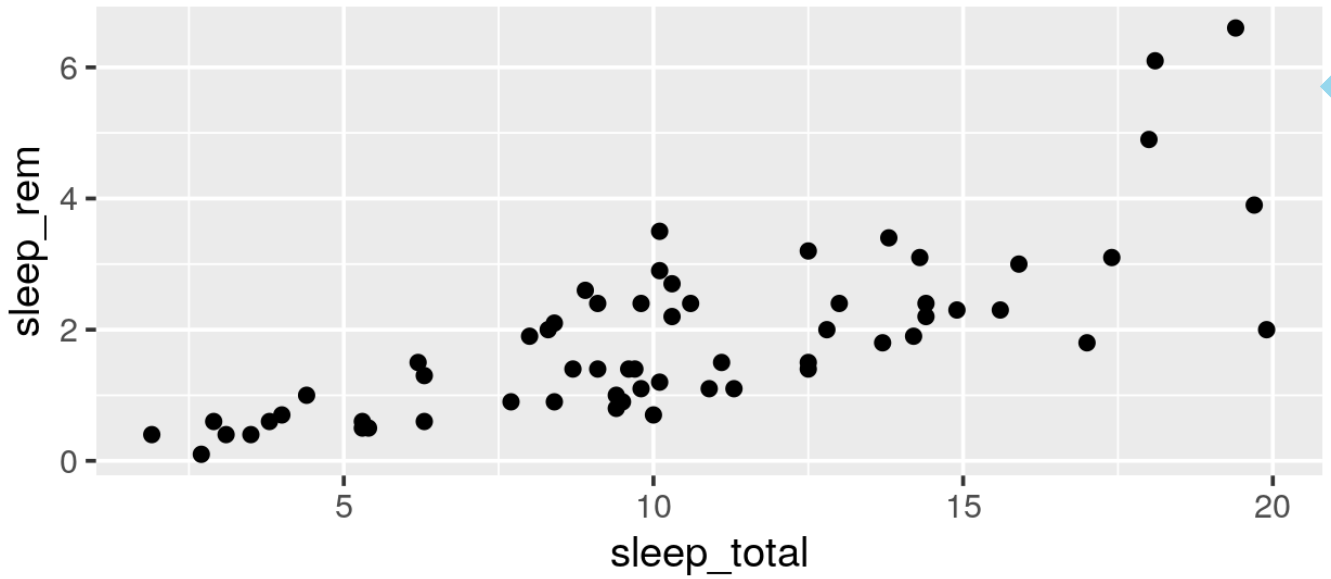
```
# First load the package and data
library(tidyverse)
sleep <- read_csv("https://git.io/Je6DF")

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

+
(Specific to **ggplot**)



First Code

```
# First load the package and data
library(tidyverse)
sleep <- read_csv("https://git.io/Je6DF")

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

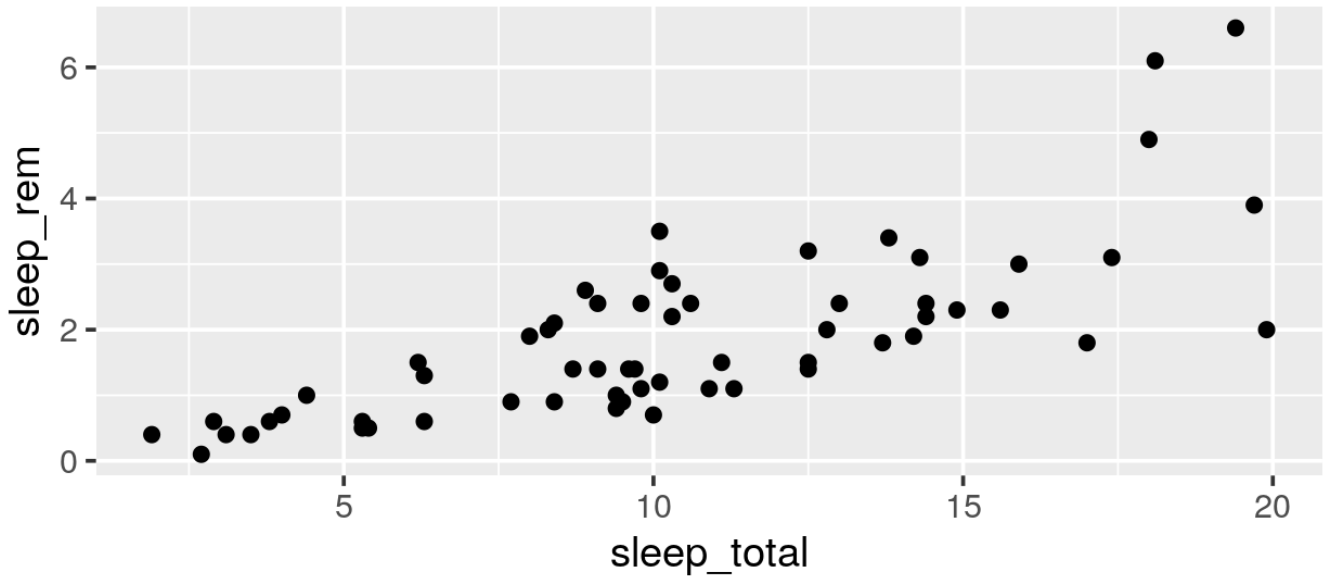


Figure!

First Code

```
# First load the package
library(tidyverse)
sleep <- read_csv("https://git.io/Je6DF")

# Now create the figure
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

Comments



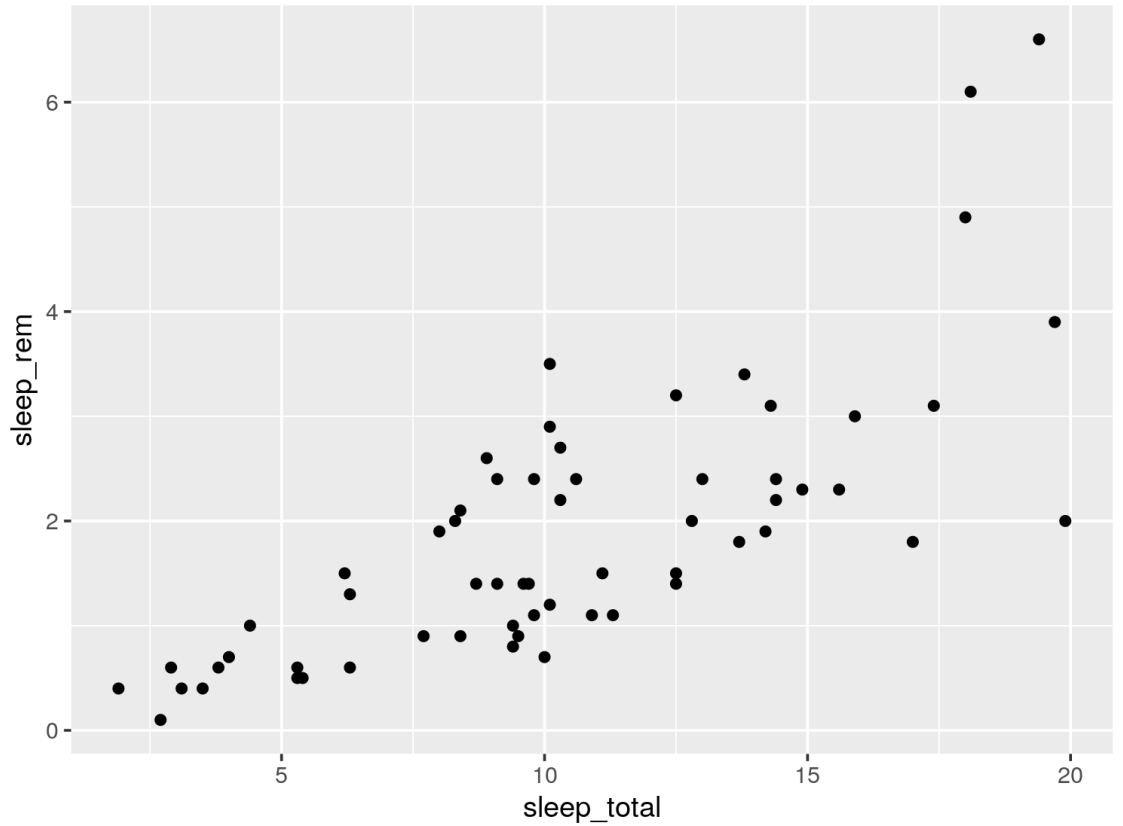
Hooray!

Time for Figures!

@allison_horst

A basic plot

```
library(ggplot2) # Also inside the tidyverse
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```



Break it down

```
library(ggplot2) # Also inside the tidyverse
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

`ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem))`

- Load the **`ggplot2`** (or **`tidyverse`**) package
- Set the attributes of your plot
- **`data`** = Dataset
- **`aes`** = Aesthetics (how the data are used)
- Think of this as your plot defaults

Break it down

```
library(ggplot2) # Also inside the tidyverse
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
    geom_point()
```

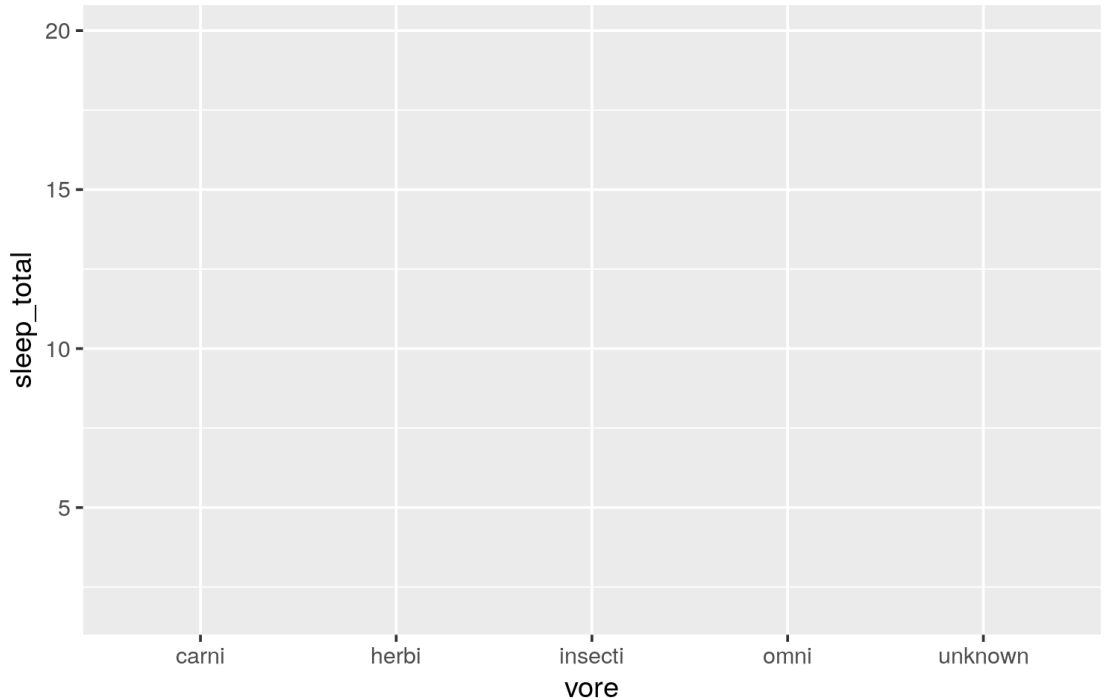
`geom_point()`

- Choose a **`geom`** function to display the data
- Always *added* to a **`ggplot()`** call with **+**

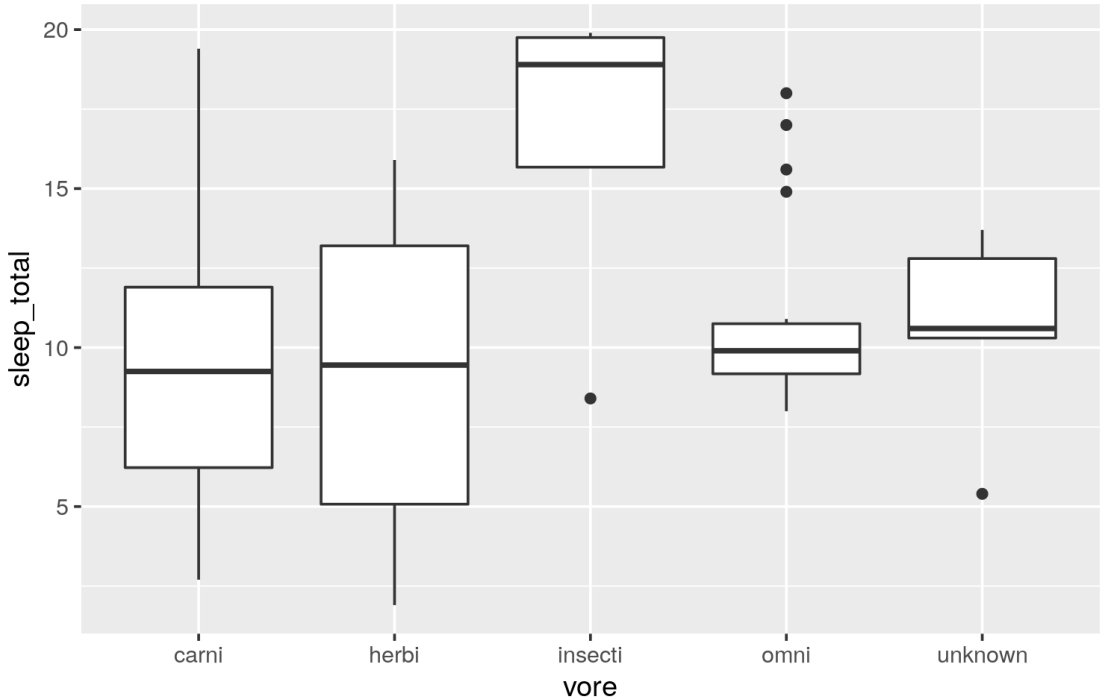
ggplots are essentially layered objects, starting with a call to **`ggplot()`**

Plots are layered

```
ggplot(data = sleep, aes(x = vore, y = sleep_total))
```

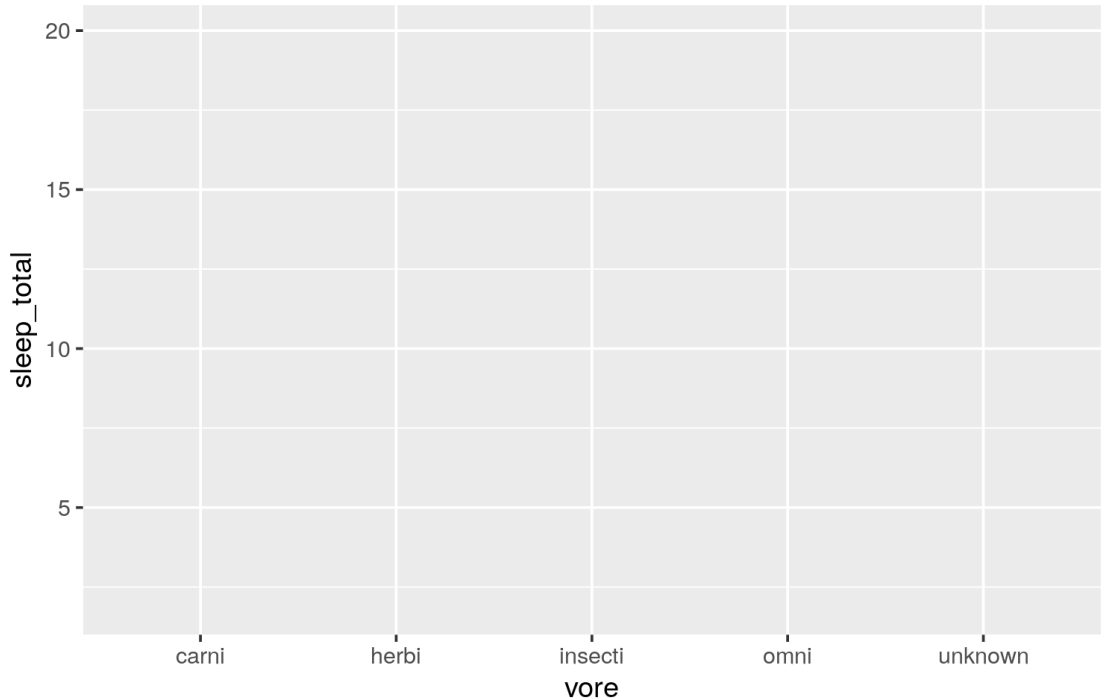



```
ggplot(data = sleep, aes(x = vore, y = sleep_total)) +
  geom_boxplot()
```

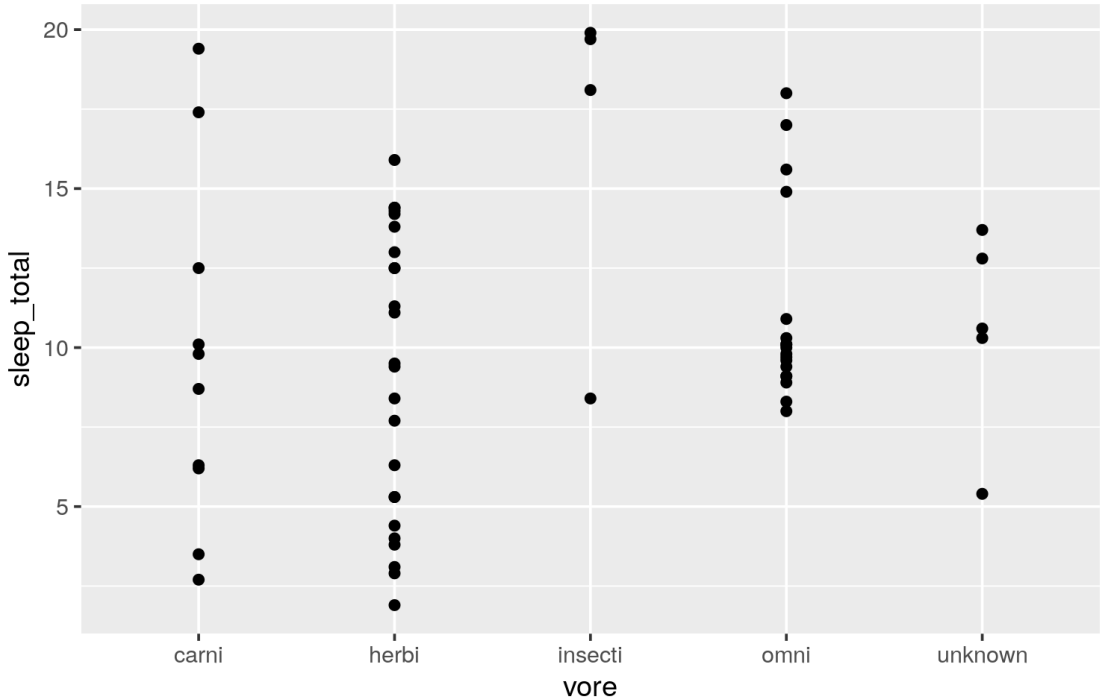


Plots are layered

```
ggplot(data = sleep, aes(x = vore, y = sleep_total))
```

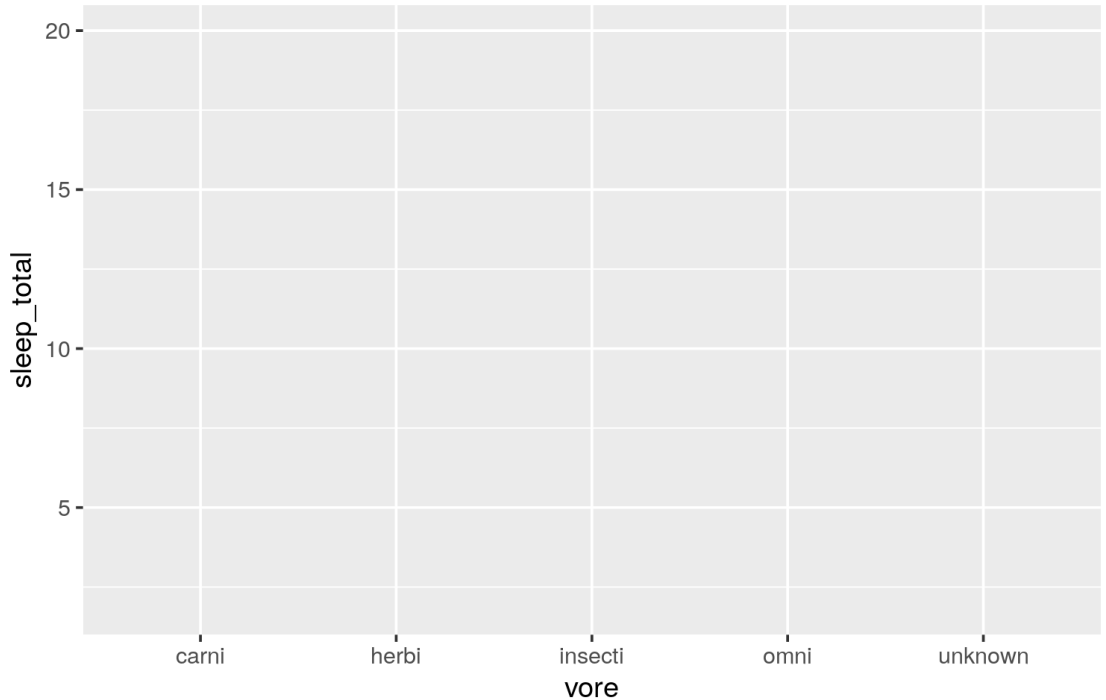



```
ggplot(data = sleep, aes(x = vore, y = sleep_total)) +
  geom_point()
```

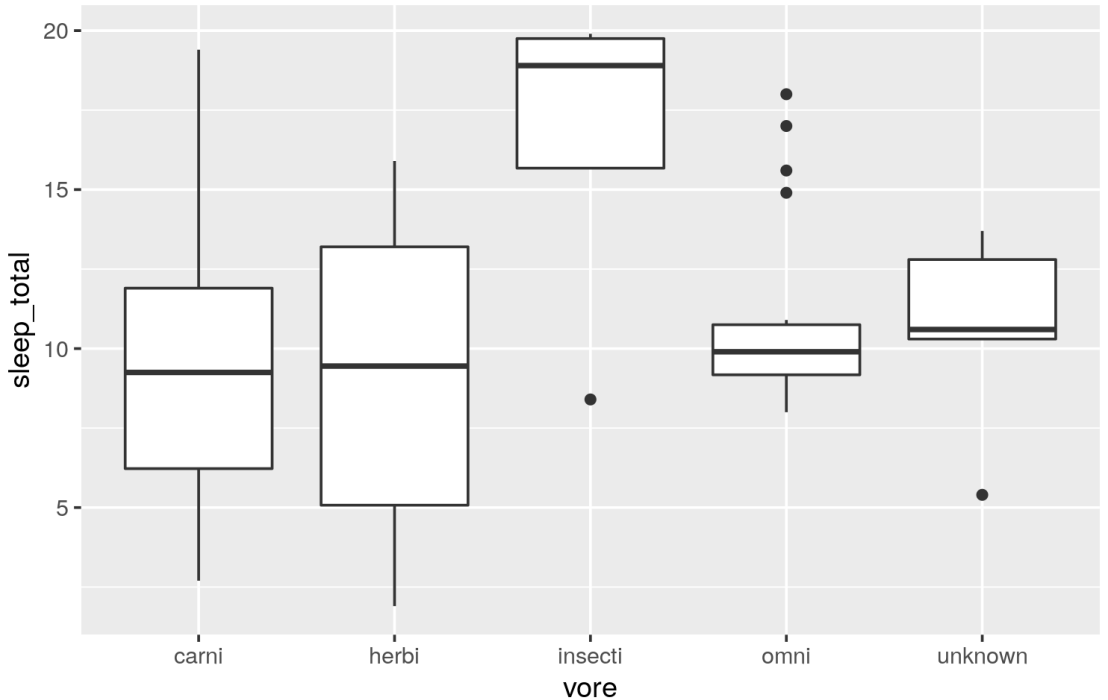


Plots are layered

```
ggplot(data = sleep, aes(x = vore, y = sleep_total))
```

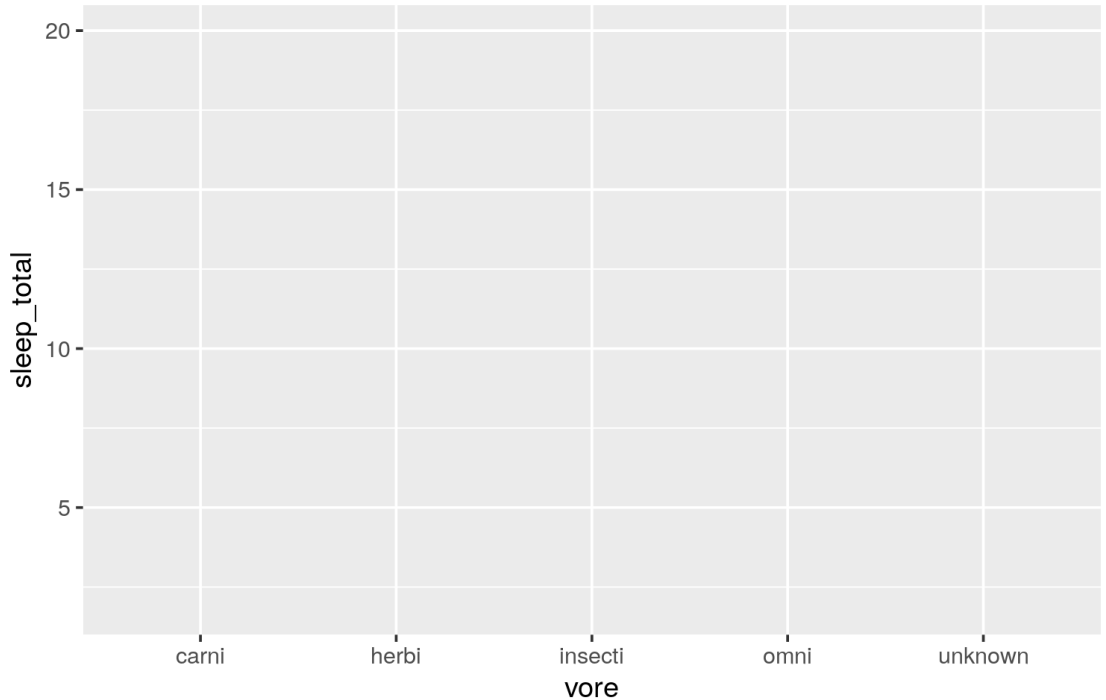



```
ggplot(data = sleep, aes(x = vore, y = sleep_total)) +
  geom_boxplot()
```

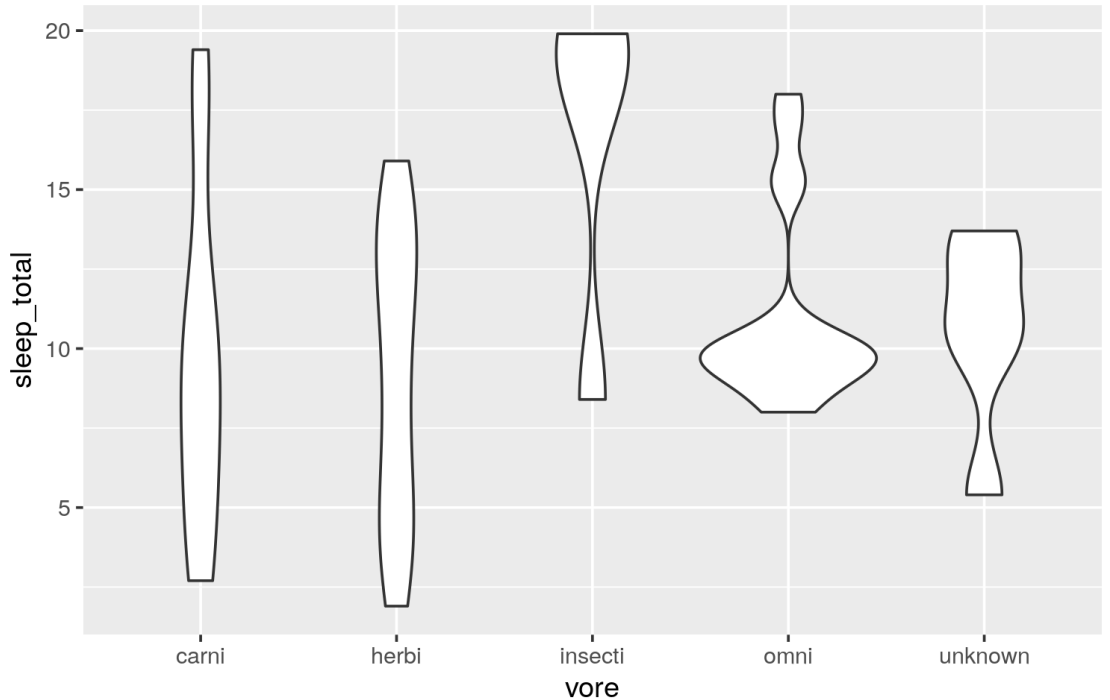


Plots are layered

```
ggplot(data = sleep, aes(x = vore, y = sleep_total))
```




```
ggplot(data = sleep, aes(x = vore, y = sleep_total)) +
  geom_violin()
```

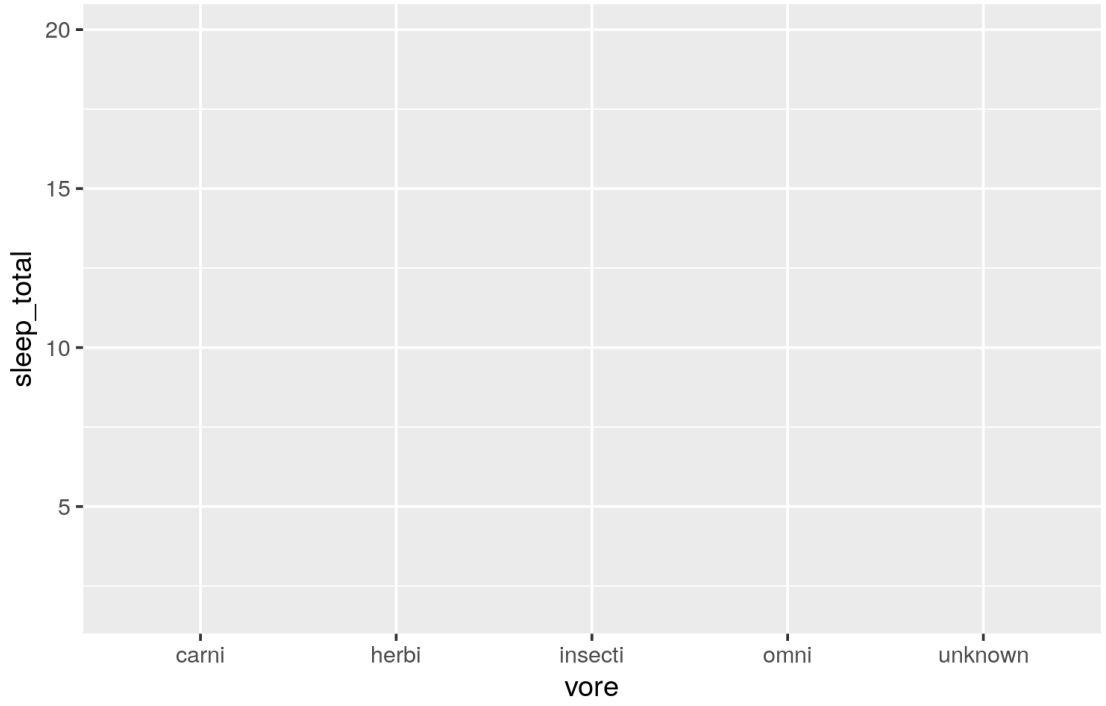


Plots are objects

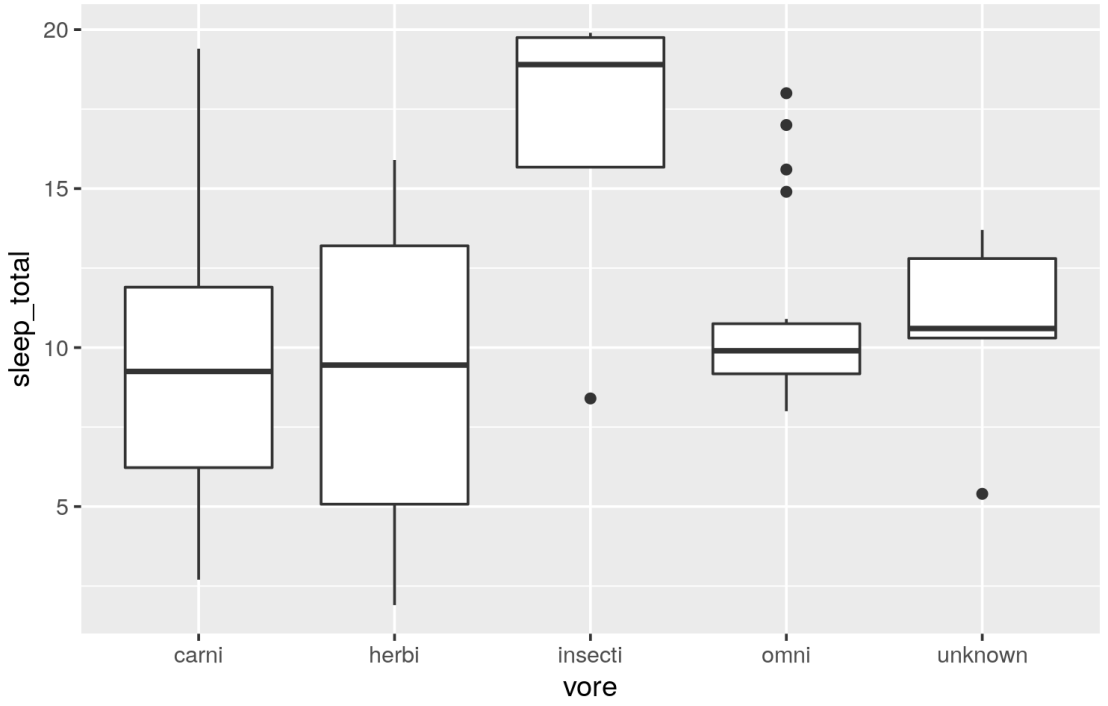
Any ggplot can be saved as an object

```
g <- ggplot(data = sleep, aes(x = vore, y = sleep_total))
```

```
g
```



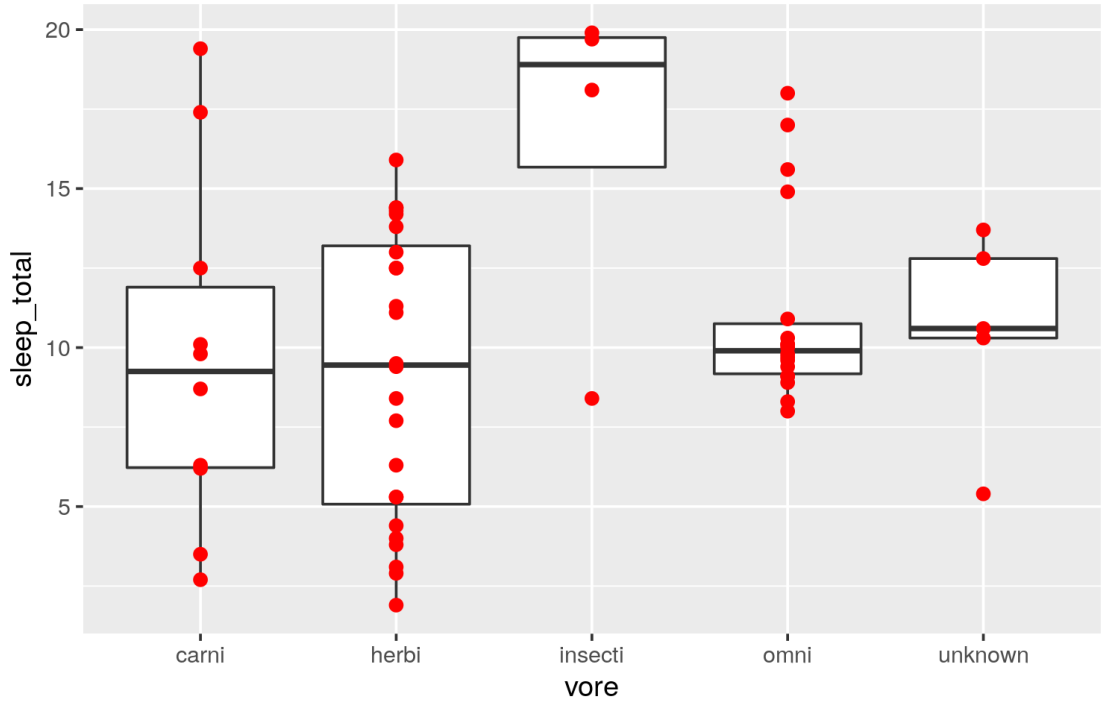

```
g + geom_boxplot()
```



Layering

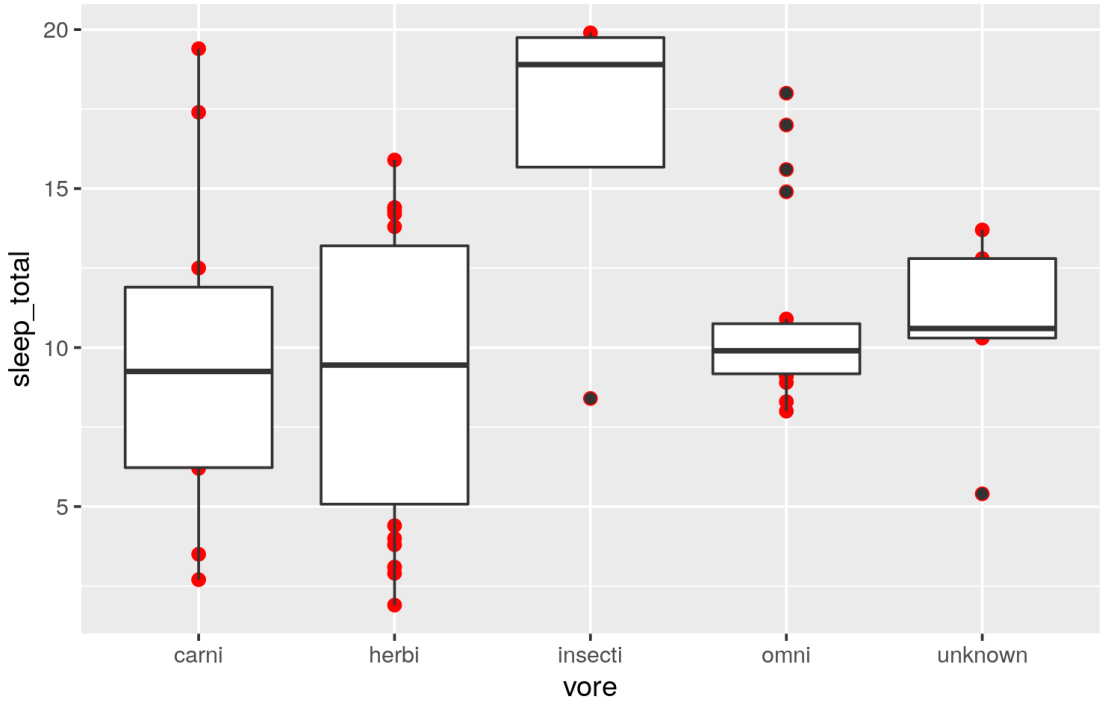
You can add multiple layers

```
g +
  geom_boxplot() +
  geom_point(size = 2, colour = "red")
```



Order matters

```
g +
  geom_point(size = 2, colour = "red") +
  geom_boxplot()
```

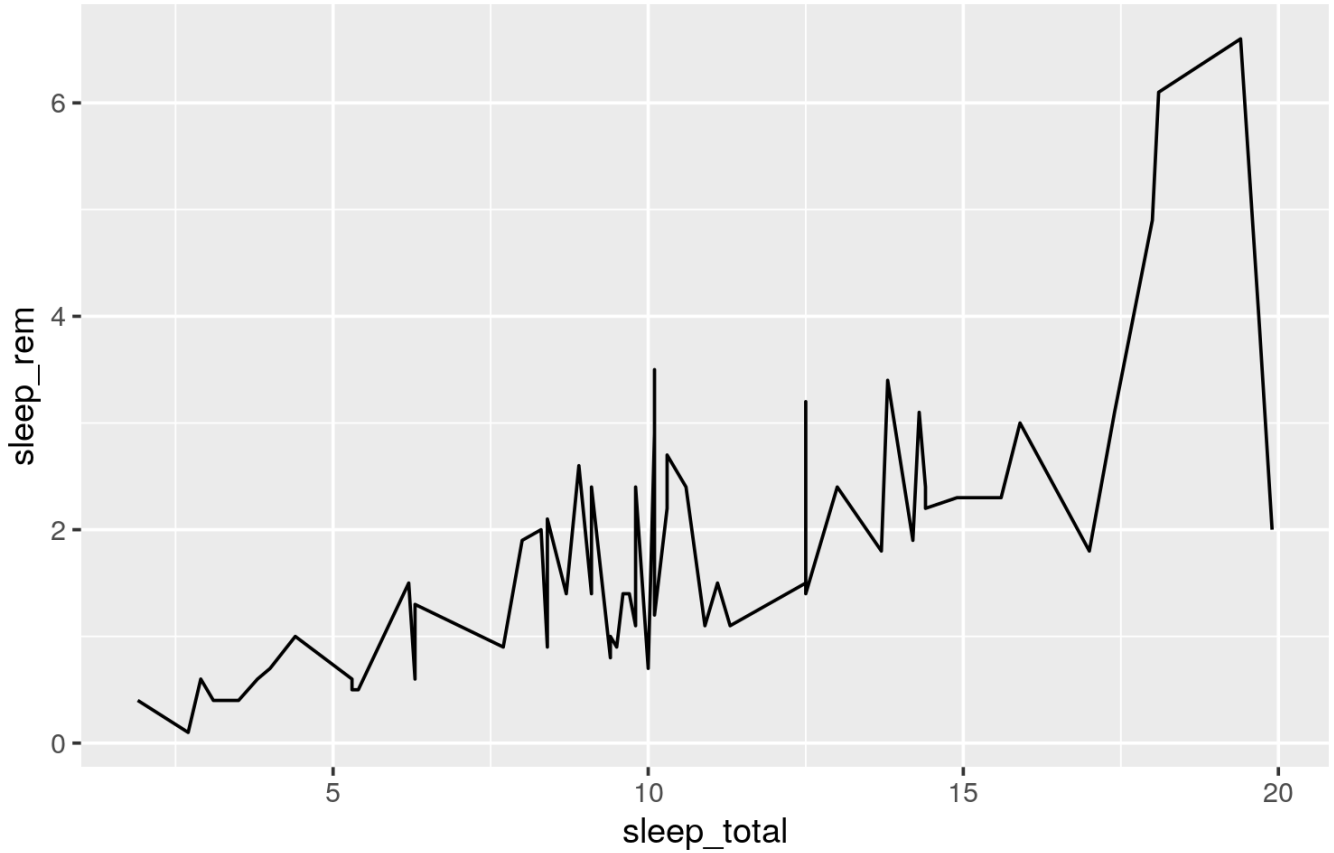


More Geoms

(Plot types)

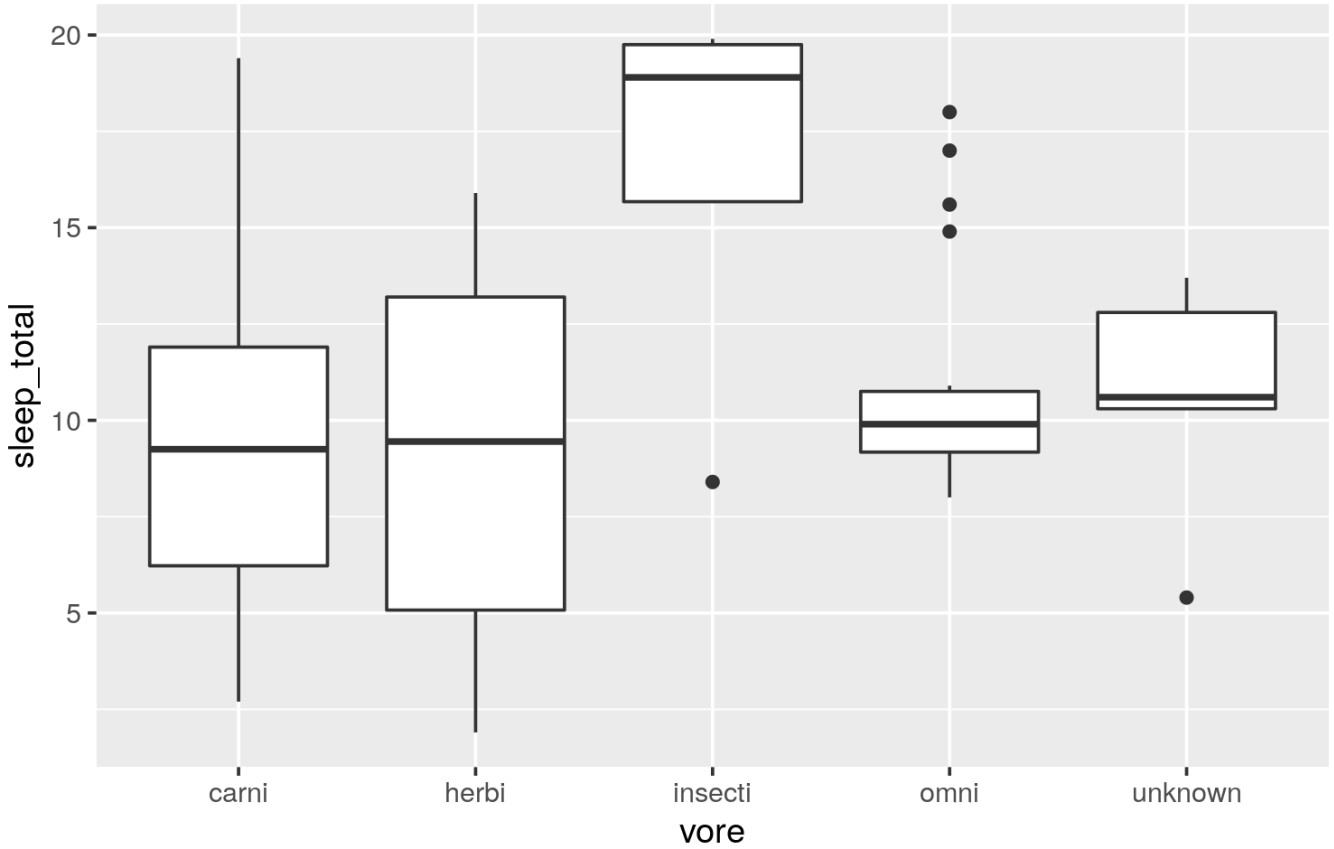
Geoms: Lines

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
  geom_line()
```



Geoms: Boxplots

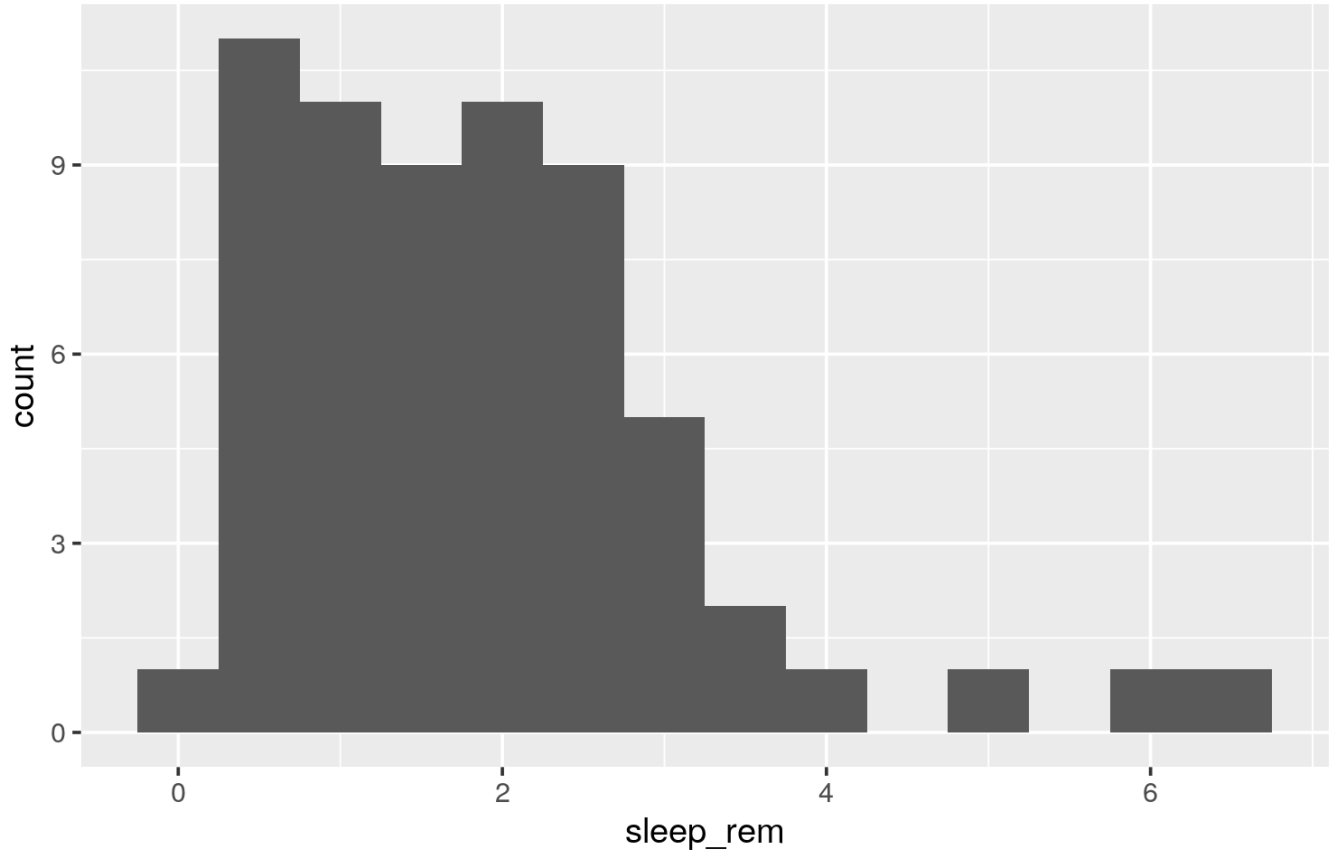
```
ggplot(data = sleep, aes(x = vore, y = sleep_total)) +
  geom_boxplot()
```



Geoms: Histogram

```
ggplot(data = sleep, aes(x = sleep_rem)) +
  geom_histogram(binwidth = 0.5)
```

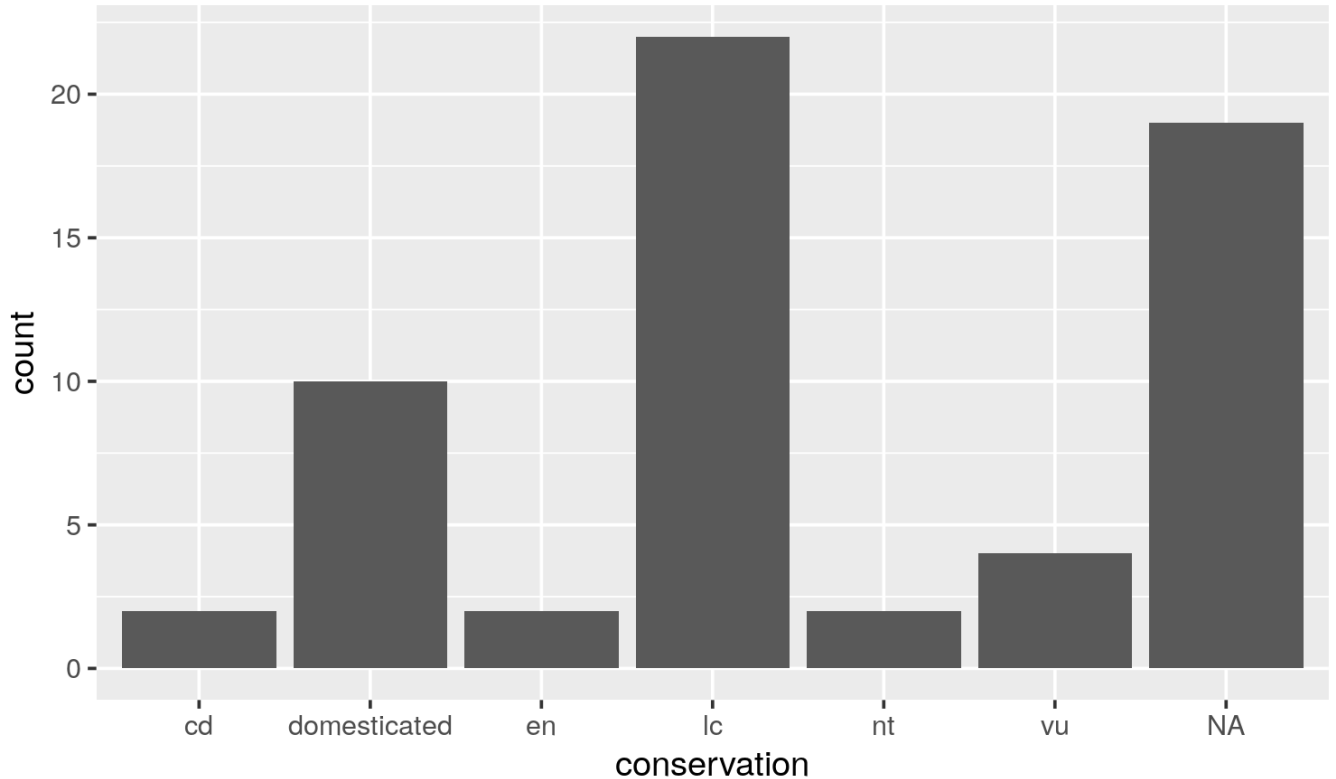
Note: We only need 1 aesthetic here (x)



Geoms: Barplots

Let `ggplot` count your data

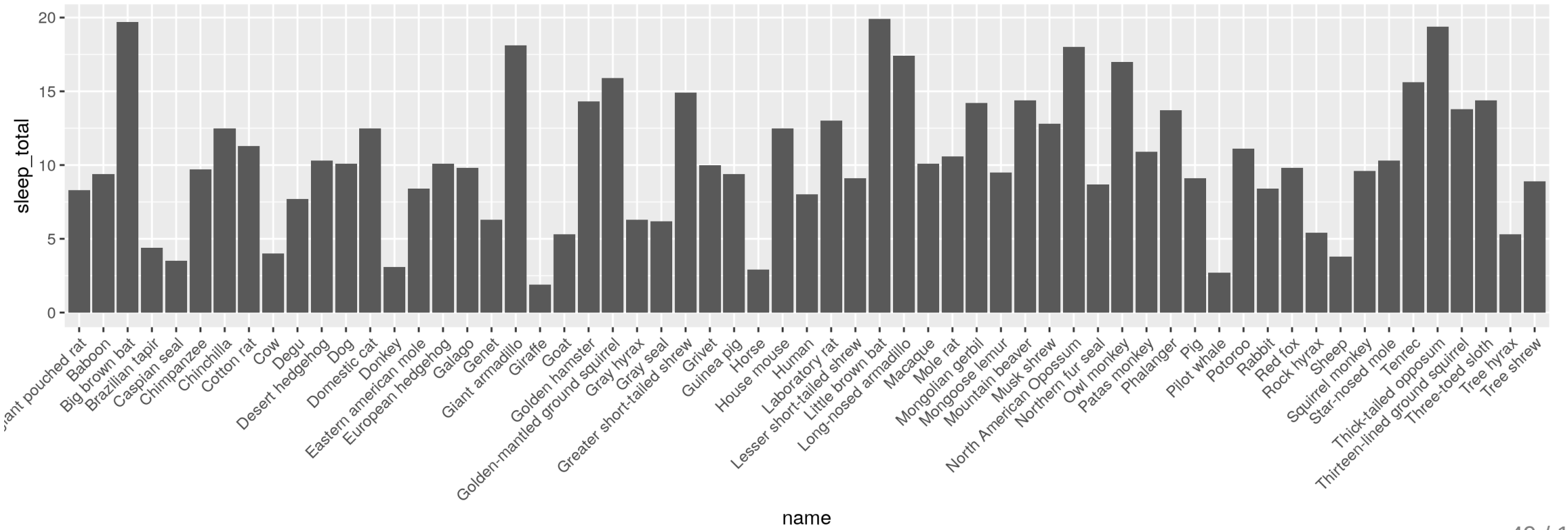
```
ggplot(data = sleep, aes(x = conservation)) +
  geom_bar()
```



Geoms: Barplots

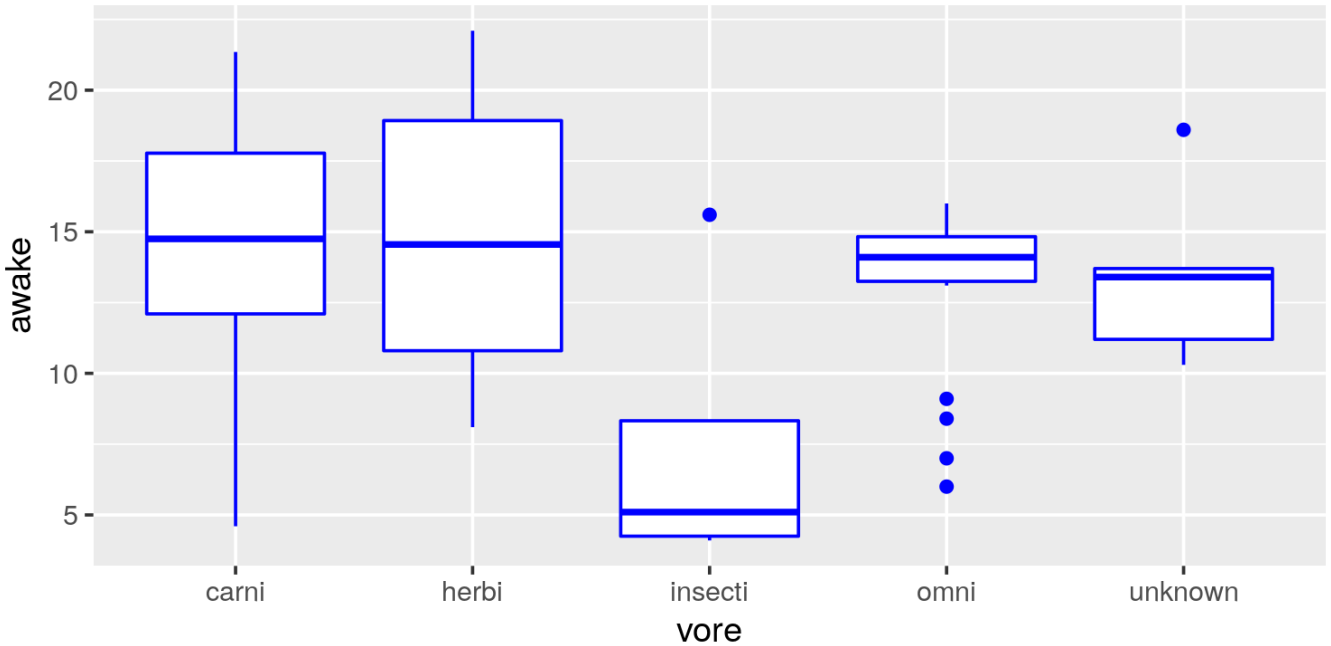
You can also provide the counts

```
ggplot(data = sleep, aes(x = name, y = sleep_total)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 45, hjust =1))
```



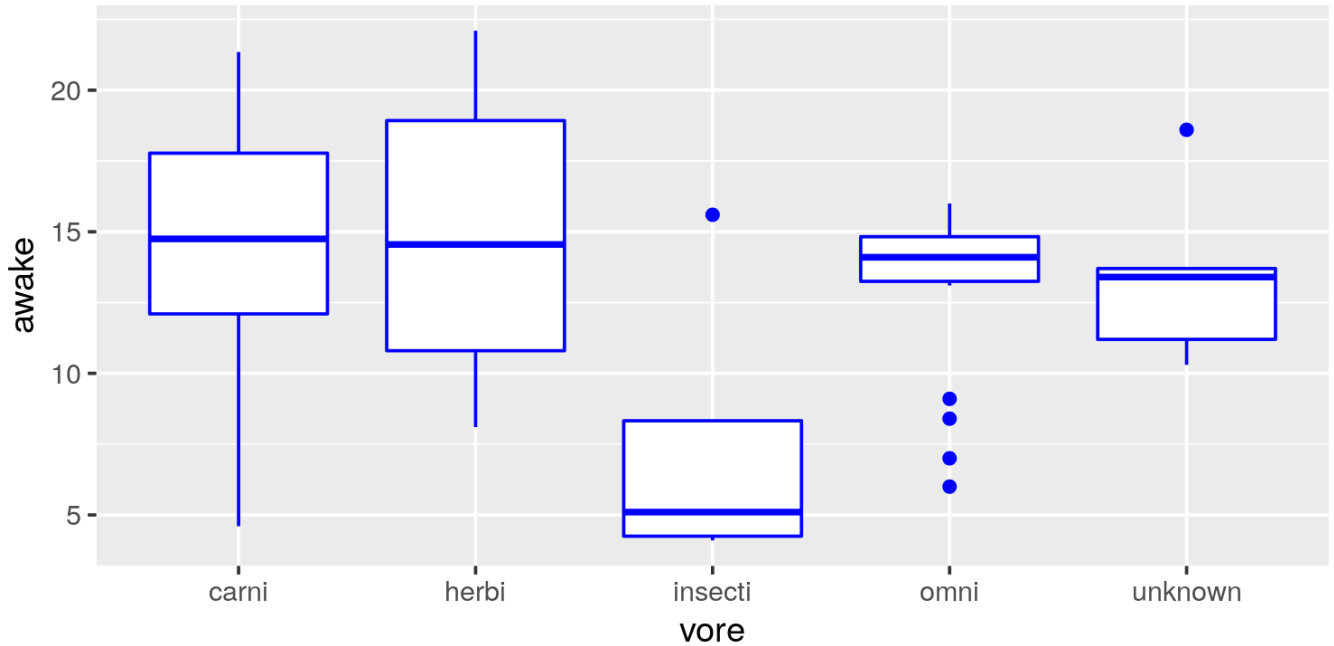
Your Turn: Create this plot

```
library(ggplot2)

ggplot(data = ???, aes(x = ???, y = ???)) +
  geom_???(???)
```



Your Turn: Create this plot

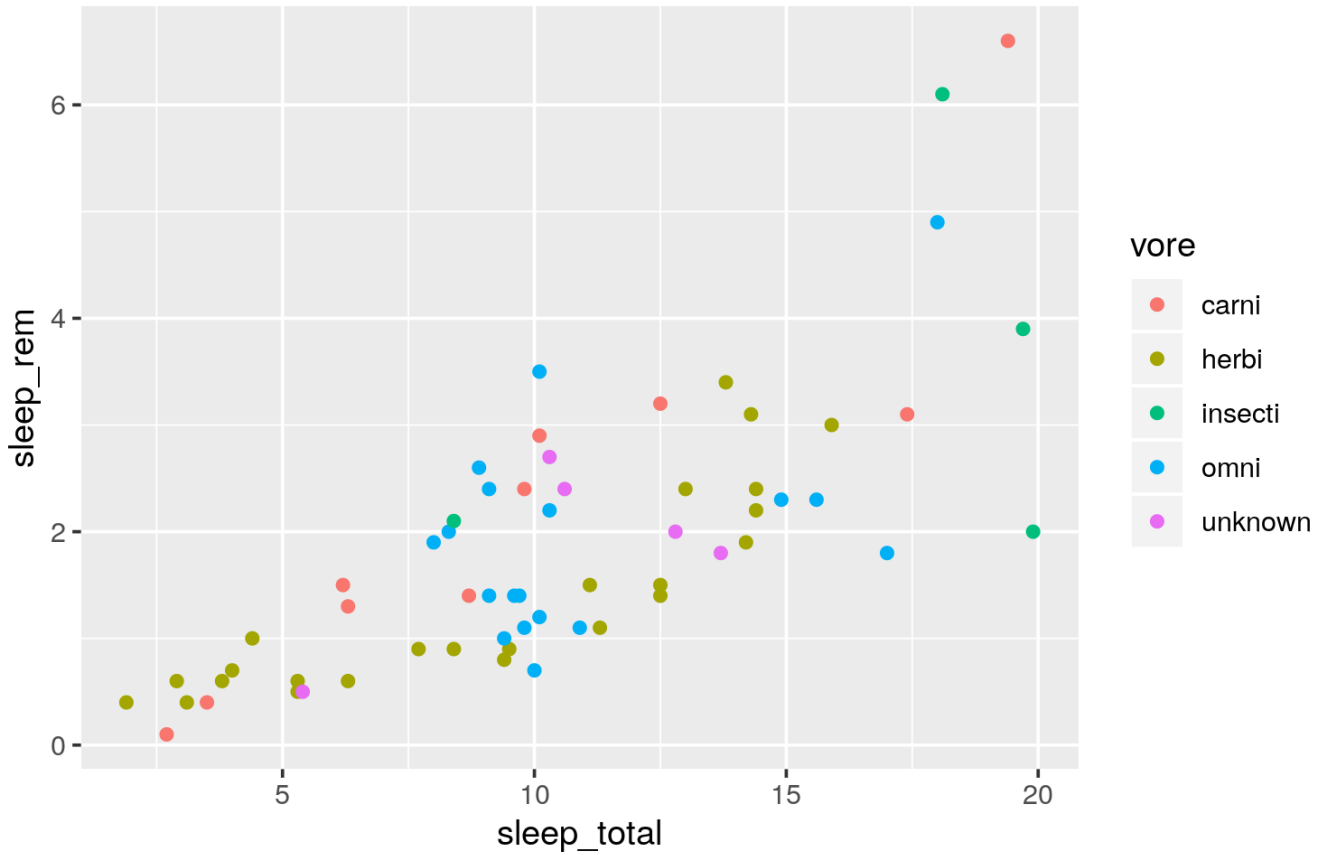
```
library(ggplot2)

ggplot(data = sleep, aes(x = vore, y = awake)) +
  geom_boxplot(colour = "blue")
```



Showing data by group

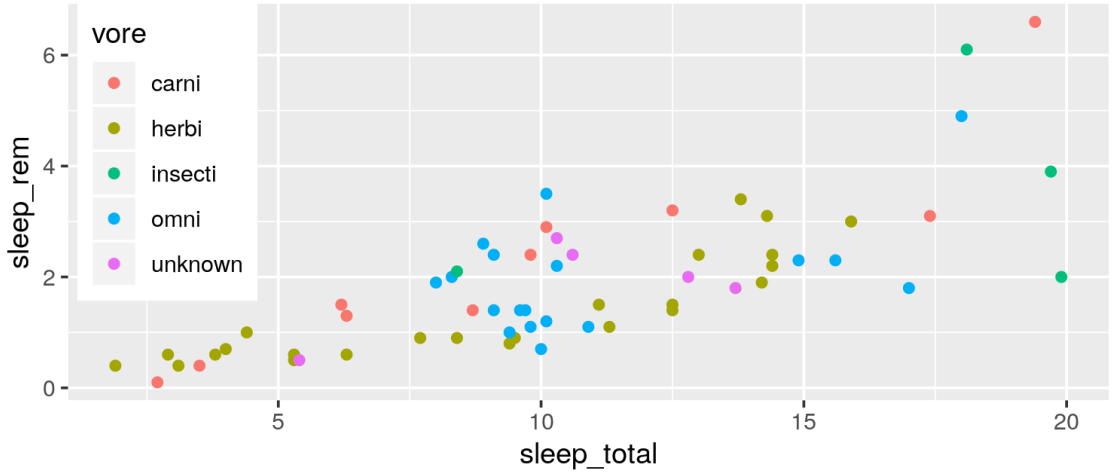
Mapping aesthetics

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point()
```

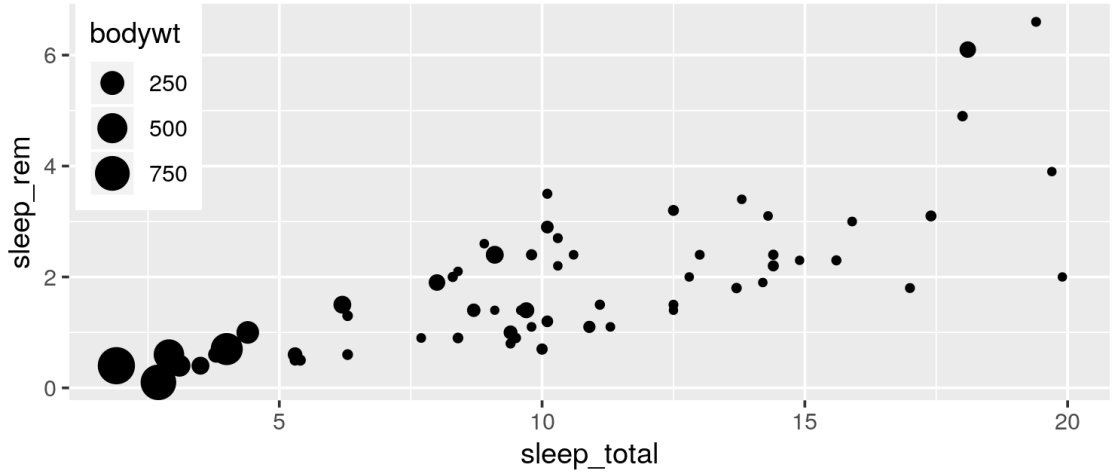


Mapping aesthetics

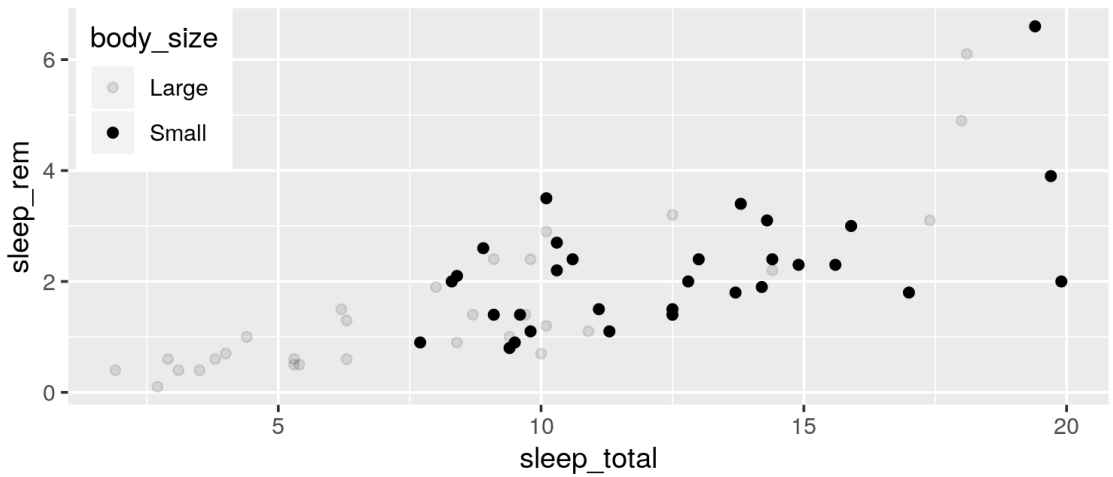
colour = vore



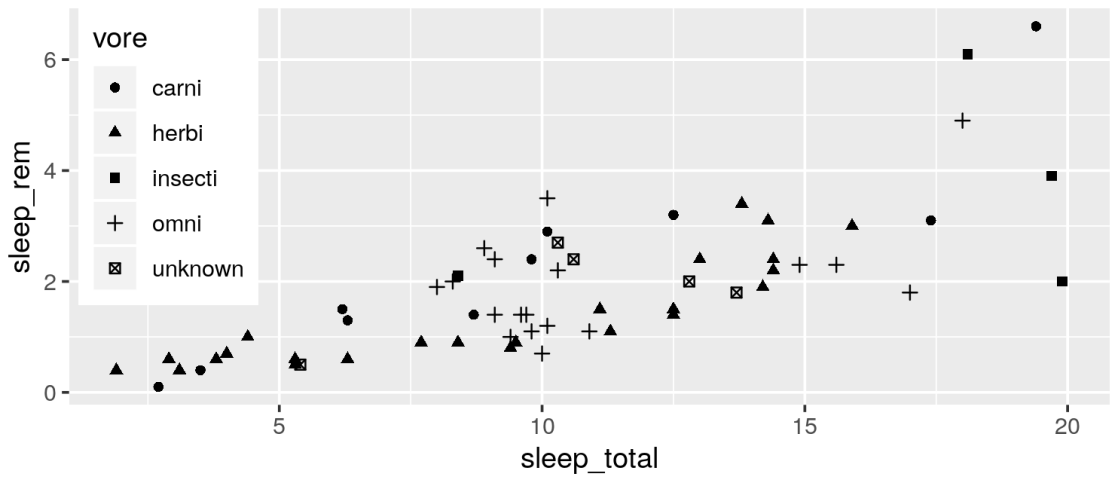
size = bodywt



alpha = body_size



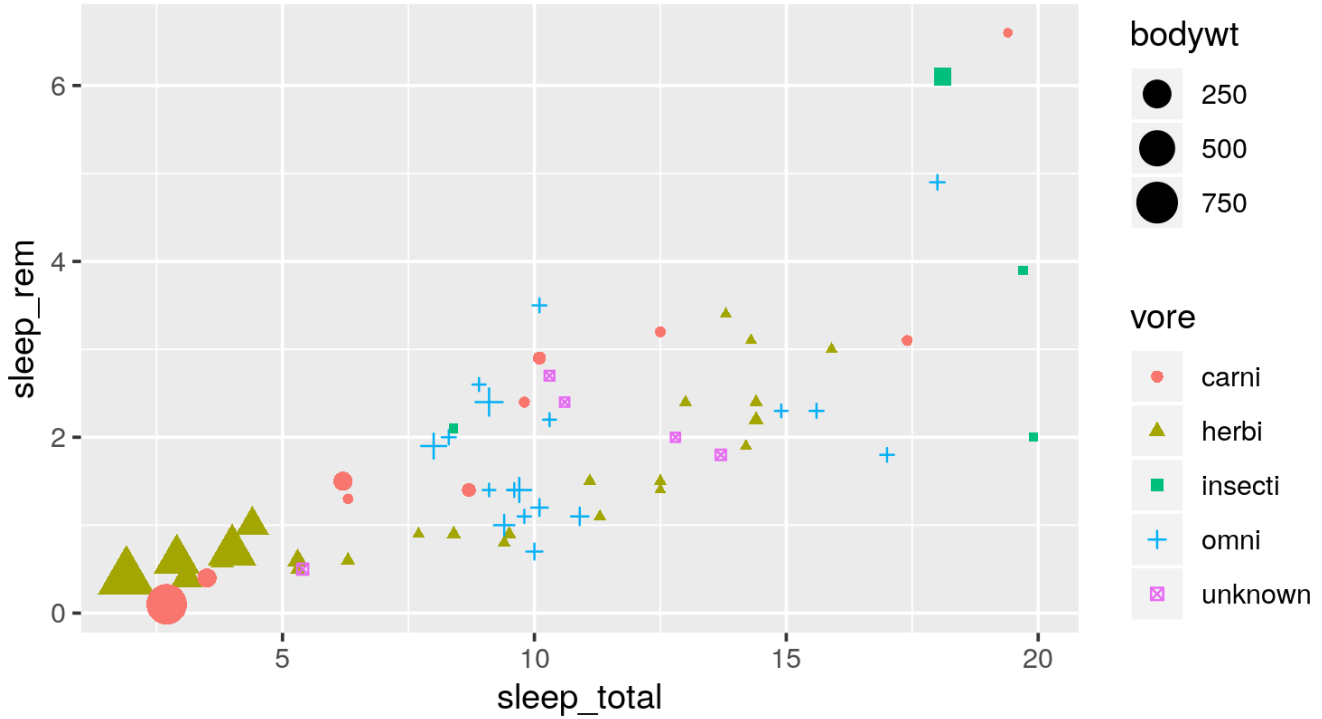
shape = vore



Mapping aesthetics

`ggplot` automatically populates the legends (combining where it can)

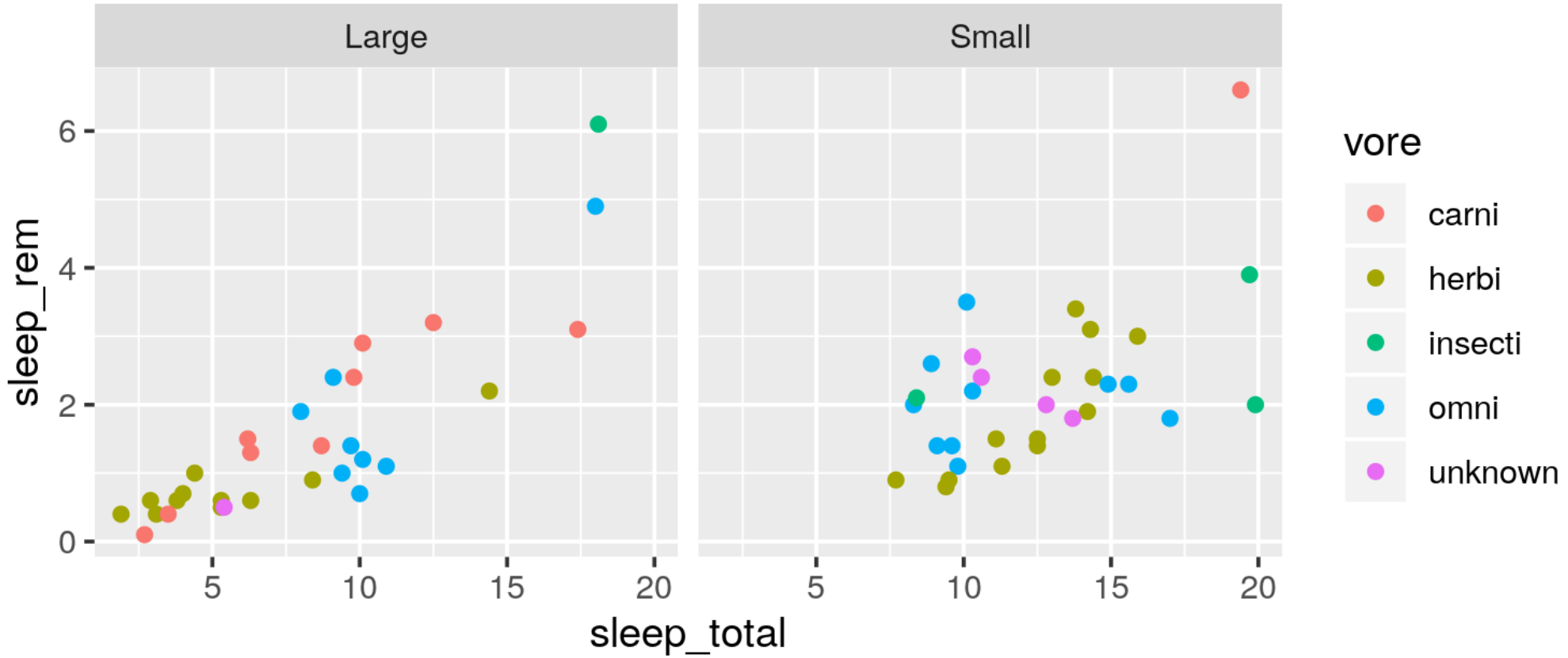
```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem,
                         colour = vore, shape = vore, size = bodywt)) +
  geom_point()
```



Faceting: `facet_wrap()`

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point() +
  facet_wrap(~ body_size, nrow = 1)
```

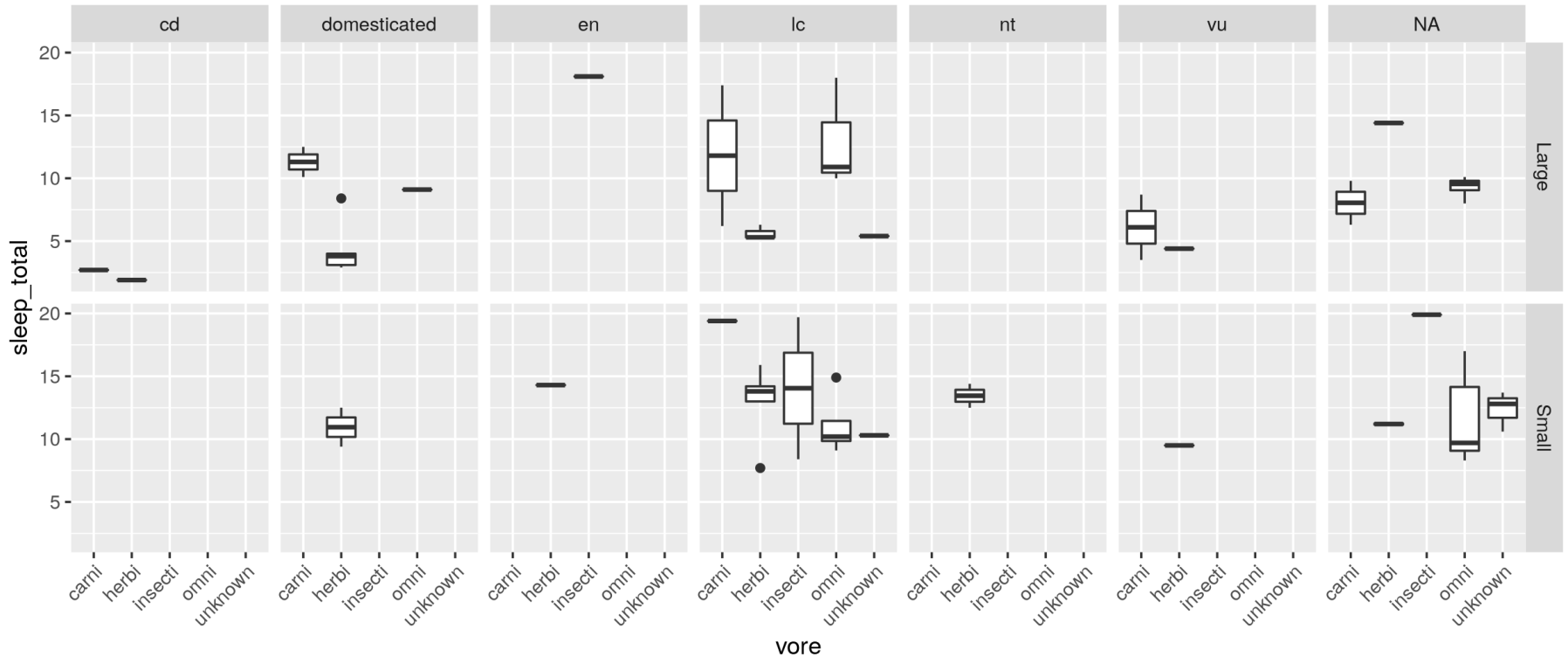
Split plots by **one** grouping variable



Faceting: `facet_grid()`

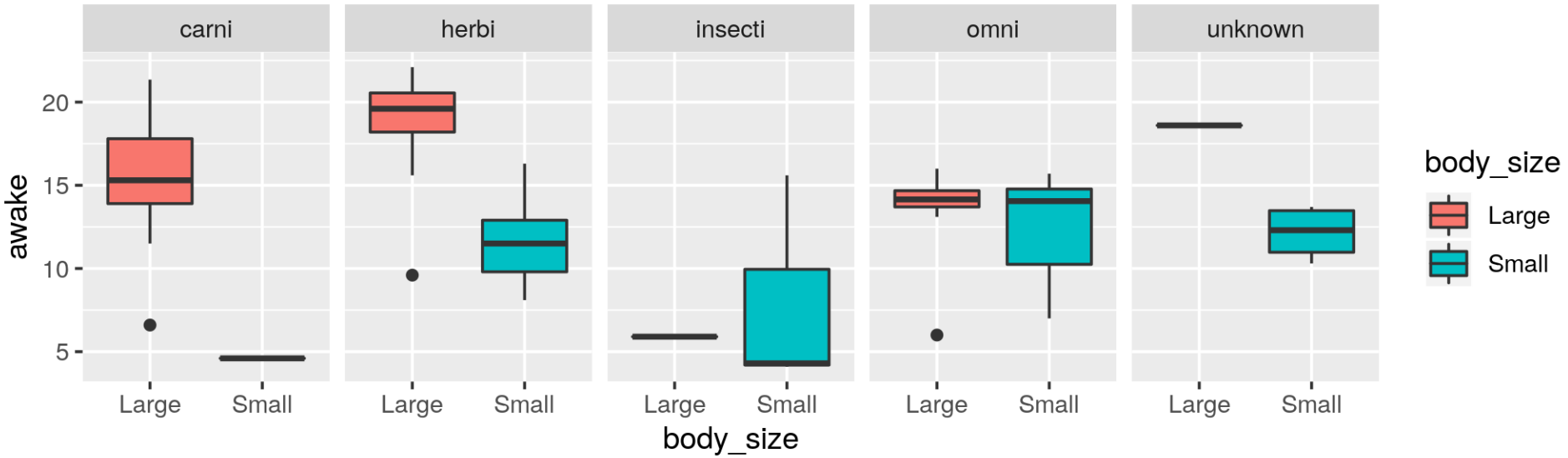
```
ggplot(data = sleep, aes(x = vore, y = sleep_total)) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  geom_boxplot() +
  facet_grid(body_size ~ conservation)
```

Split plots by **two** grouping variables



Your Turn: Create this plot

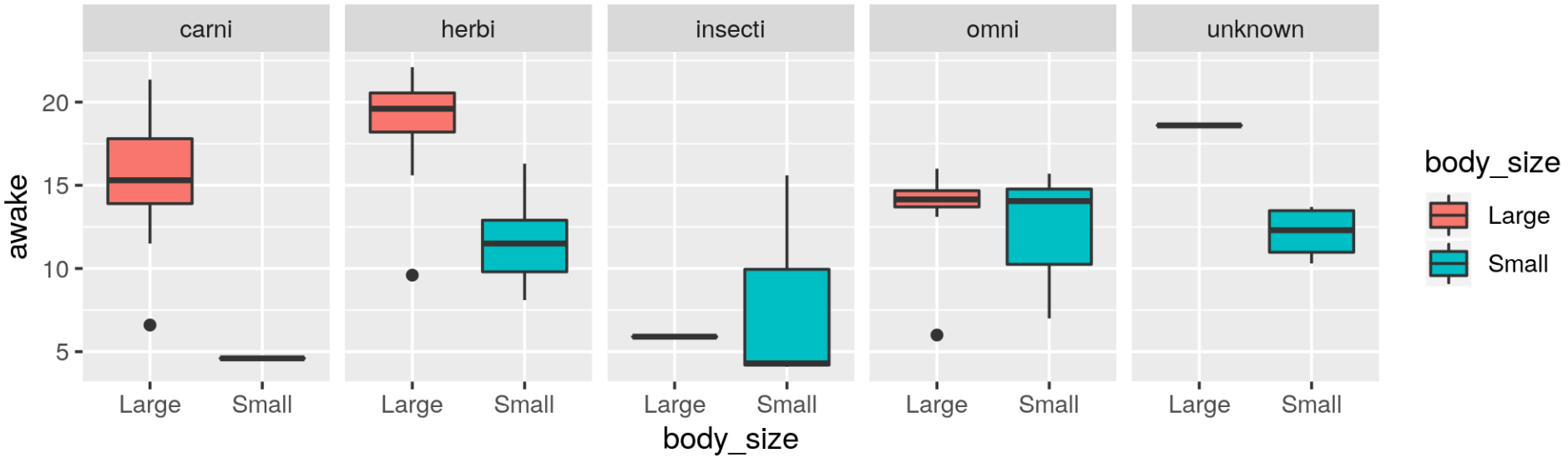
```
ggplot(data = ???, aes(???)) +
  ??? +
  ???
```



Hint: `colour` is for outlining with a colour, `fill` is for 'filling' with a colour

Your Turn: Create this plot

```
ggplot(data = sleep, aes(x = body_size, y = awake, fill = body_size)) +
  geom_boxplot() +
  facet_wrap(~ vore, nrow = 1)
```



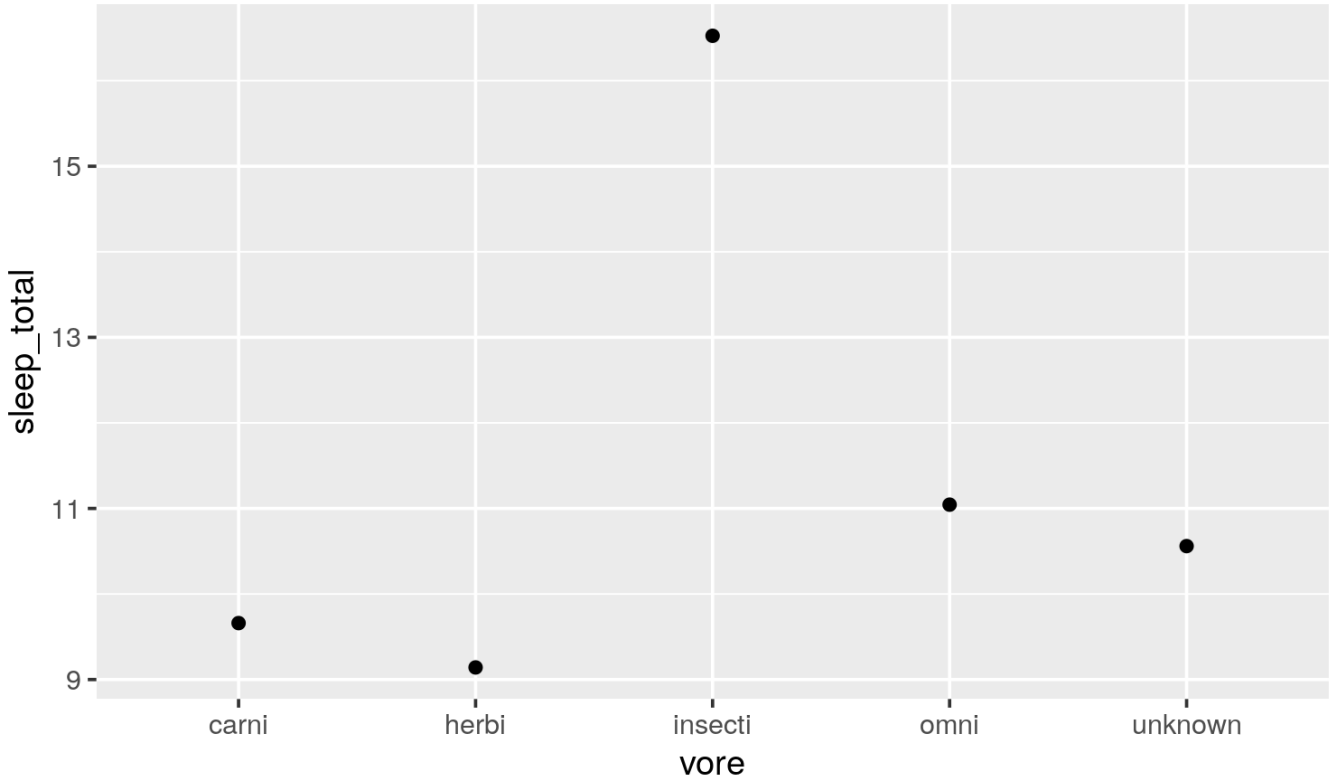
Hint: `colour` is for outlining with a colour, `fill` is for 'filling' with a colour

Adding Statistics to Plots

Using stats: Summarizing data

Add data means as points

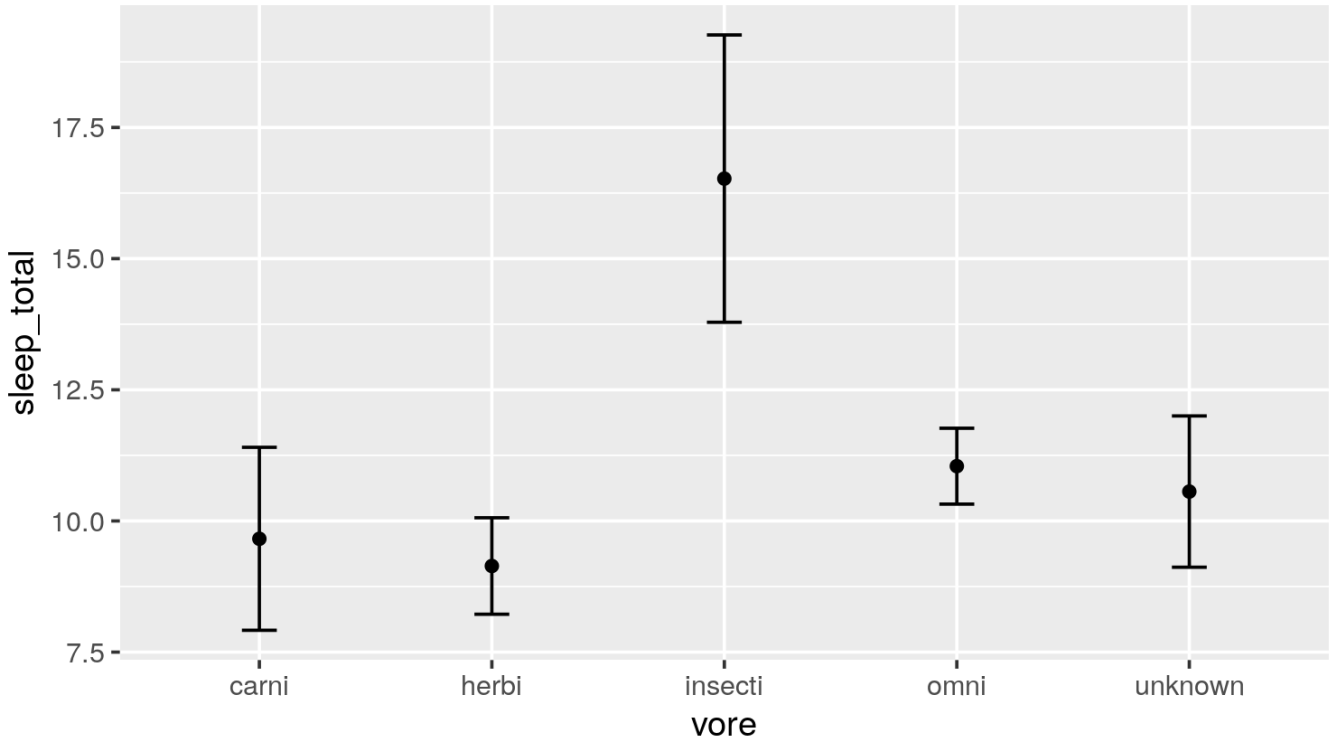
```
ggplot(data = sleep, aes(x = vore, y = sleep_total)) +
  stat_summary(geom = "point", fun.y = mean)
```



Using stats: Summarizing data

Add error bars, calculated from the data

```
ggplot(data = sleep, aes(x = vore, y = sleep_total)) +
  stat_summary(geom = "point", fun.y = mean) +
  stat_summary(geom = "errorbar", width = 0.15, fun.data = mean_se)
```

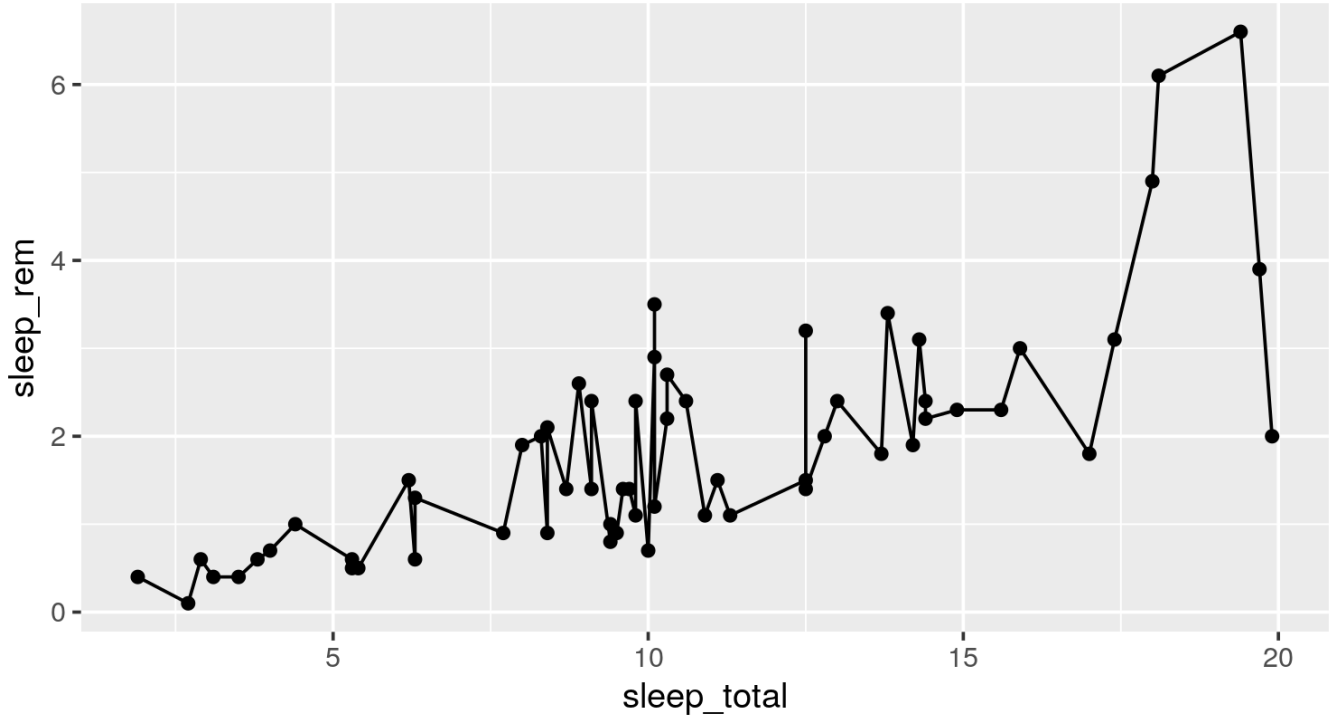


Using stats: Trendlines

`geom_line()` is connect-the-dots, not a trend or linear model

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
  geom_point() +
  geom_line()
```

Not what we're looking for

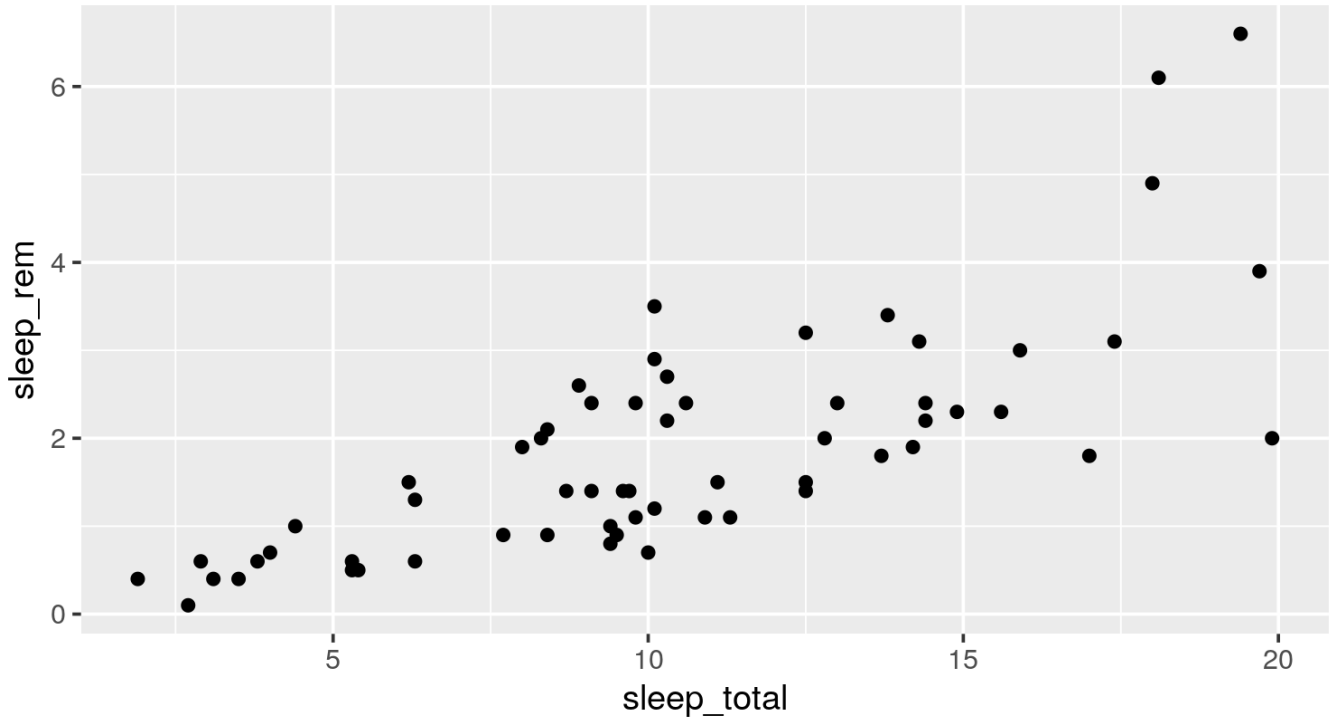


Using stats: Trendlines

Let's add a trend line properly

Start with basic plot:

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem)) +
  geom_point()
```

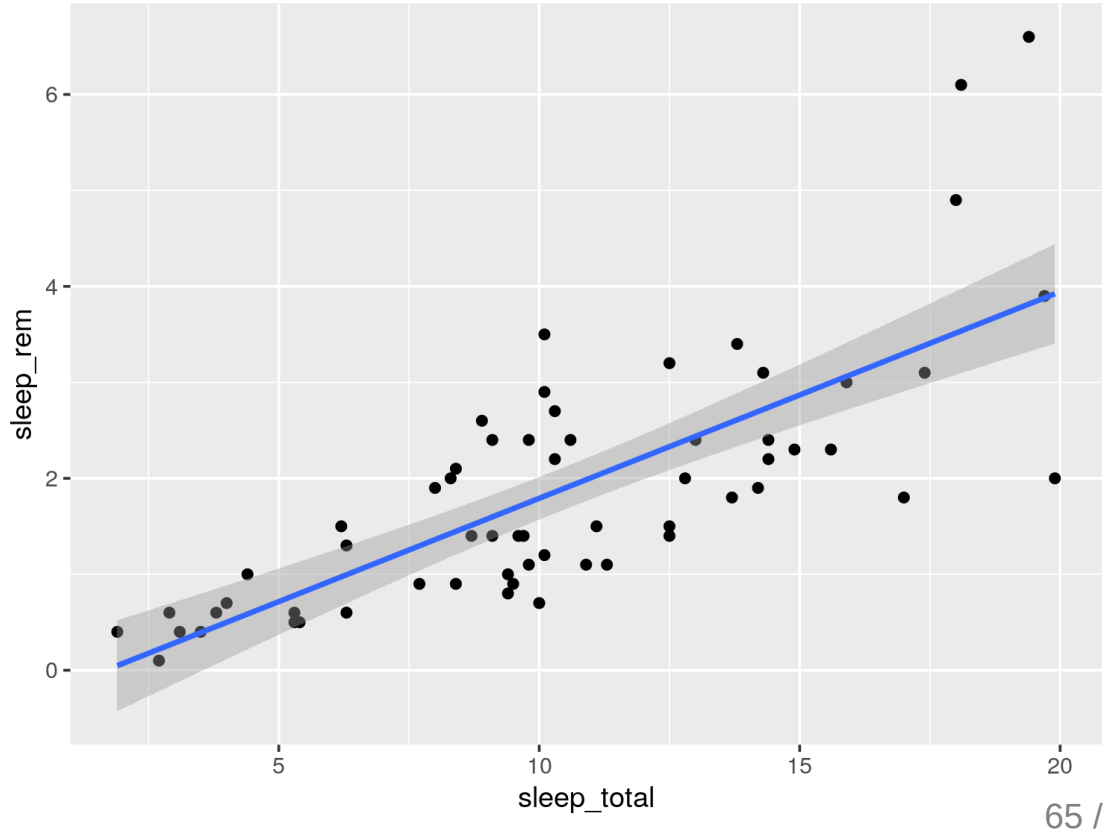


Using stats: Trendlines

Add the `stat_smooth()`

- `lm` is for "linear model" (i.e. trendline)
- the grey area represents the standard error

```
ggplot(data = sleep,
       aes(x = sleep_total, y = sleep_rem)) +
  geom_point() +
  stat_smooth(method = "lm")
```

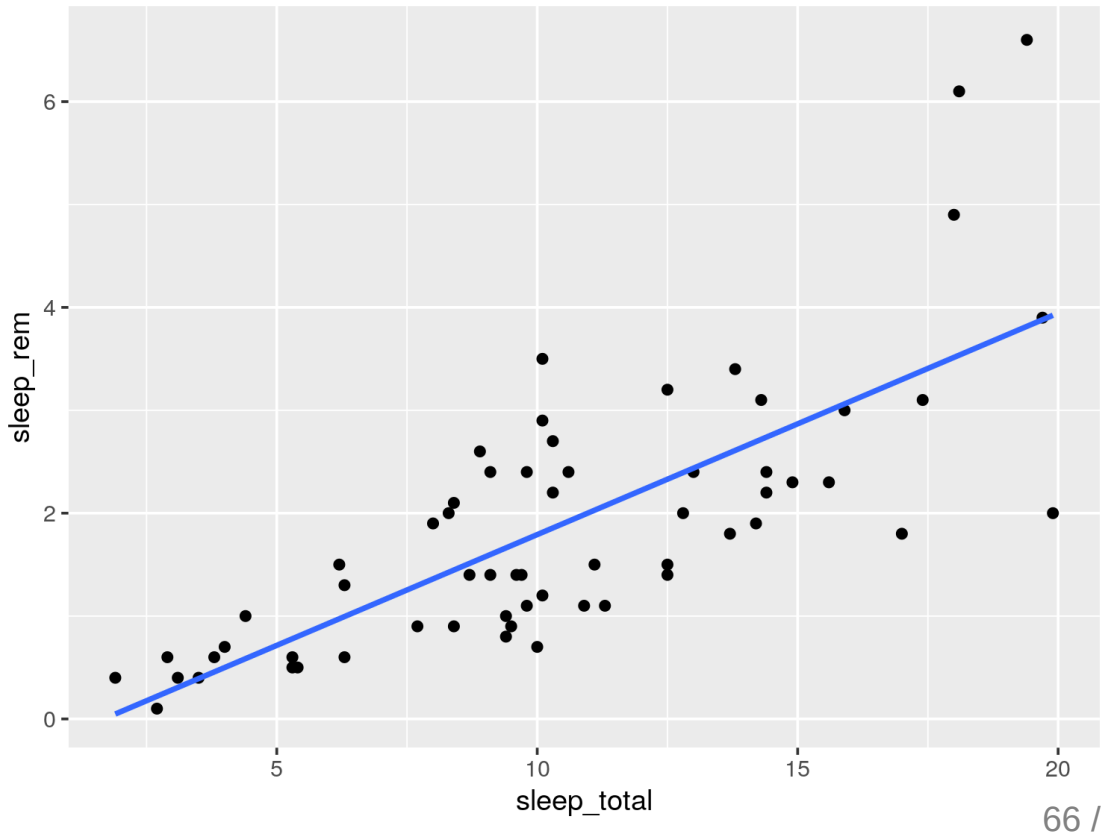


Using stats: Trendlines

Add the `stat_smooth()`

- `lm` is for "linear model" (i.e. trendline)
- the grey area represents the standard error
- remove the grey area with `se = FALSE`

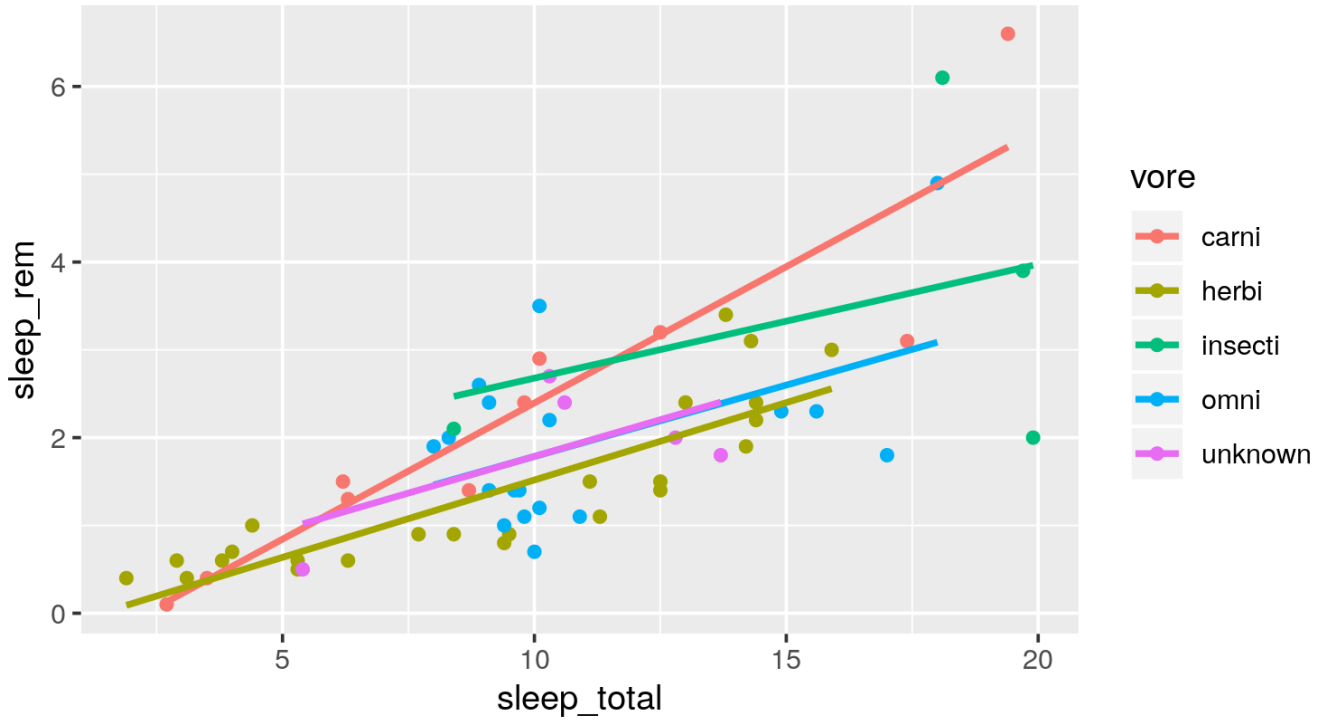
```
ggplot(data = sleep,
       aes(x = sleep_total, y = sleep_rem)) +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE)
```



Using stats: Trendlines

A line for each group

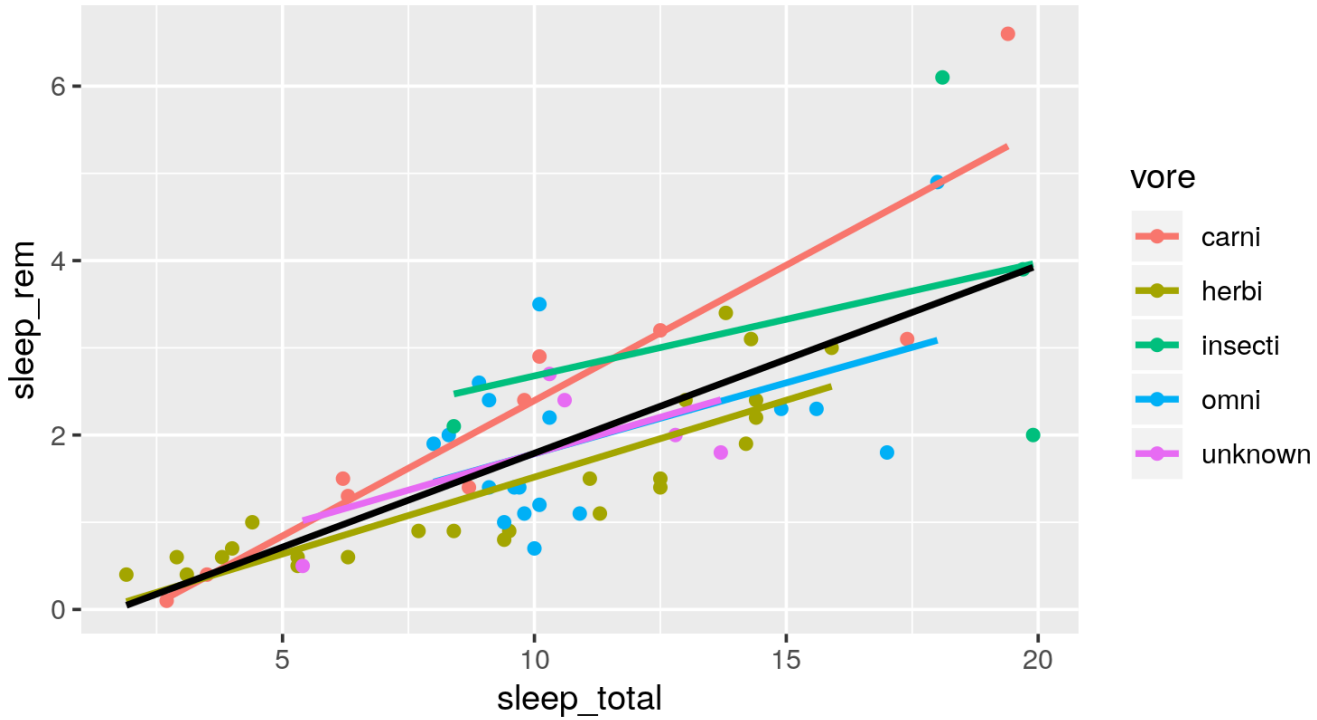
```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE)
```



Using stats: Trendlines

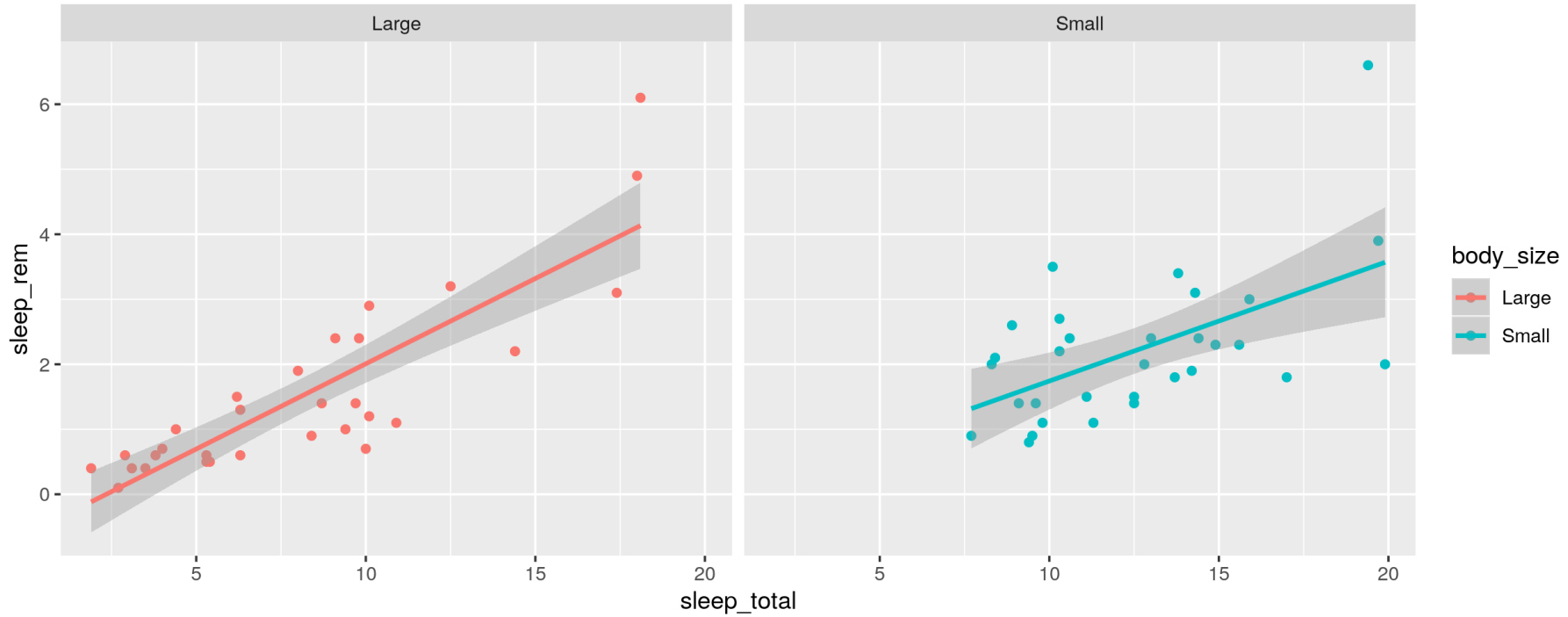
A line for each group AND overall

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE) +
  stat_smooth(method = "lm", se = FALSE, colour = "black")
```



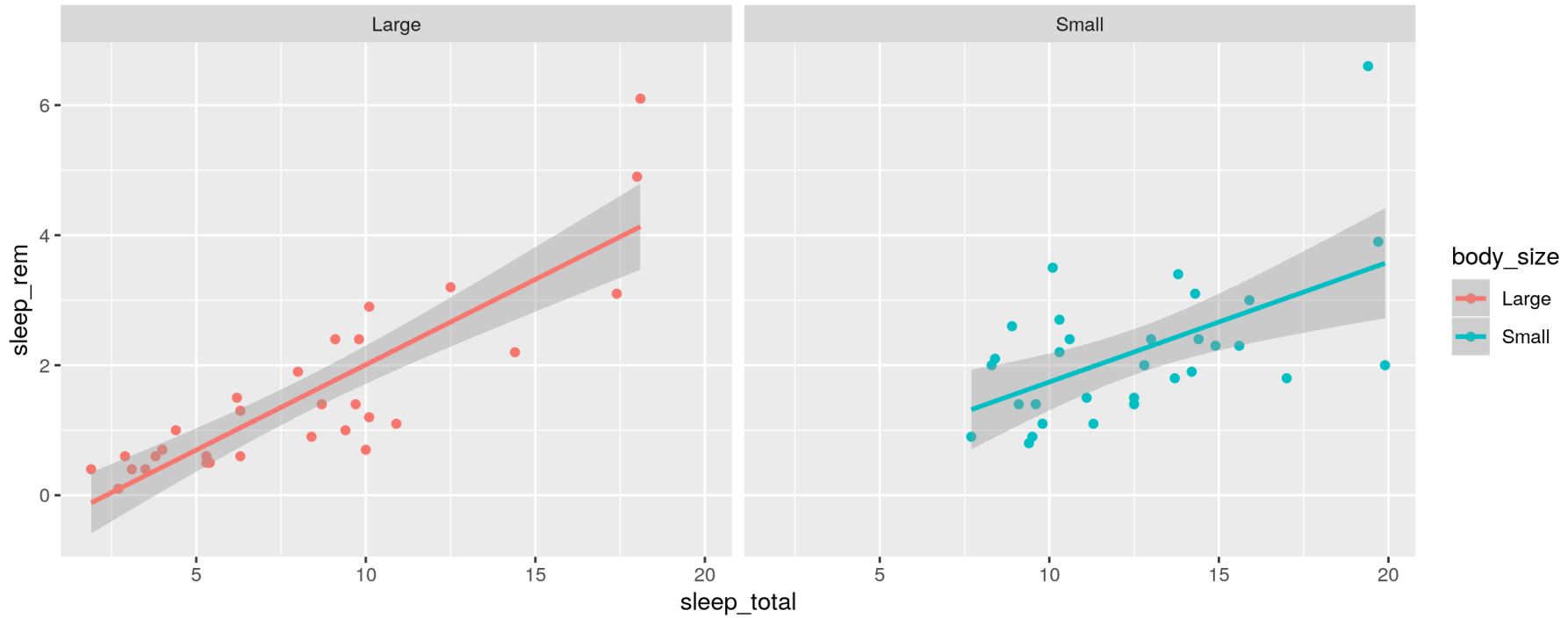
Your Turn: Create this plot

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



Your Turn: Create this plot

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem, colour = body_size)) +
  facet_wrap(~ body_size) +
  geom_point() +
  stat_smooth(method = "lm")
```

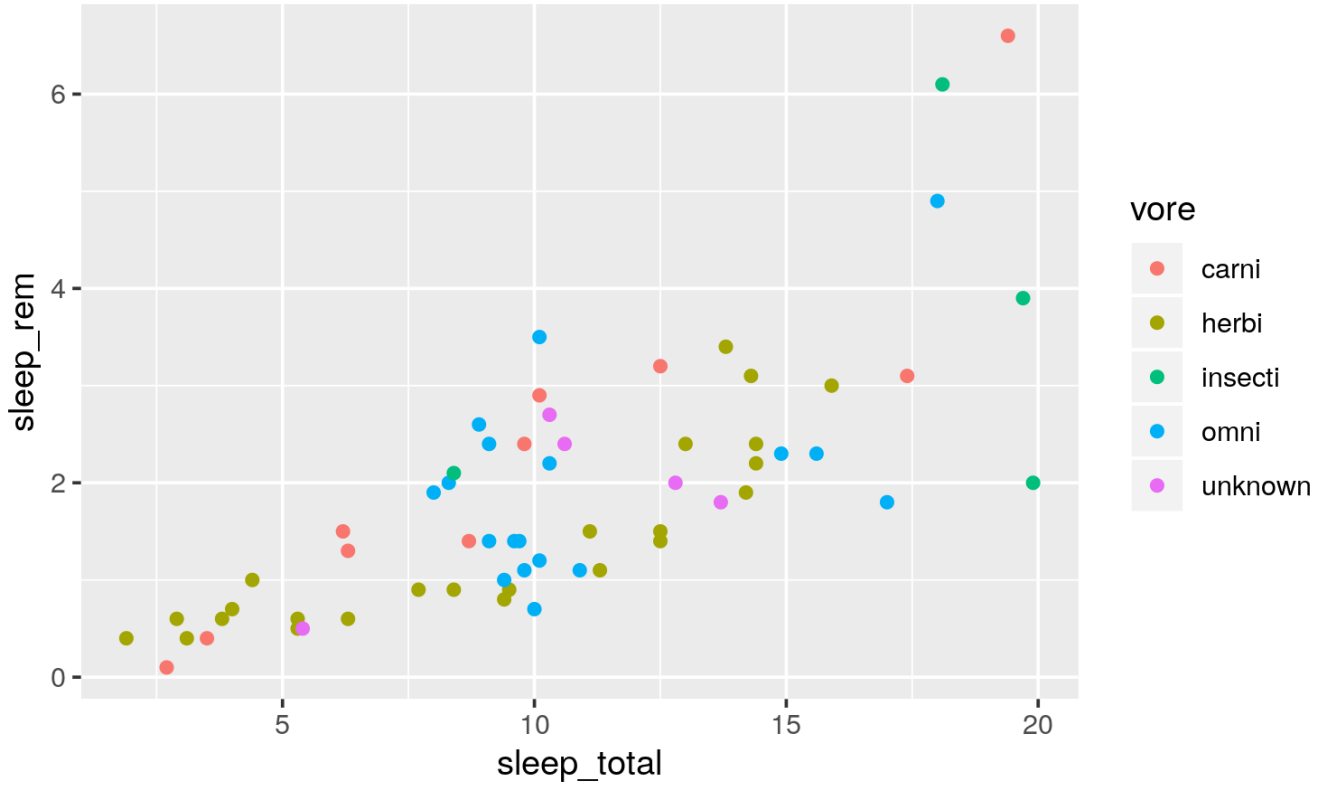


Customizing plots

Customizing: Starting plot

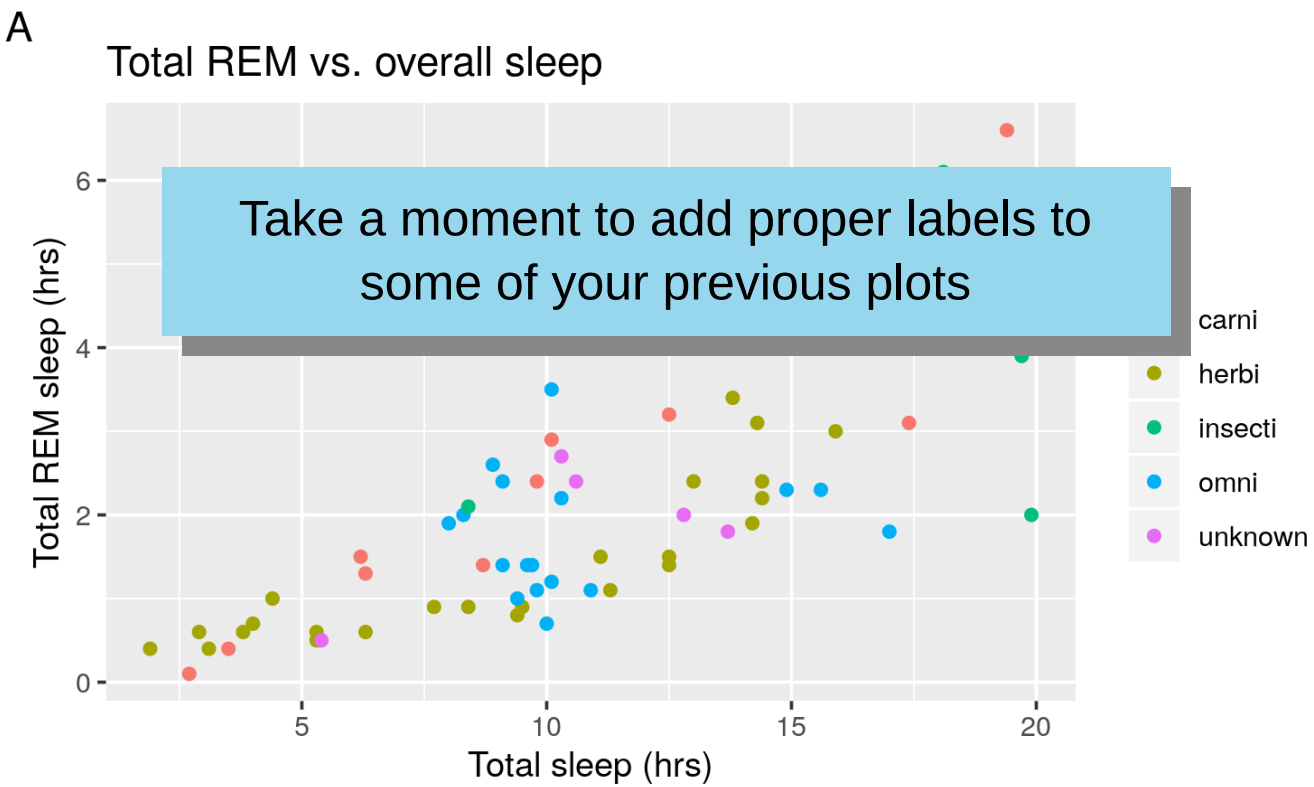
Let's work with this plot

```
g <- ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point()
```



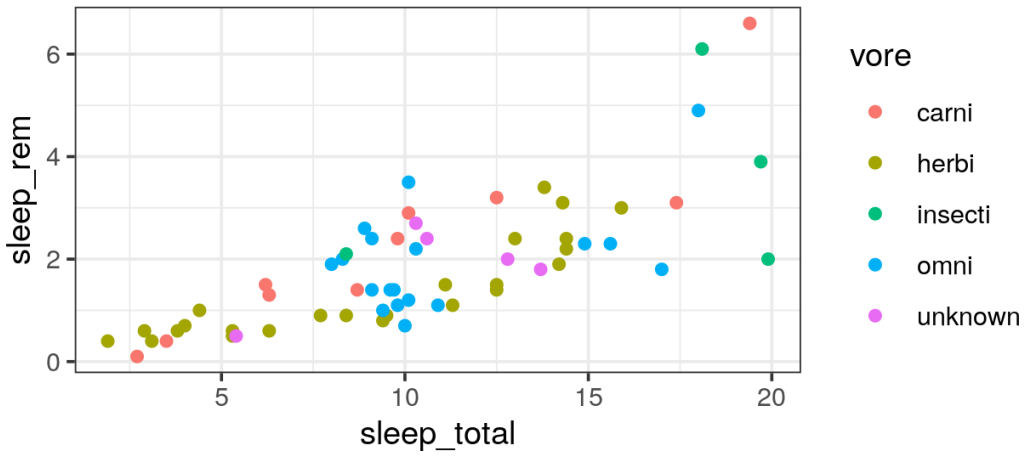
Customizing: Labels

```
g + labs(title = "Total REM vs. overall sleep",
         x = "Total sleep (hrs)",
         y = "Total REM sleep (hrs)",
         colour = "Diet", tag = "A")
```

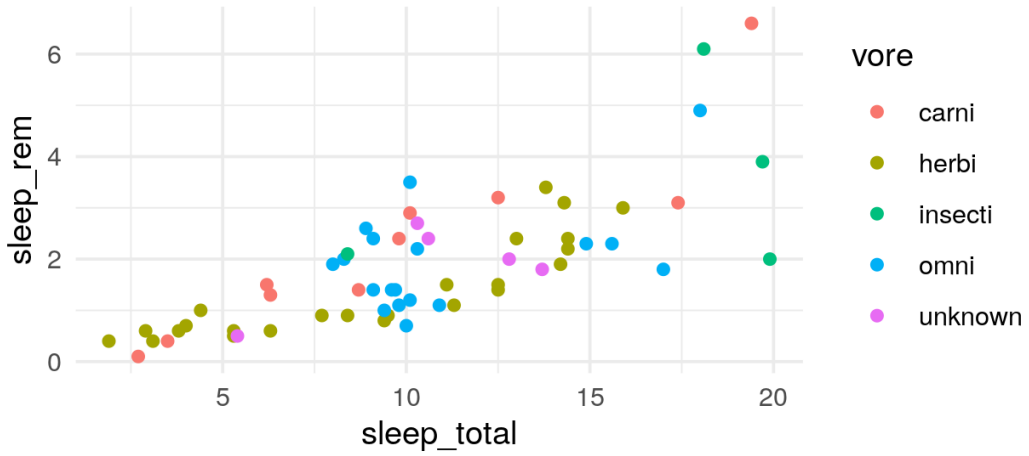


Customizing: Built-in themes

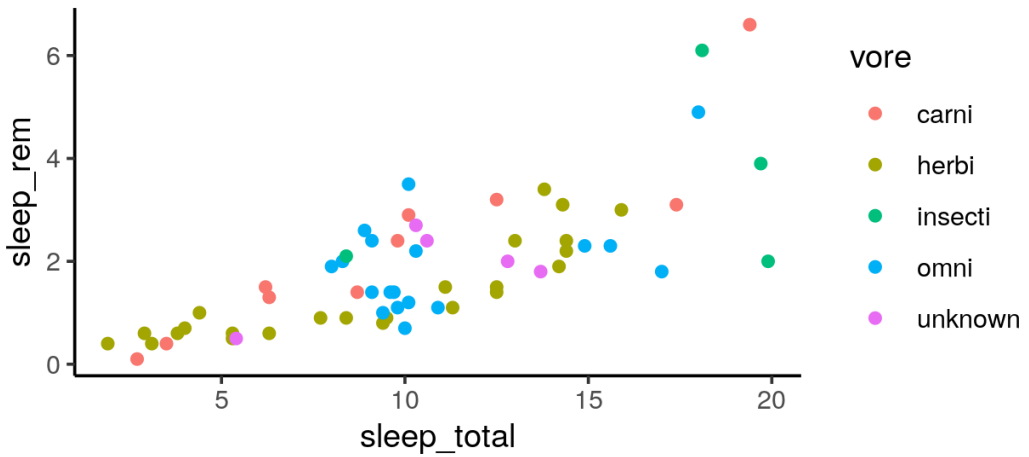
g + theme_bw()



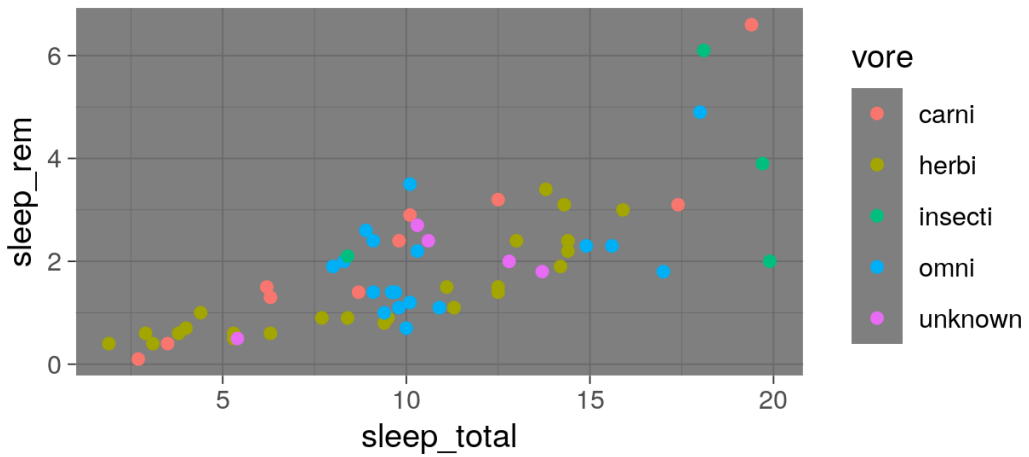
g + theme_minimal()



g + theme_classic()



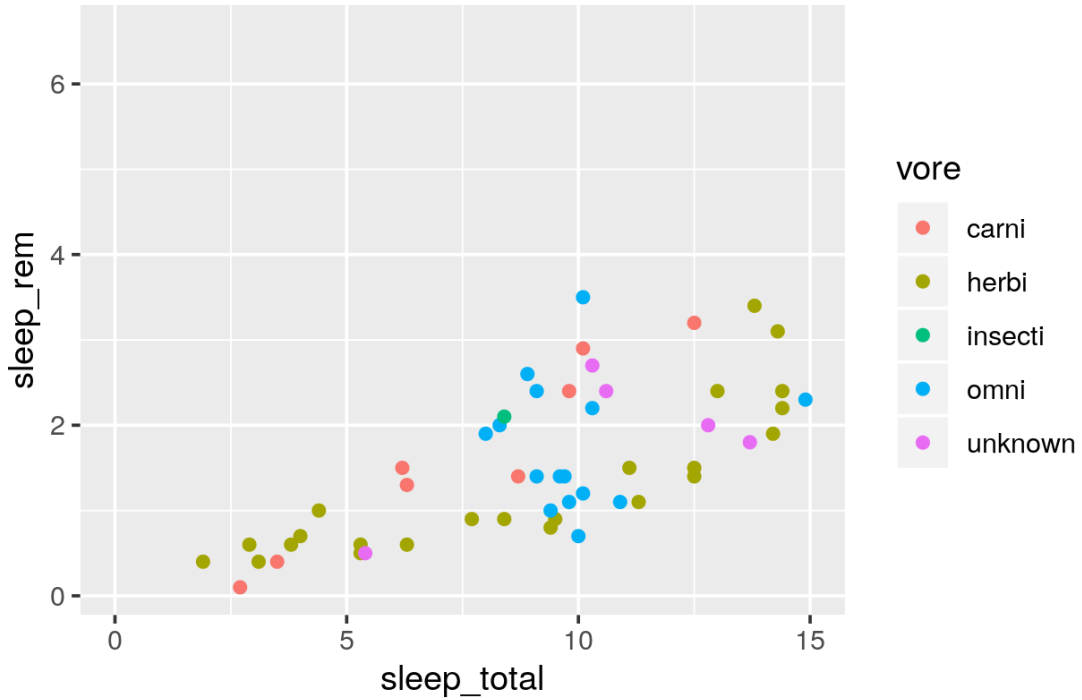
g + theme_dark()



Customizing: Data range

Limit the data (exclude data)

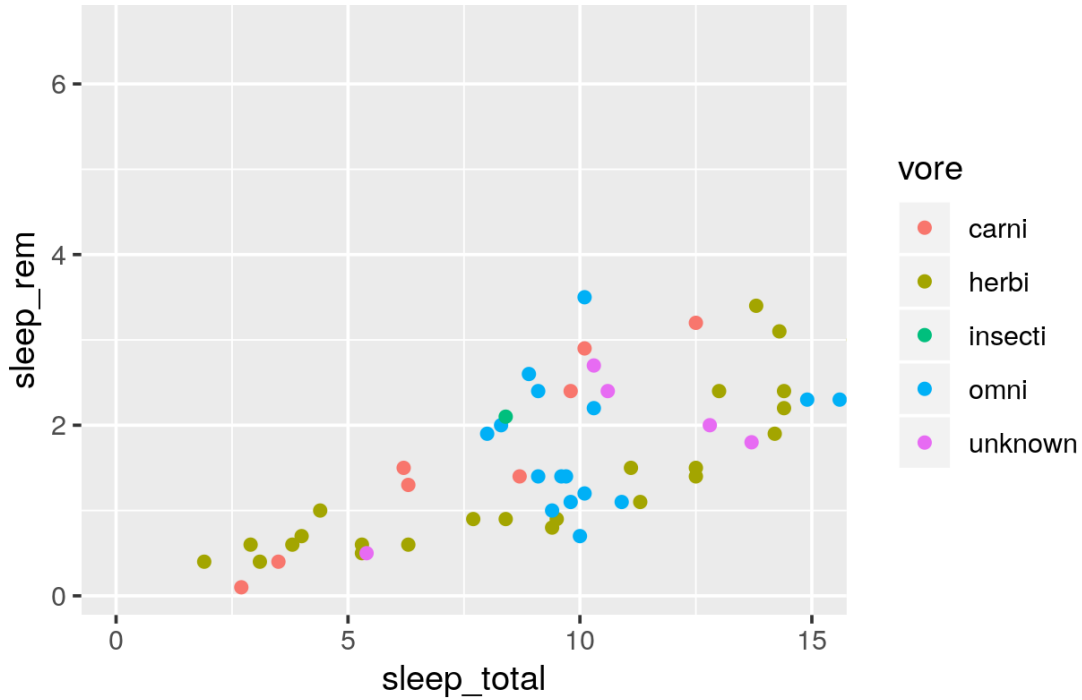
```
g + xlim(c(0, 15))
```




```
## Warning: Removed 9 rows containing missing
values (geom_point).
```

Limit the view (zoom)

```
g + coord_cartesian(xlim = c(0, 15))
```



Customizing: Axes

scale_ + (**x** or **y**) + type (**contiuous**, **discrete**, **date**, **datetime**)

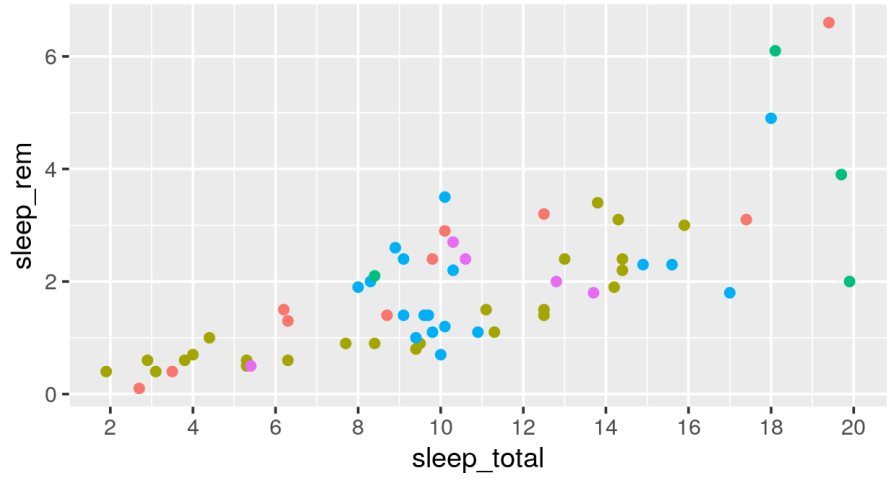
- **scale_x_continuous()**
- **scale_y_discrete()**
- etc.

Common arguments

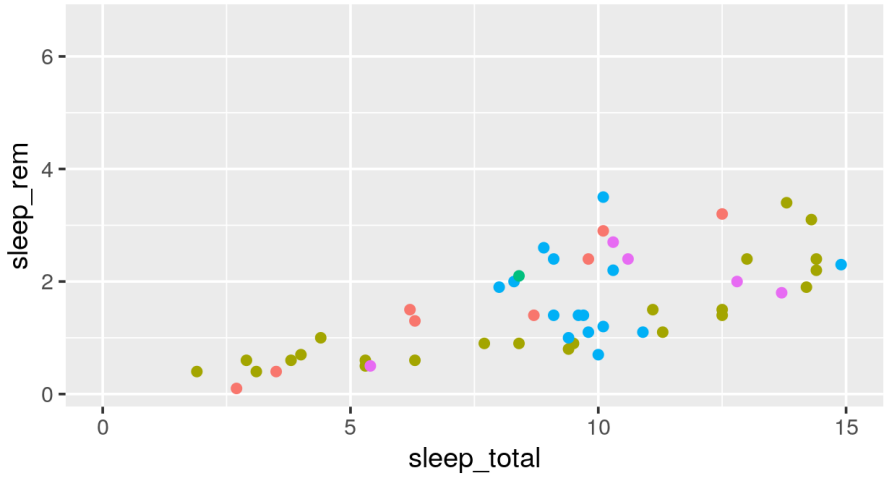
```
g + scale_x_continuous(breaks = seq(0, 20, 10)) # Tick breaks
g + scale_x_continuous(limits = c(0, 15))       # xlim() is a shortcut for this
g + scale_x_continuous(expand = c(0, 0))        # Space between axis and data
```

Customizing: Axes

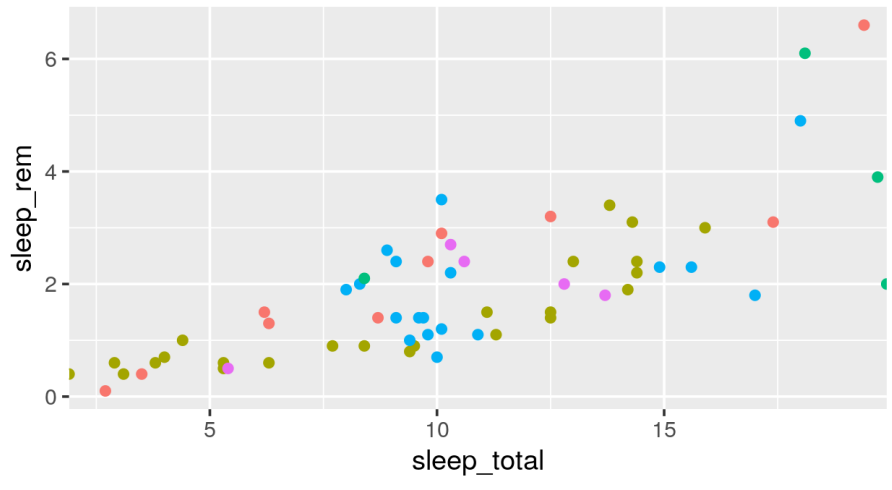
g + scale_x_continuous(breaks = seq(0, 20, 10))



g + scale_x_continuous(limits = c(0, 15))



g + scale_x_continuous(expand = c(0, 0))

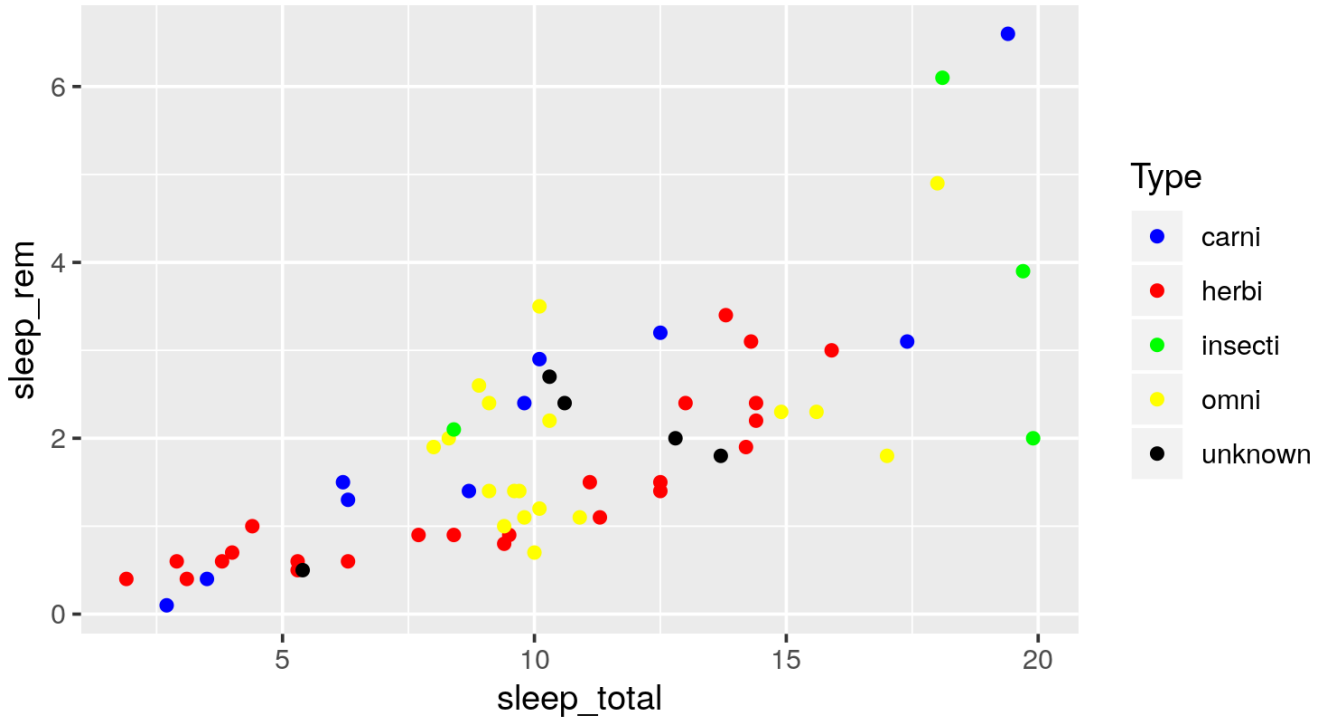


Customizing: Aesthetics

Using scales

`scale_` + aesthetic (**`colour`**, **`fill`**, **`size`**, etc.) + type (**`manual`**, **`continuous`**, **`datetime`**, etc.)

```
g + scale_colour_manual(name = "Type",
                        values = c("blue", "red", "green", "yellow", "black"))
```

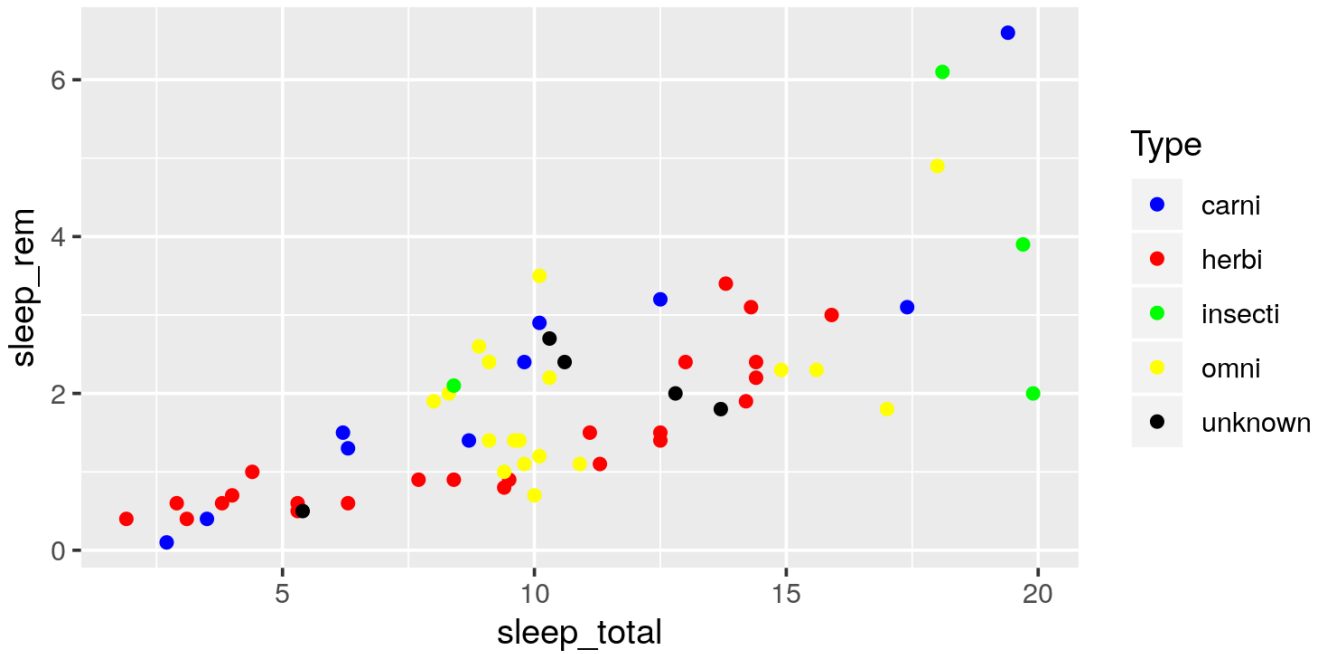


Customizing: Aesthetics

Using scales

Or be very explicit:

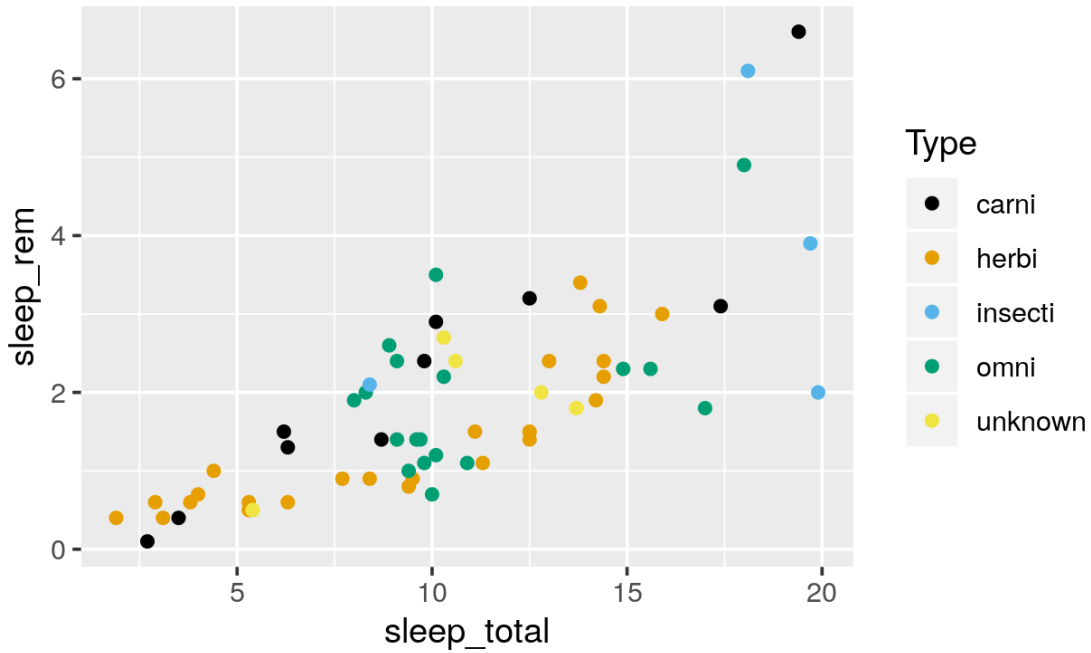
```
g + scale_colour_manual(name = "Type",
                        values = c("carni" = "blue", "herbi" = "red", "insecti" = "green",
                                   "omni" = "yellow", "unknown" = "black"))
```

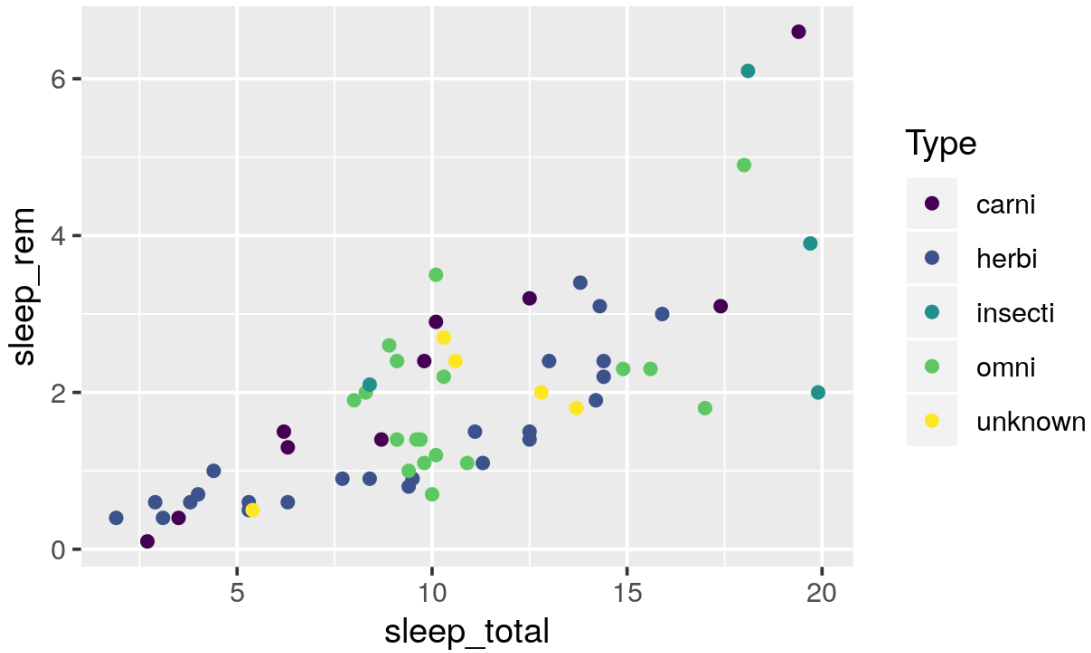


Customizing: Aesthetics

For colours, consider colour-blind-friendly scales

```
library(ggthemes)
g + scale_colour_colorblind(name = "Type")
g + scale_colour_viridis_d(name = "Type")
```



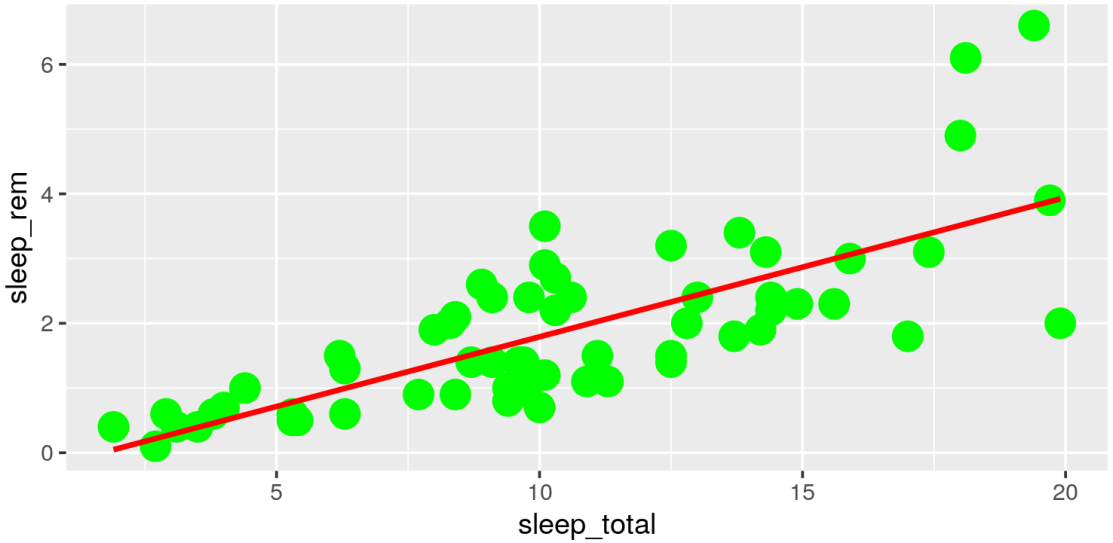


Customizing: Aesthetics

Forcing

Remove the association between a variable and an aesthetic

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point(colour = "green", size = 5) +
  stat_smooth(method = "lm", se = FALSE, colour = "red")
```

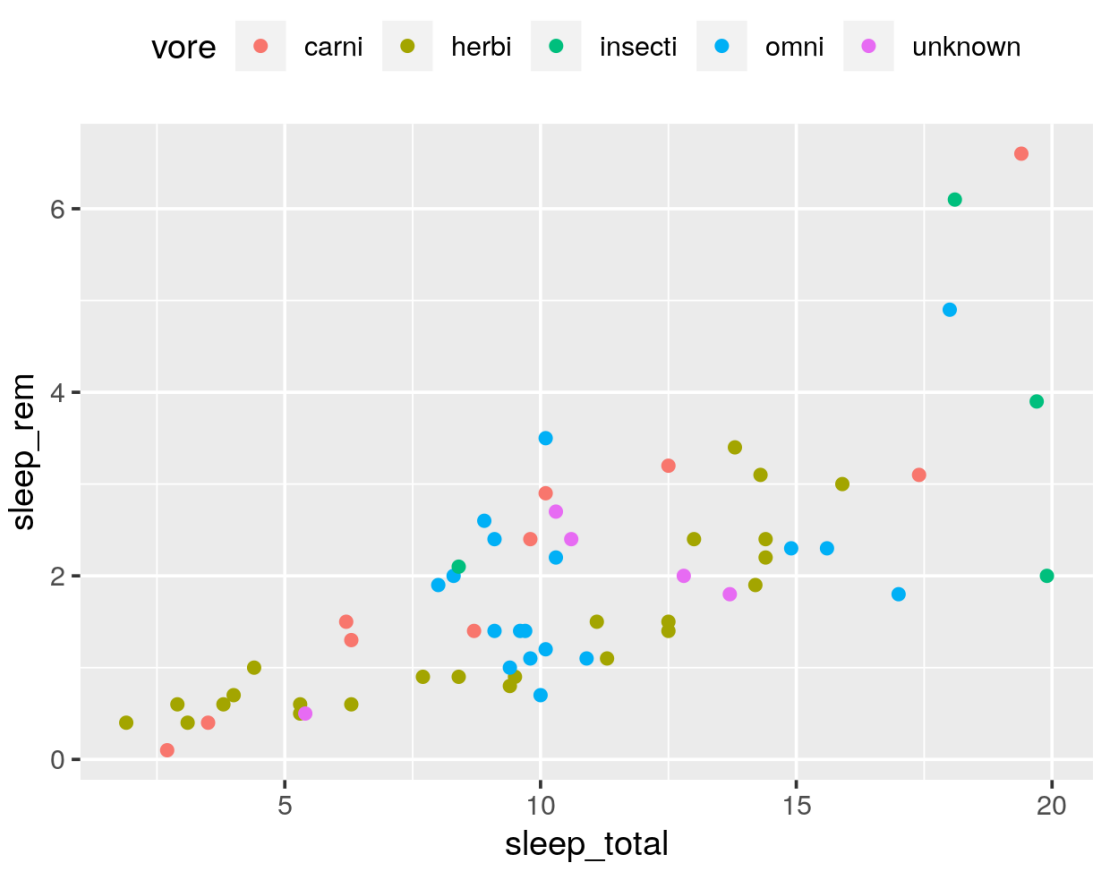


Note: When forcing, aesthetic is not inside **aes()**

Customizing: Legends placement

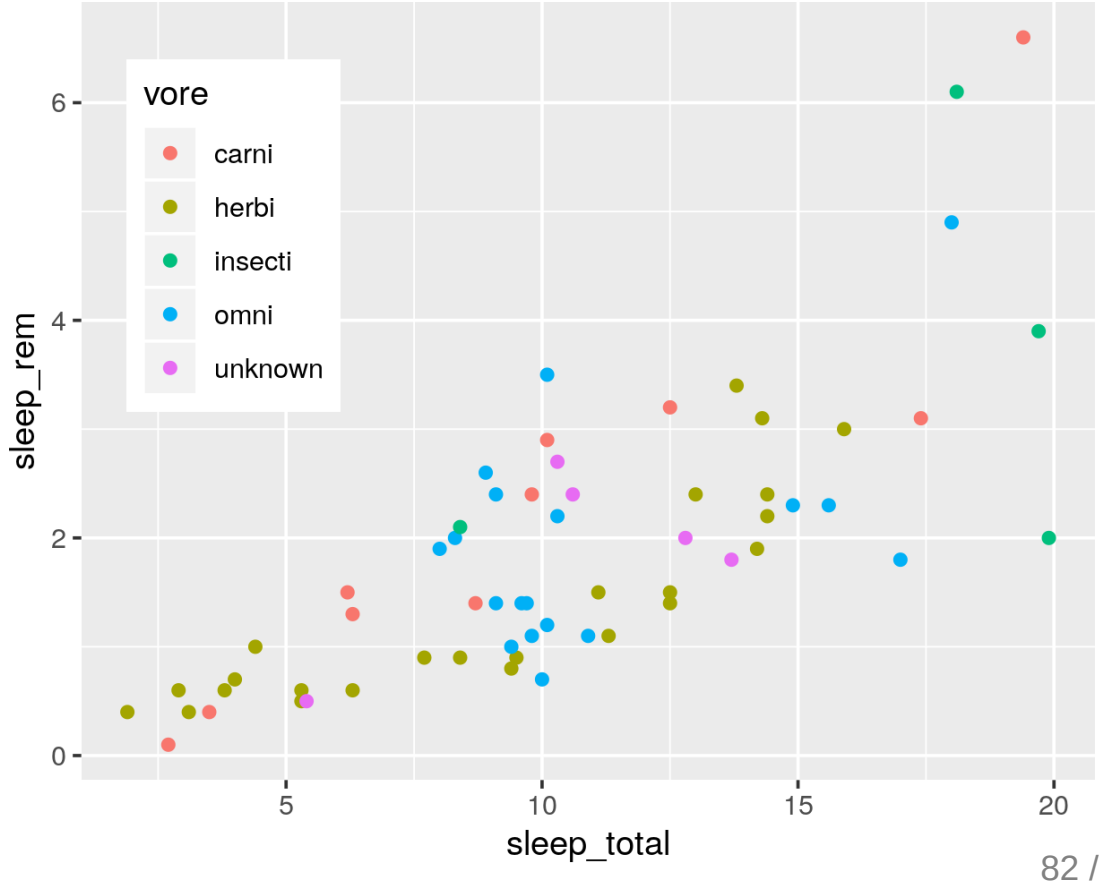
At the: top, bottom, left, right

```
g + theme(legend.position = "top")
```

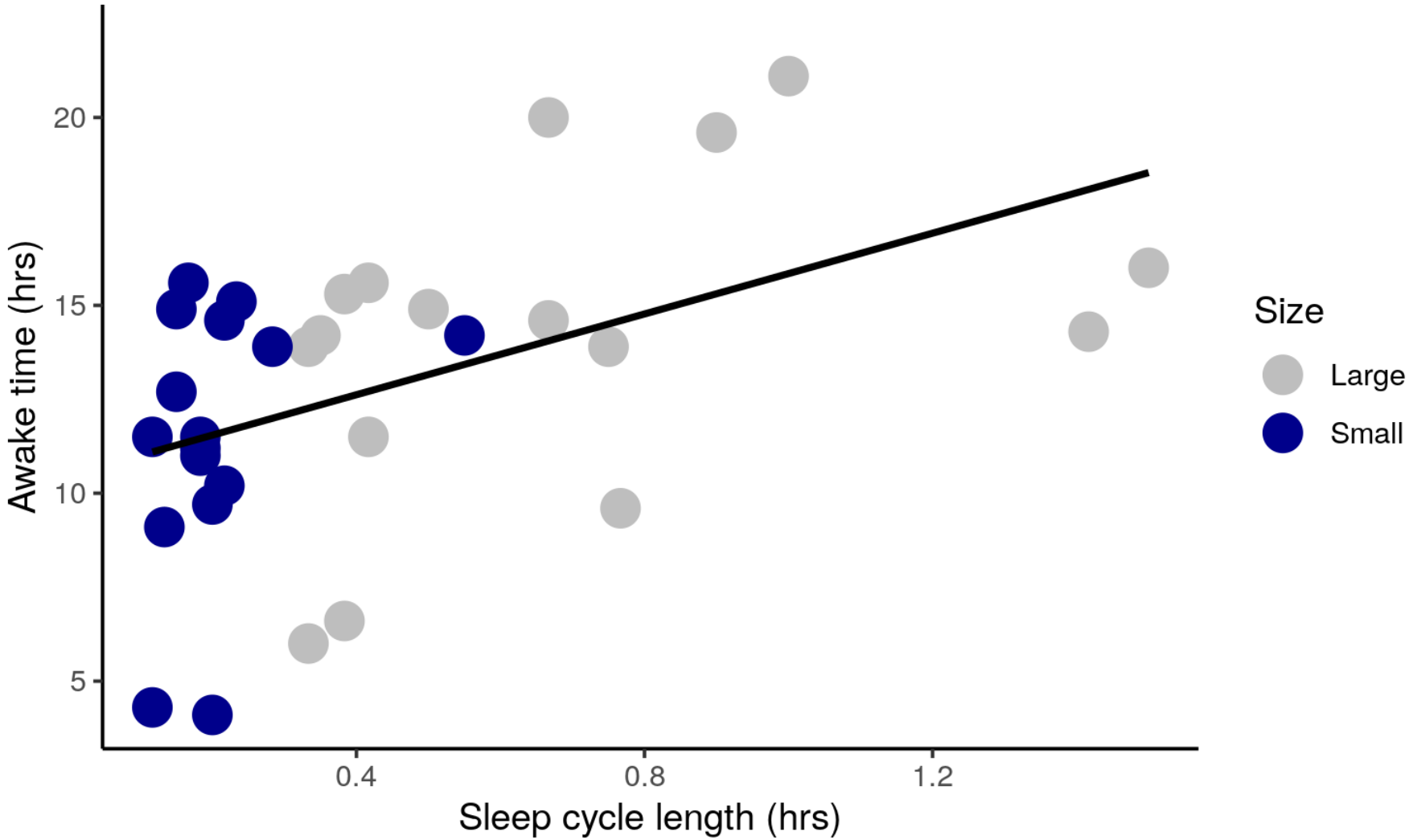


Exactly here

```
g + theme(legend.position = c(0.15, 0.7))
```

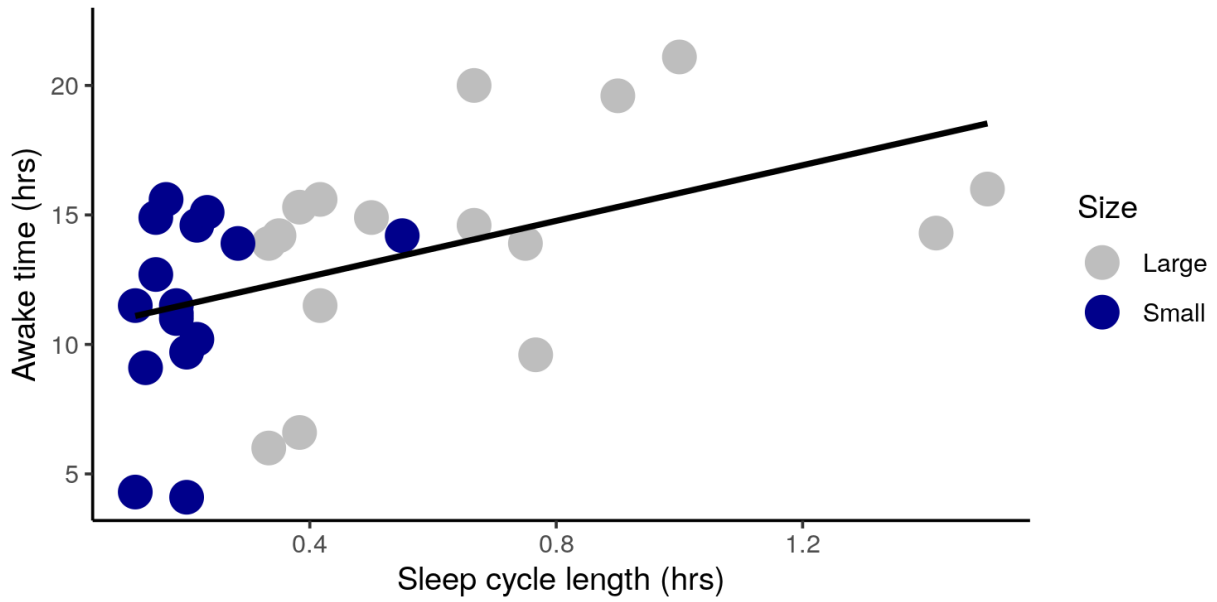


Your Turn: Create this plot



Your Turn: Create this plot

```
ggplot(sleep, aes(x = sleep_cycle, y = awake, colour = body_size)) +
  theme_classic() +
  geom_point(size = 5) +
  stat_smooth(method = "lm", se = FALSE, colour = "black") +
  scale_colour_manual(name = "Size", values = c("grey", "darkblue")) +
  labs(x = "Sleep cycle length (hrs)",
       y = "Awake time (hrs)")
```

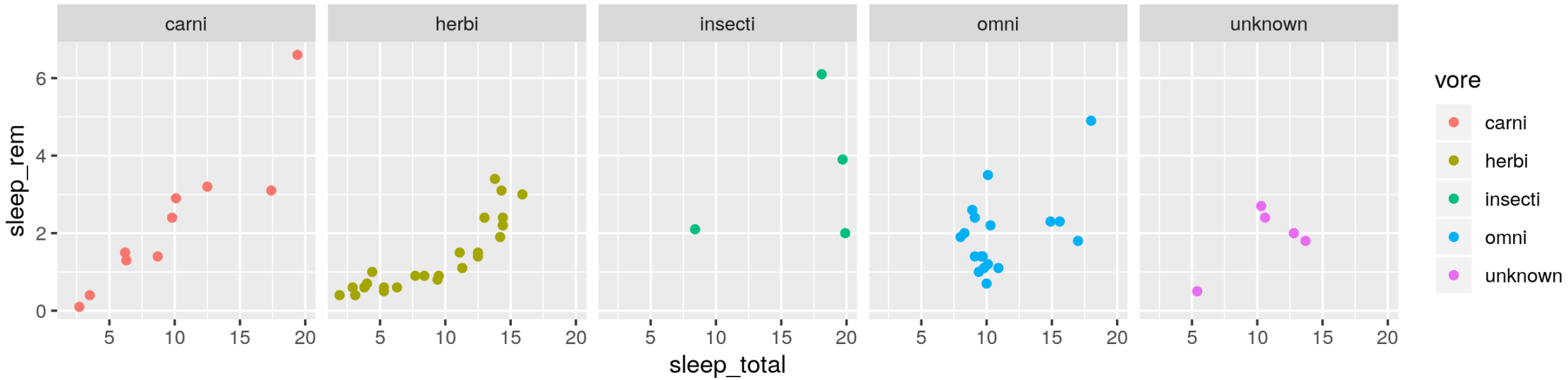


Annotating plots

Annotating

Plot to be annotated: Let's add sample sizes

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point() +
  facet_grid(~ vore)
```



Annotating

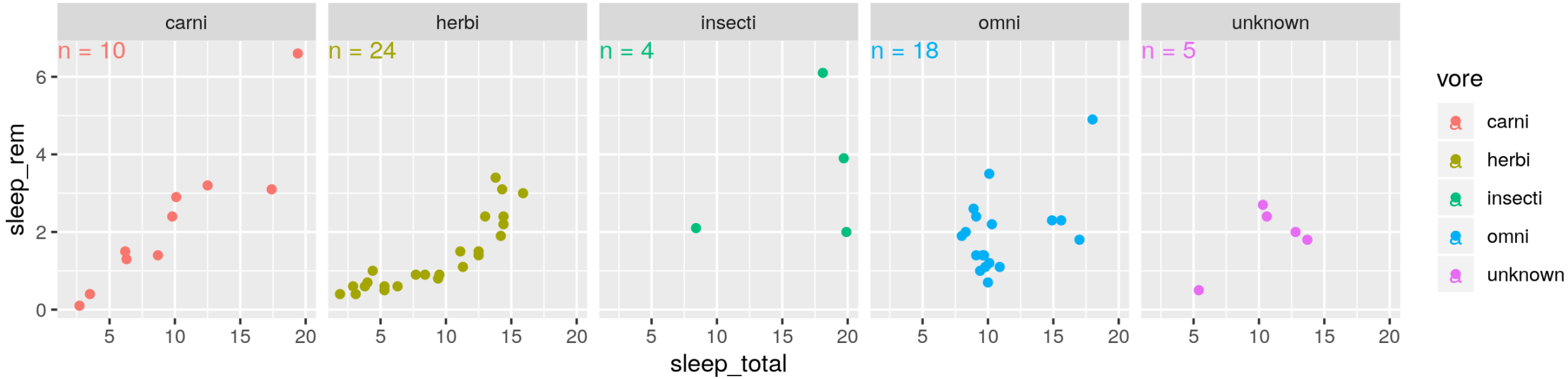
Create data to use in our annotations

```
n <- data.frame(vore = c("carni", "herbi", "insecti", "omni", "unknown"),
                text = c("n = 10", "n = 24", "n = 4", "n = 18", "n = 5"))
n
```

```
##      vore   text
## 1   carni n = 10
## 2   herbi n = 24
## 3 insecti  n = 4
## 4    omni n = 18
## 5 unknown  n = 5
```

Annotating

```
ggplot(data = sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point() +
  facet_grid(~ vore) +
  geom_text(data = n,                    # Use 'n' data (not the `sleep` data)
            x = -Inf, y = +Inf,          # Hard coded location (left, top)
            aes(label = text),           # Map 'text' to label
            hjust = 0, vjust = 1)        # Adjust horizontal and vertical placement
```



ggplot will automatically apply aesthetics in the **`ggplot()`** line to all data included

Saving plots

Saving plots

RStudio Export

Demo

`ggsave()`

```
g <- ggplot(sleep, aes(x = vore, y = sleep_total)) +
  geom_boxplot()

ggsave(filename = "sleep_boxplot.png", plot = g)
```

```
## Saving 6 x 3.9 in image
```

Saving plots

Publication quality plots

- Many publications require 'lossless' (pdf, svg, eps, ps) or high quality formats (tiff, png)
- Specific sizes corresponding to columns widths
- Minimum resolutions

```
g <- ggplot(sleep, aes(x = vore, y = sleep_total)) +
  geom_boxplot() +
  labs(x = "Carnivore", y = "Total sleep (hrs)") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

ggsave(filename = "sleep_boxplot_sm.pdf", plot = g, dpi = 300,
       height = 80, width = 129, units = "mm")
```

Wrapping Up!

Wrapping up: Common mistakes

- The **package** is **`ggplot2`**, the function is just **`ggplot()`**
- Did you remember to put the **`+`** at the **end** of the line?
- Order matters! If you're using custom **`theme()`**'s, make sure you put these lines **after** bundled themes like **`theme_bw()`**, or they will be overwritten

Wrapping up: Common mistakes

I get an error regarding a missing aesthetic? But I know it's there!

You are probably trying to plot two different datasets, and you make references to variables in the `ggplot()` call that don't exist in one of the datasets:

Here, `conservation` only exists in the `sleep` dataset, not in the `n` dataset

```
n <- dplyr::count(sleep, vore)

ggplot(sleep, aes(x = sleep_total, y = sleep_rem, colour = conservation)) +
  geom_point() +
  facet_grid(~ vore) +
  geom_text(data = n, aes(label = n),
            x = -Inf, y = +Inf, hjust = 0, vjust = 1)
```

```
## Error in FUN(X[[i]], ...): object 'conservation' not found
```

Wrapping up: Common mistakes

I get an error regarding a missing aesthetic? But I know it's there!

Either move the aesthetic...

```
ggplot(sleep, aes(x = sleep_total, y = sleep_rem)) +
  geom_point(aes(colour = conservation)) +
  facet_grid(~ vore) +
  geom_text(data = n, aes(label = n),
            x = -Inf, y = +Inf, hjust = 0, vjust = 1)
```

Or assign it to NULL where it is missing...

```
ggplot(sleep, aes(x = sleep_total, y = sleep_rem, colour = conservation)) +
  geom_point() +
  facet_grid(~ vore) +
  geom_text(data = n, aes(label = n, colour = NULL),
            x = -Inf, y = +Inf, hjust = 0, vjust = 1)
```

Wrapping up

R is hard: But have no fear!

- Don't expect to remember everything!
- Copy/Paste is your friend (never apologize for using it!)
- Consider this workshop a resource to return to (Answers to Activities)

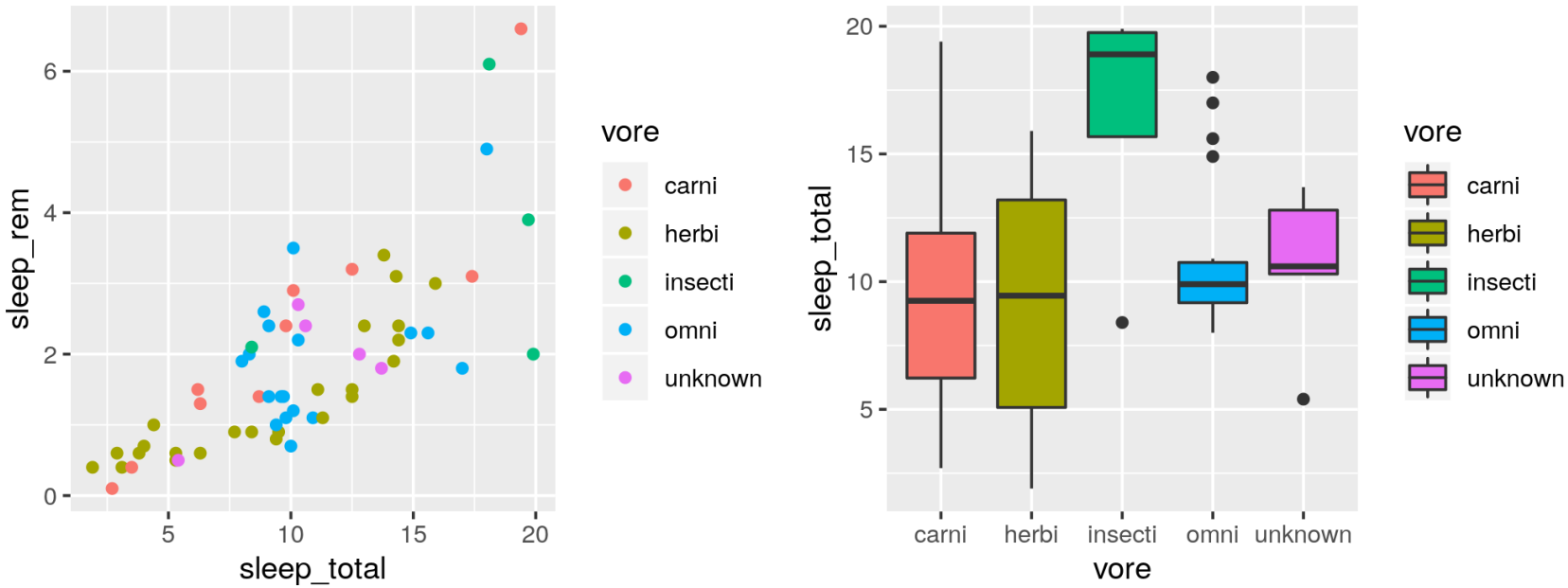
Further reading

- RStudio > Help > Cheatsheets > Data Visualization with ggplot2
- R for Data Science: Chapter 3
- Cookbook for R - by Winston Chang
  - See also R Graphics Cookbook by Winston Chang
- ggplot2 book v2 Hadley Wickham
  - You can compile it yourself from GitHub, or ask me for my compiled version (pdf)
- gridExtra Vignette: Arranging Multiple Grobs

EXTRA: Combining multiple plots

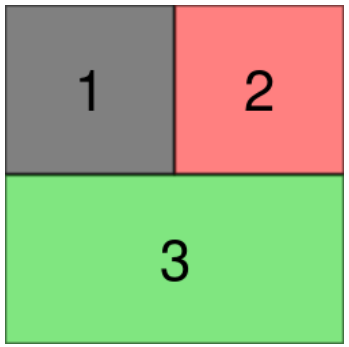
Combining with `gridExtra`

```
library(gridExtra)

g1 <- ggplot(sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point()

g2 <- ggplot(sleep, aes(x = vore, y = sleep_total, fill = vore)) +
  geom_boxplot()

grid.arrange(g1, g2, nrow = 1)
```



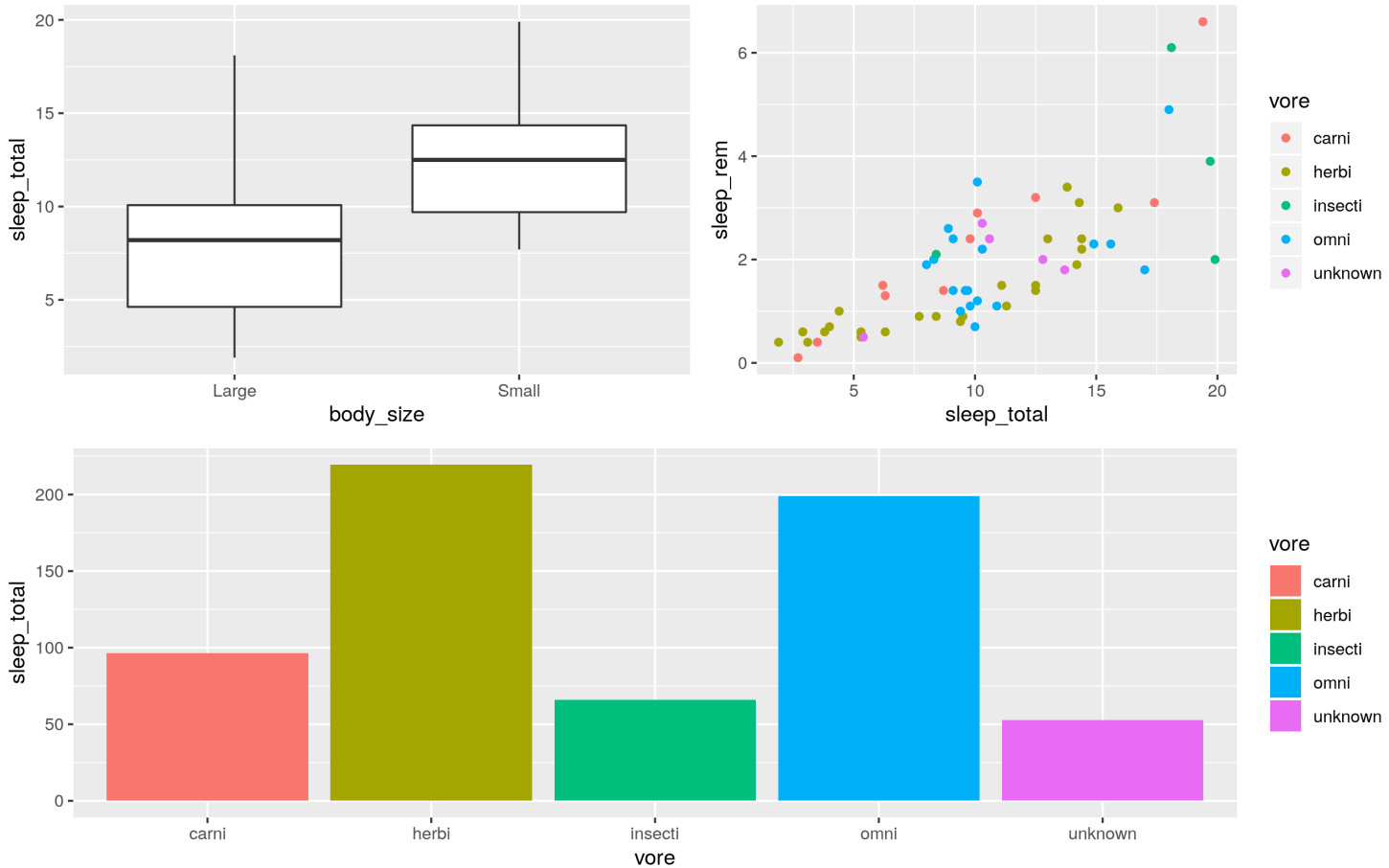
Combining multiple plots

```
library(gridExtra)
g1 <- ggplot(sleep, aes(x = body_size, y = sleep_total)) +
  geom_boxplot()

g2 <- ggplot(sleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +
  geom_point()

g3 <- ggplot(sleep, aes(x = vore, y = sleep_total, fill = vore)) +
  geom_bar(stat = "identity")

layout <- rbind(c(1,2),
                c(3,3))
```



Combining multiple plots

```
grid.arrange(g1, g2, g3, layout_matrix = layout)
```



Your turn

Combine any 3 figures

- feel free to try out the layout matrix option if you wish

```
library(gridExtra) # you'll probably have to install this first!

g1 <- ???
g2 <- ???
g3 <- ???

grid.arrange(???)
```