Datacamp_Data Visualization with ggplot2 (Part 2)___Best Practices

dizhen

2019/4/12

Bar plot

- Two types

Absolute values

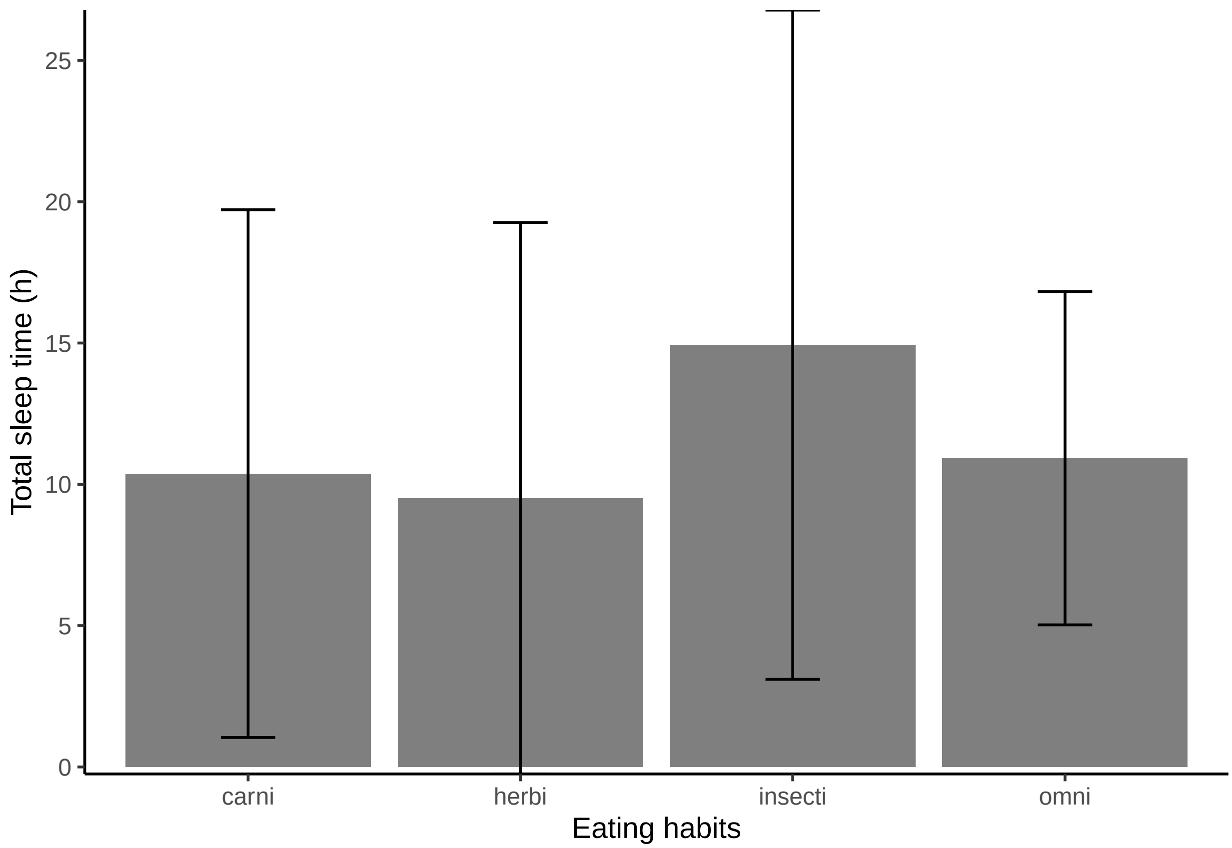
Distribution

```
library("ggplot2")

# Mammalian sleep
data(msleep)
sleep <- na.omit(msleep[,c("vore", "name", "sleep_total")])
str(sleep)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    76 obs. of  3 variables:
##  $ vore       : chr  "carni" "omni" "herbi" "omni" ...
##  $ name       : chr  "Cheetah" "Owl monkey" "Mountain beaver" "Greater short-tailed shrew" ...
##  $ sleep_total: num  12.1 17 14.4 14.9 4 14.4 8.7 10.1 3 5.3 ...
##  - attr(*, "na.action")= 'omit' Named int  8 55 57 58 63 69 73
##   ..- attr(*, "names")= chr  "8" "55" "57" "58" ...
```

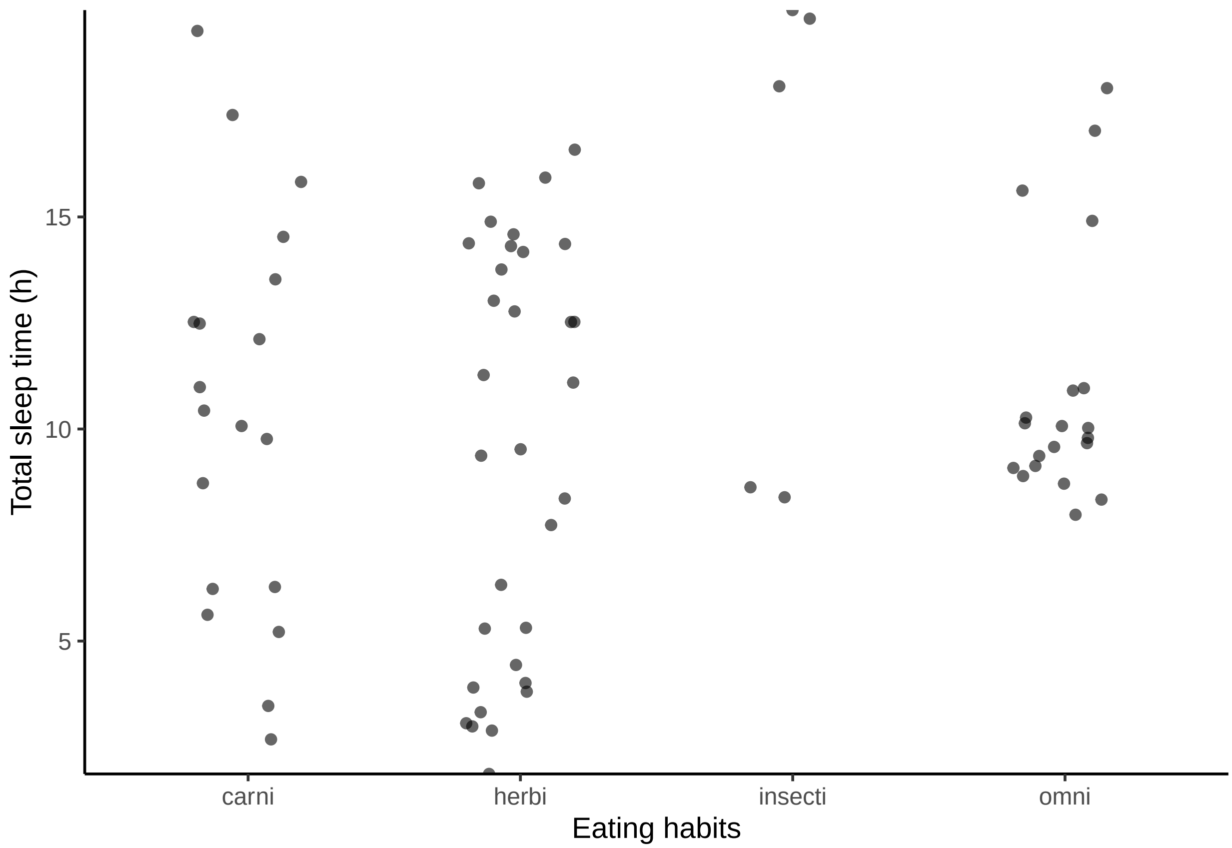
```
# Dynamite plot
d <- ggplot(sleep, aes(vore, sleep_total)) +
  scale_y_continuous("Total sleep time (h)",  expand = c(0, 0)) +
  scale_x_discrete("Eating habits") +
  theme_classic()

d +
  stat_summary(fun.y = mean, geom = "bar", fill = "grey50") +
  stat_summary(fun.data = mean_sdl, mult = 1, geom = "errorbar", width = 0.2)
```

```
## Warning: Ignoring unknown parameters: mult
```



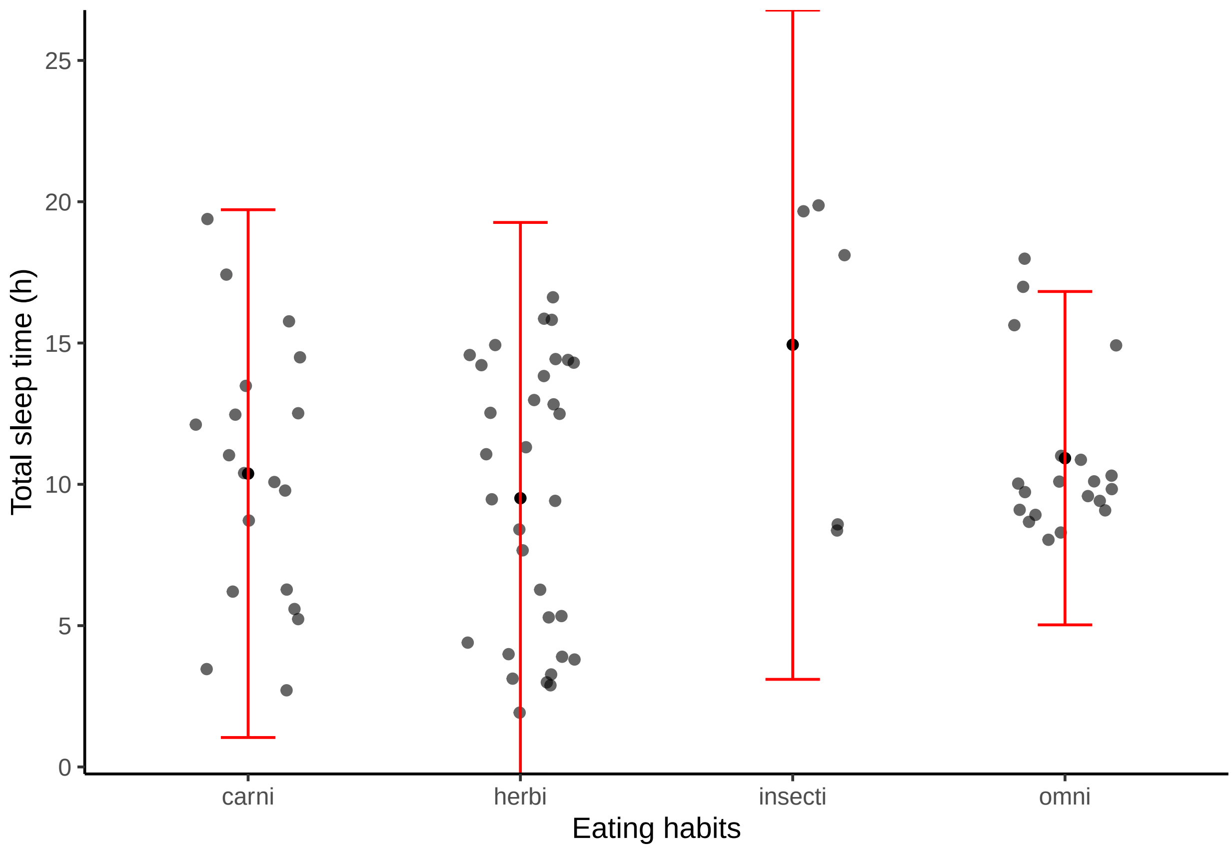
Individual data points

```
d + geom_point(alpha = 0.6, position = position_jitter(width = 0.2))
```



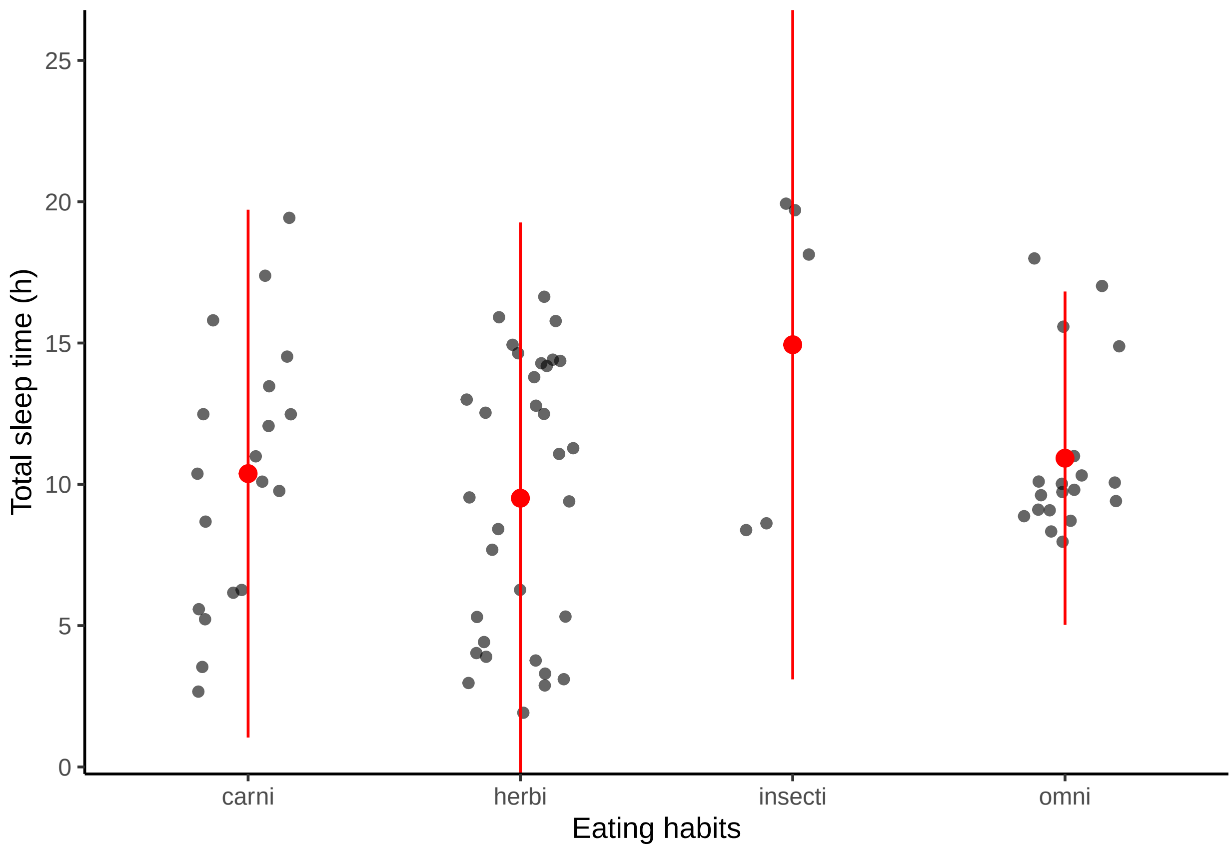

```
# error bar
d +
   geom_point(alpha = 0.6, position = position_jitter(width = 0.2)) +
   stat_summary(fun.y = mean, geom = "point", fill = "red") +
   stat_summary(fun.data = mean_sdl, mult = 1, geom = "errorbar", width = 0.2, col = "red")
```

```
## Warning: Ignoring unknown parameters: mult
```



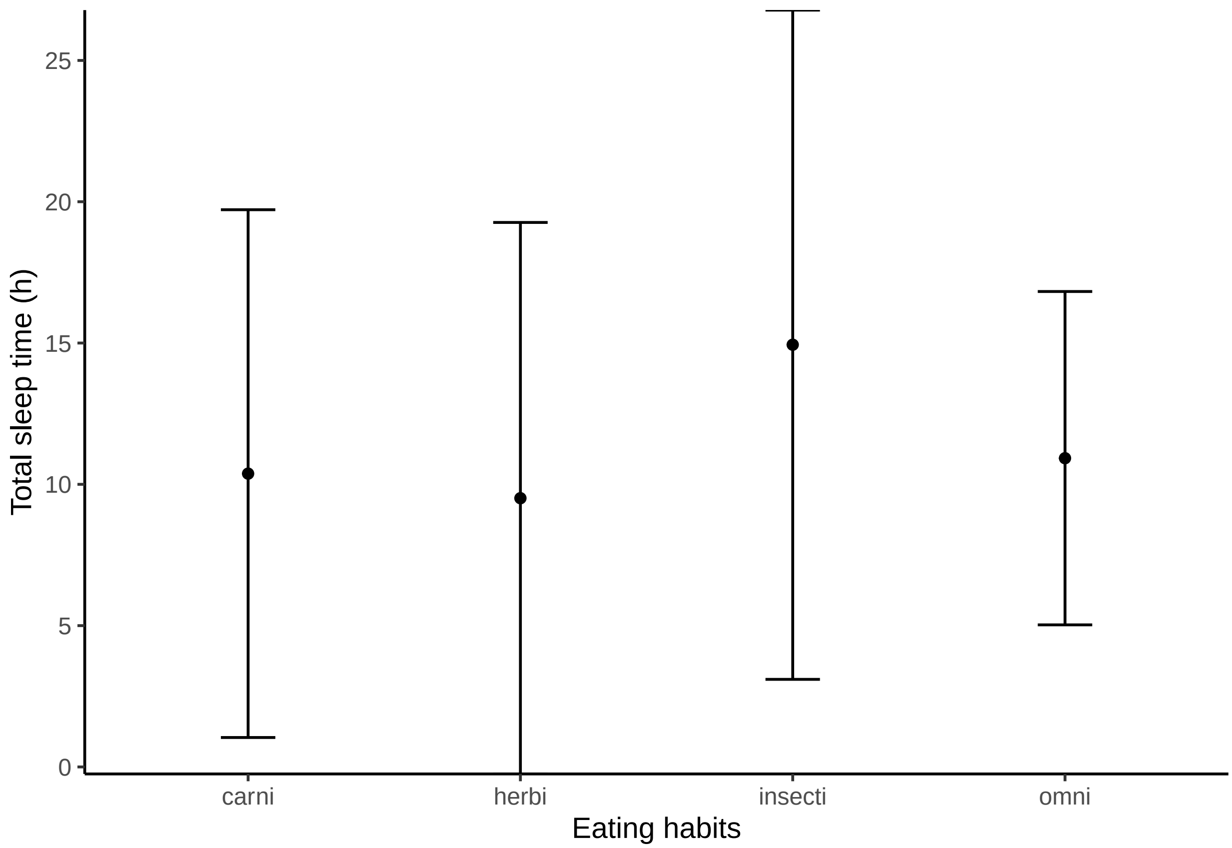

```r
# pointrange
d +
  geom_point(alpha = 0.6, position = position_jitter(width = 0.2)) +
  stat_summary(fun.data = mean_sdl, mult = 1, width = 0.2, col = "red")
```

```
## Warning: Ignoring unknown parameters: mult, width
```




```
# Without data points
d +
  stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.data = mean_sdl, mult = 1, geom = "errorbar", width = 0.2)
```

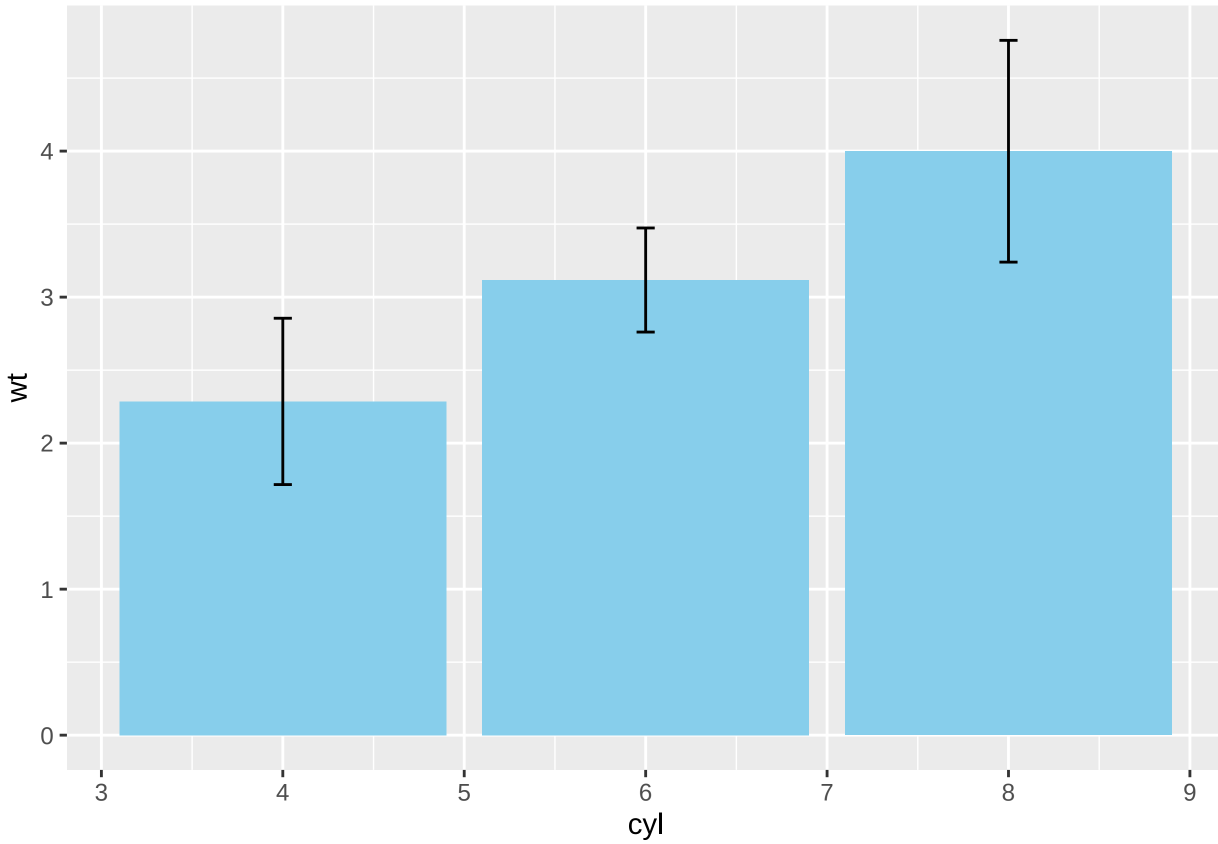
```
## Warning: Ignoring unknown parameters: mult
```



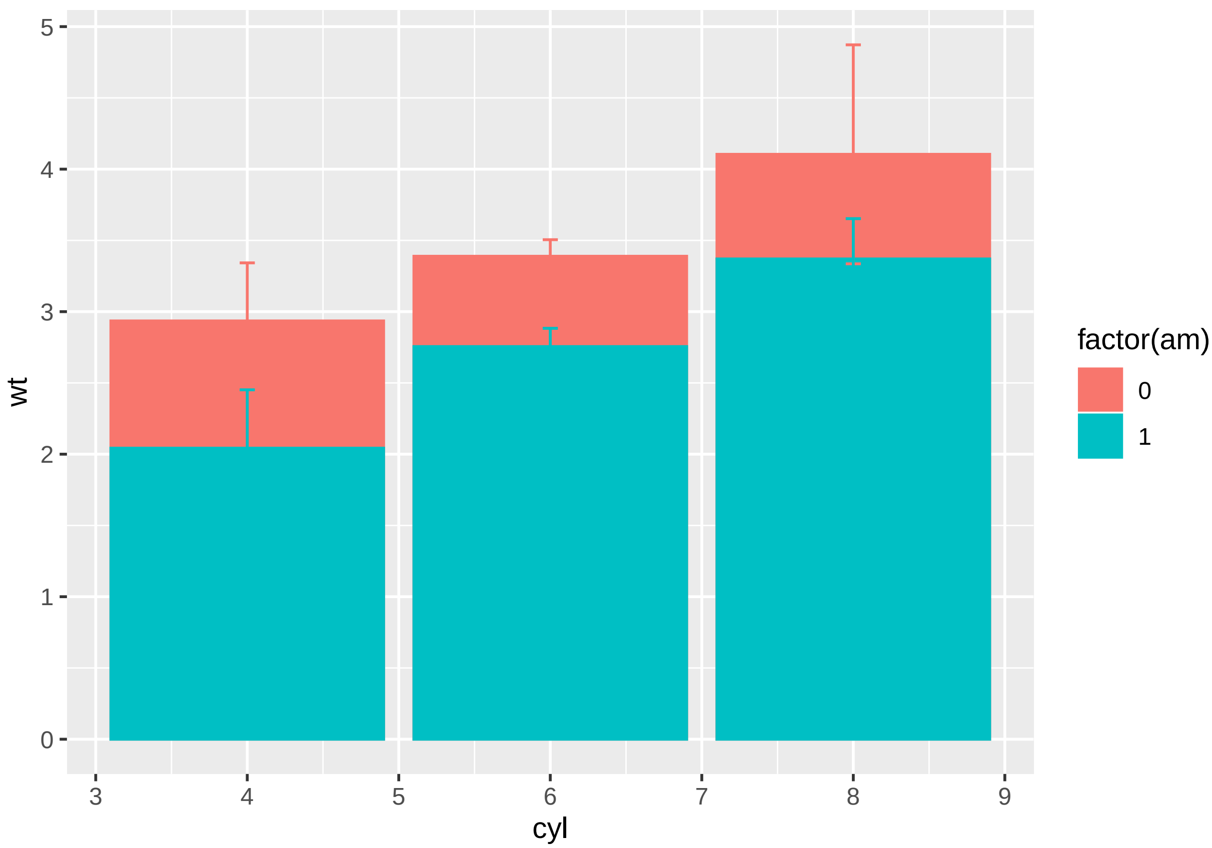
Practice

We saw "dynamite plots" (bar plots with error bars) are NOT well suited for their intended purpose of depicting distributions. If you really want error bars on bar plots, you can still get that. However, you'll need to set the positions manually. A point geom will typically serve you much better.

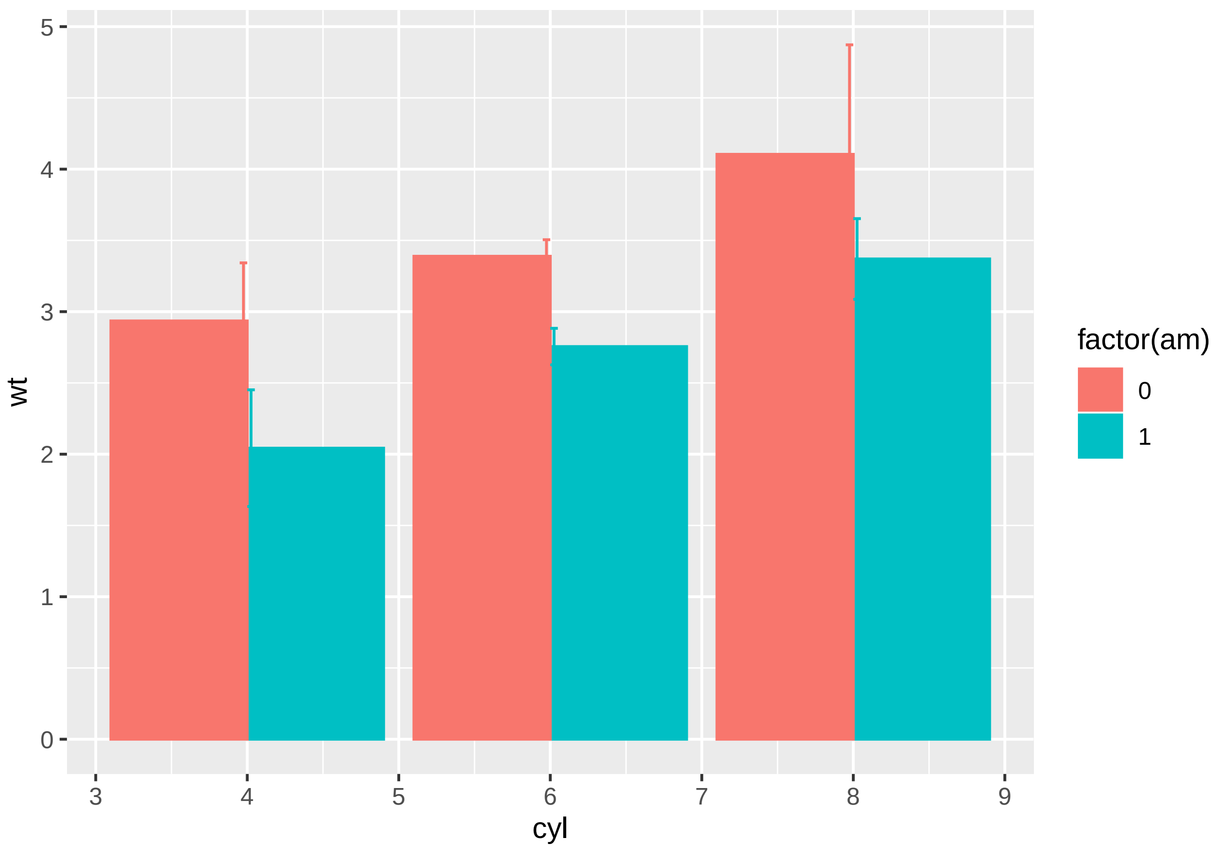
```
# Base layers
m <- ggplot(mtcars, aes(x = cyl, y = wt))

# Draw dynamite plot
m +
  stat_summary(fun.y = mean, geom = "bar", fill = "skyblue") +
  stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1), geom = "errorbar", width = 0.1)
```



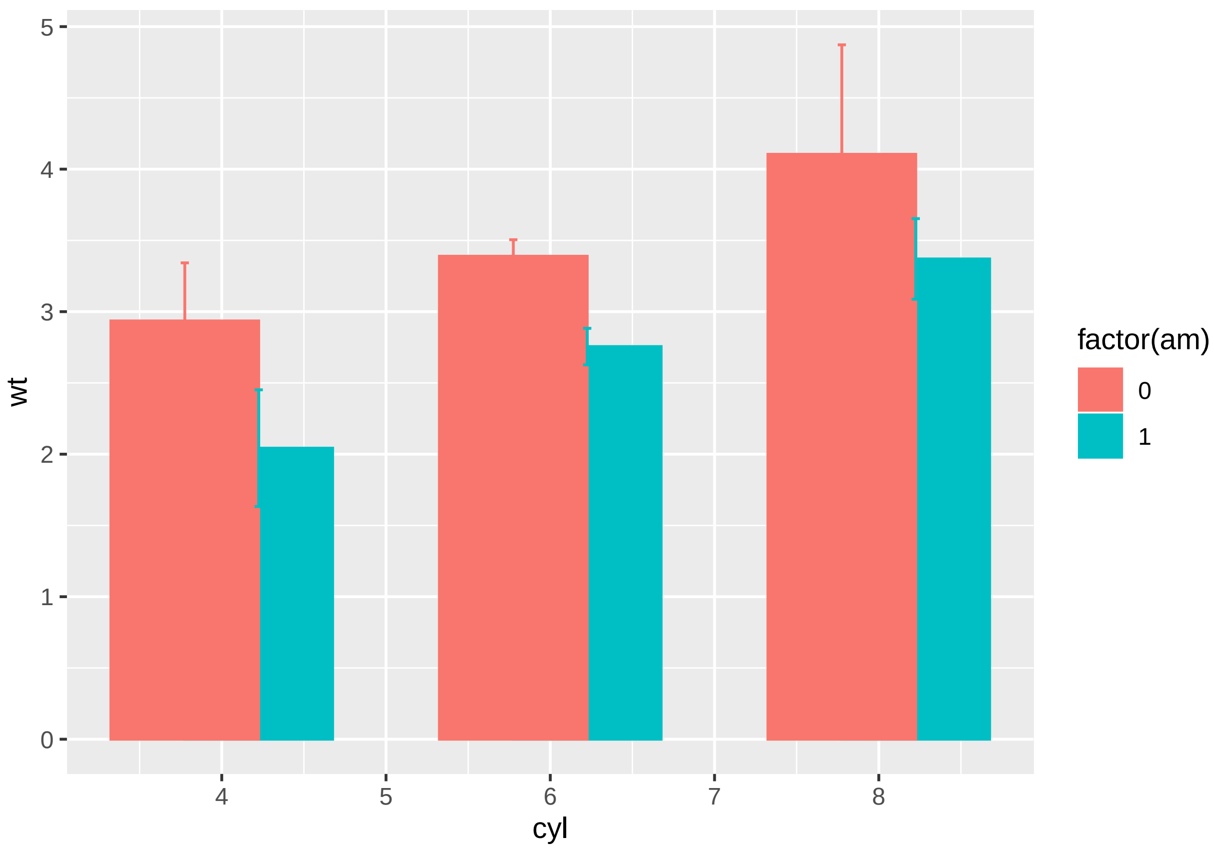

```r
# Base layers
m <- ggplot(mtcars, aes(x = cyl,y = wt, col = factor(am), fill = factor(am)))

# Plot 1: Draw dynamite plot
m +
  stat_summary(fun.y = mean, geom = "bar") +
  stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1), geom = "errorbar", width = 0.1)
```




```r
# Plot 2: Set position dodge in each stat function
m +
  stat_summary(fun.y = mean, geom = "bar", position = "dodge") +
  stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1),
               geom = "errorbar", width = 0.1, position = "dodge")
```




```r
# Set your dodge posn manually
posn.d <- position_dodge(0.9)

# Plot 3: Redraw dynamite plot
m +
  stat_summary(fun.y = mean, geom = "bar", position = posn.d) +
  stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1), geom = "errorbar", width = 0.1, position
```



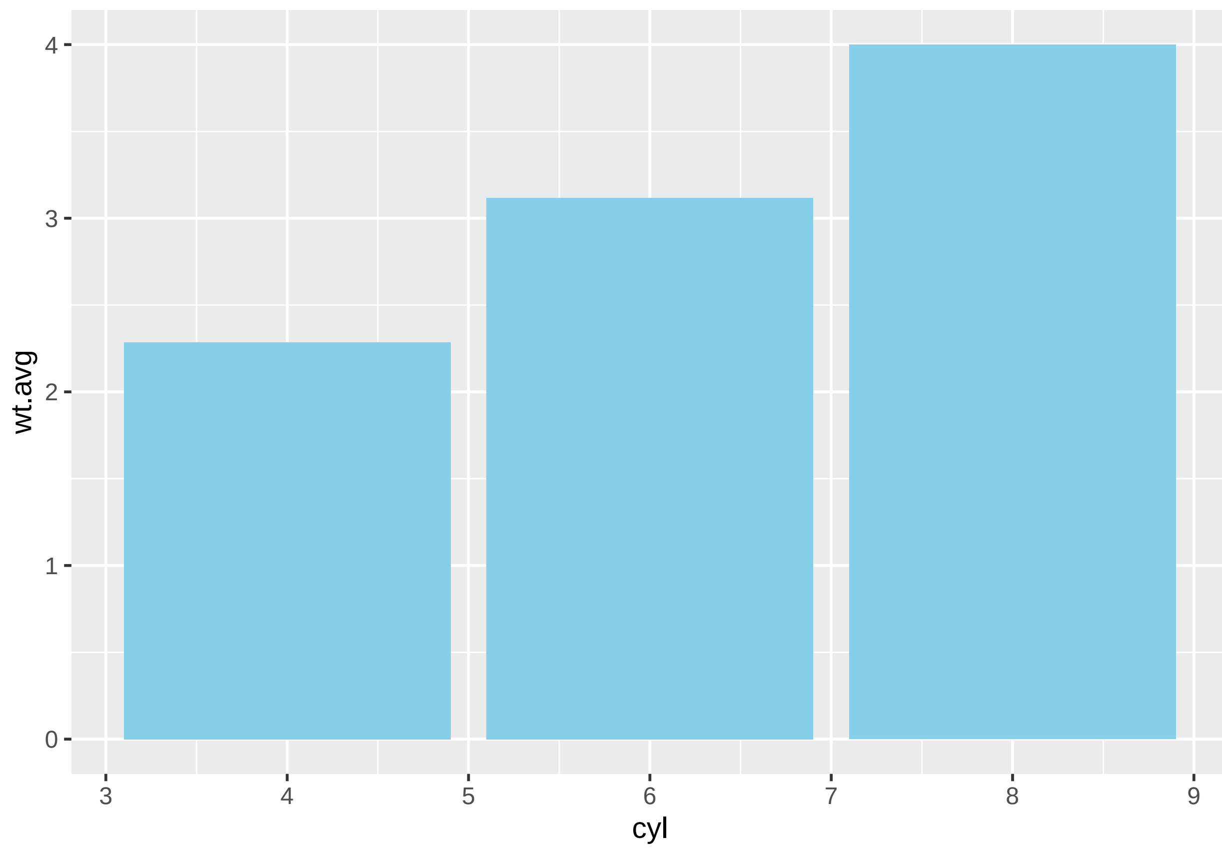
stat_summary() doesn't keep track of the count. stat_sum() does (that's the whole point), but it's difficult to access. In this case, the most straightforward thing to do is calculate exactly what we want to plot beforehand. For this exercise we've created a summary data frame called mtcars.cyl which contains the average (wt.avg), standard deviations (sd) and count (n) of car weights, according to cylinders, cyl. It also contains the proportion (prop) of each cylinder represented in the entire dataset. Use the console to familiarize yourself with the mtcars.cyl data frame.

```
library(dplyr)
```

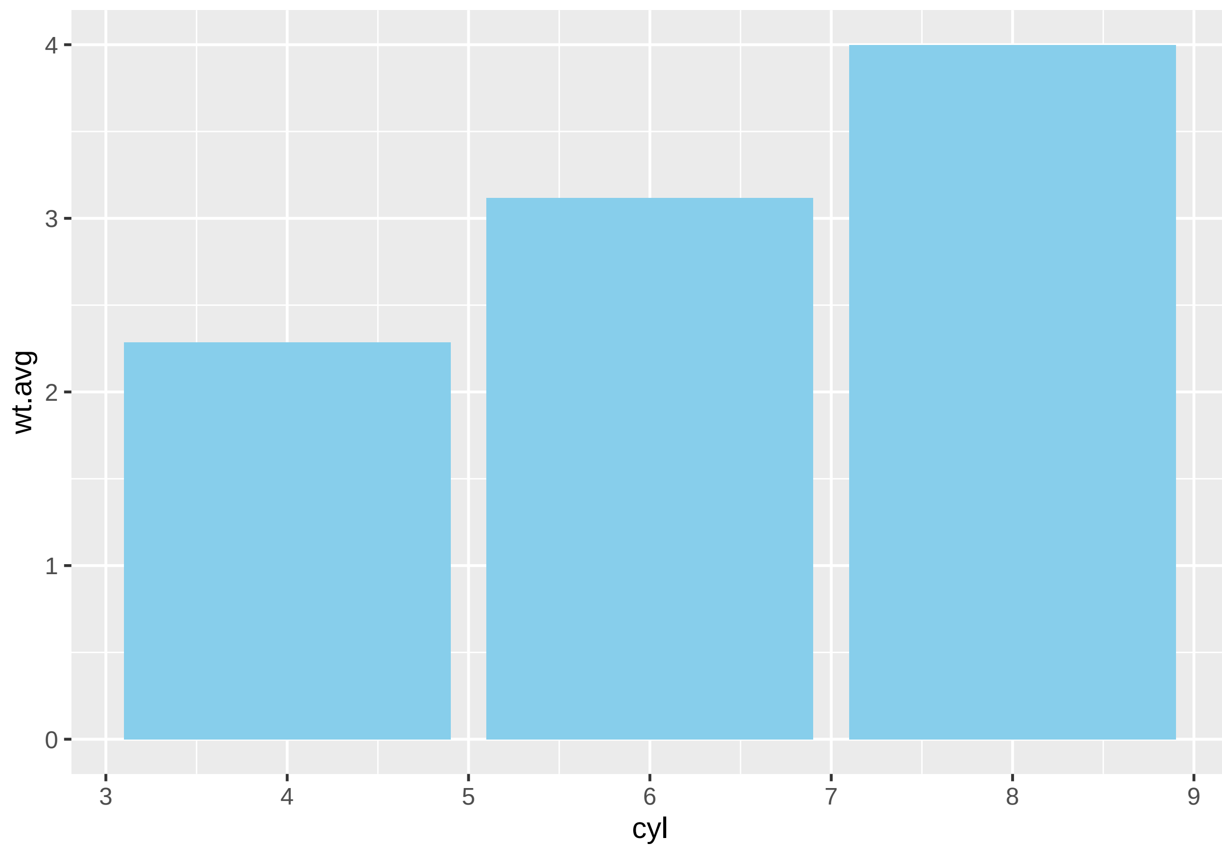
```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
mtcars1 <- mtcars[,c("cyl","wt")]
mtcars.cyl <- mtcars1 %>% group_by(cyl) %>% summarise(wt.avg = mean(wt), sd = sd(wt), n = n()) %>% muta


# Base layers
```

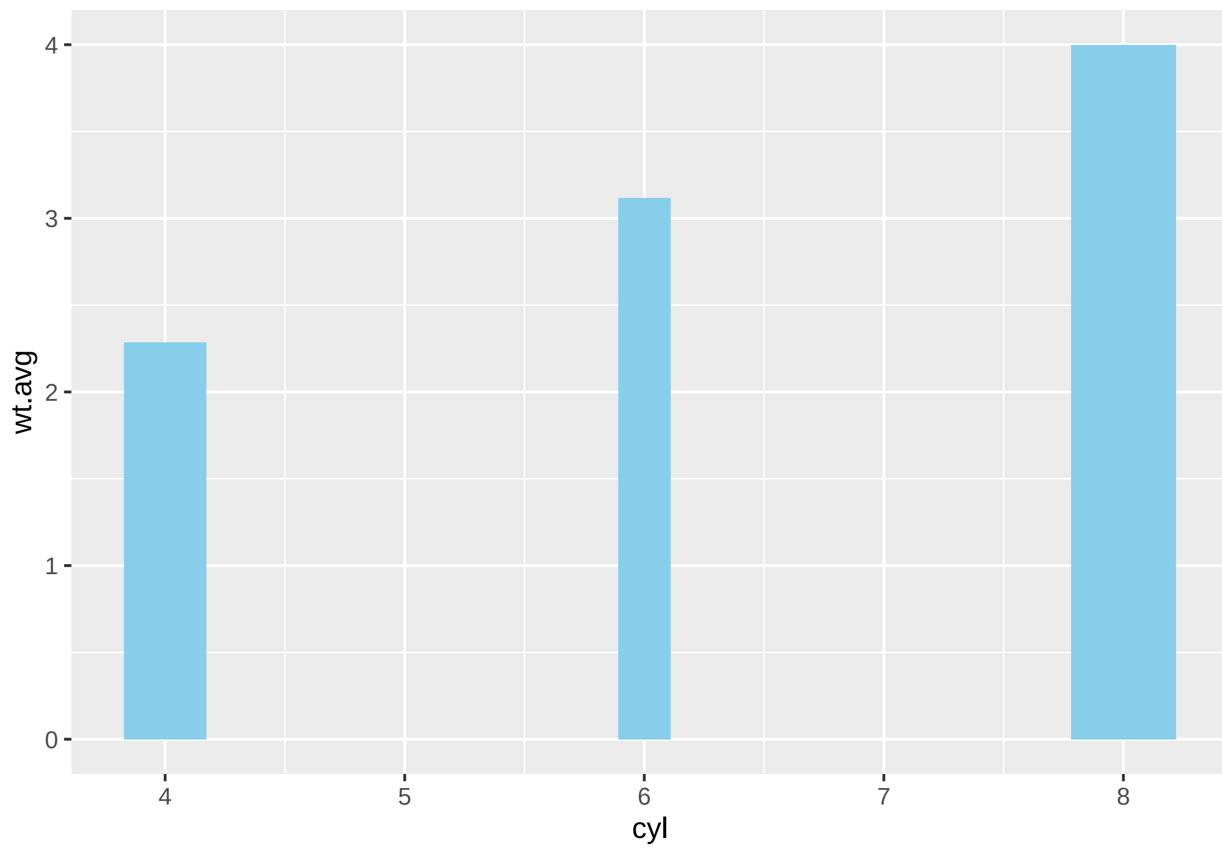
```r
m <- ggplot(mtcars.cyl, aes(x = cyl, y = wt.avg))

# Plot 1: Draw bar plot with geom_bar
m + geom_bar(stat = "identity", fill = "skyblue")
```



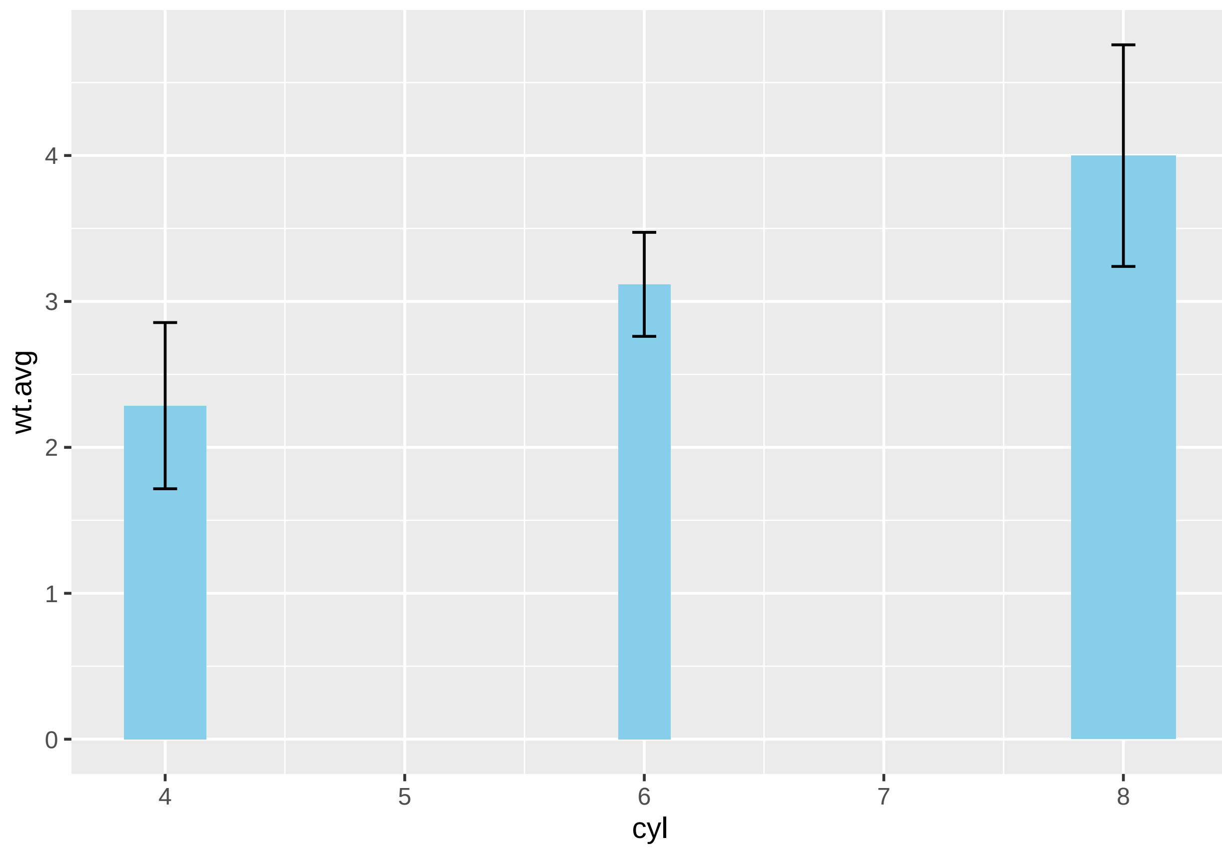

```r
# Plot 2: Draw bar plot with geom_col
m + geom_col(fill = "skyblue")
```




```r
# Plot 3: geom_col with variable widths.
m + geom_col(fill = "skyblue", width = mtcars.cyl$prop)
```

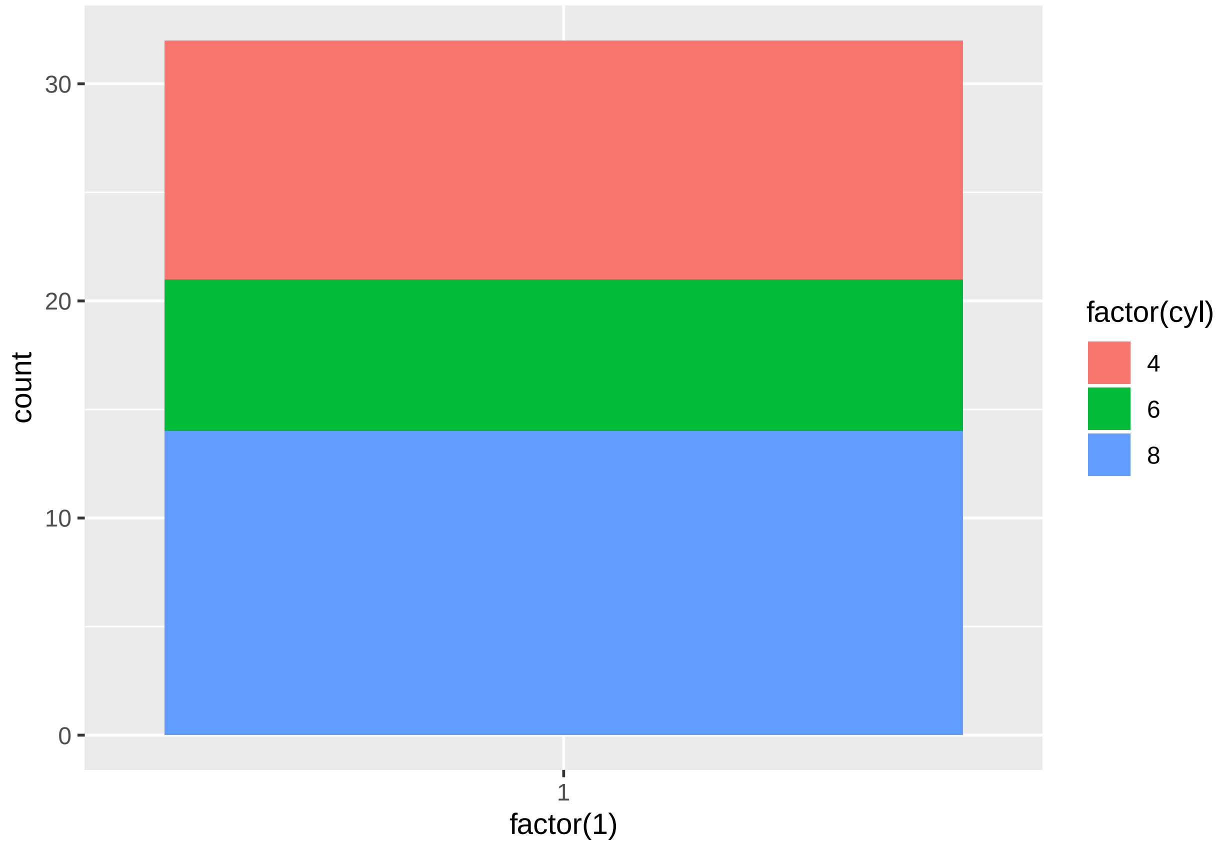



```
# Plot 4: Add error bars
m + geom_col(fill = "skyblue", width = mtcars.cyl$prop) +
  geom_errorbar(aes(ymin = wt.avg - sd, ymax = wt.avg + sd), width = 0.1)
```

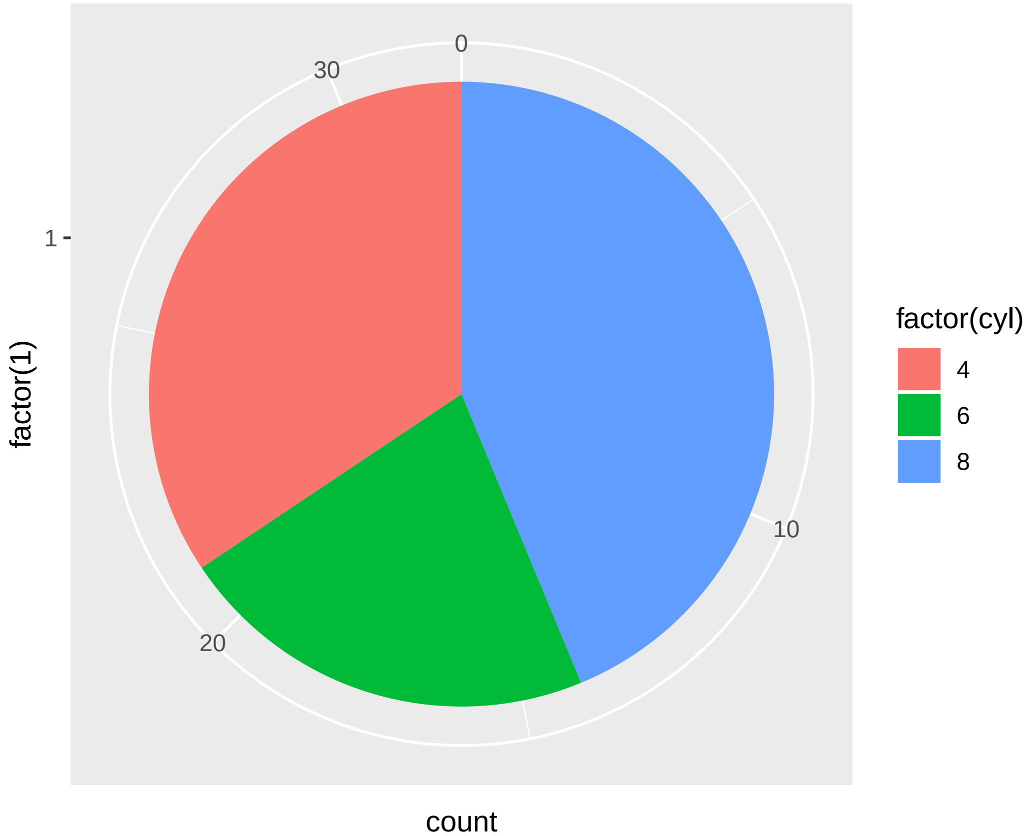


pie charts

```r
ggplot(mtcars, aes(x = factor(1), fill = factor(cyl))) +
  geom_bar(width = 1)
```



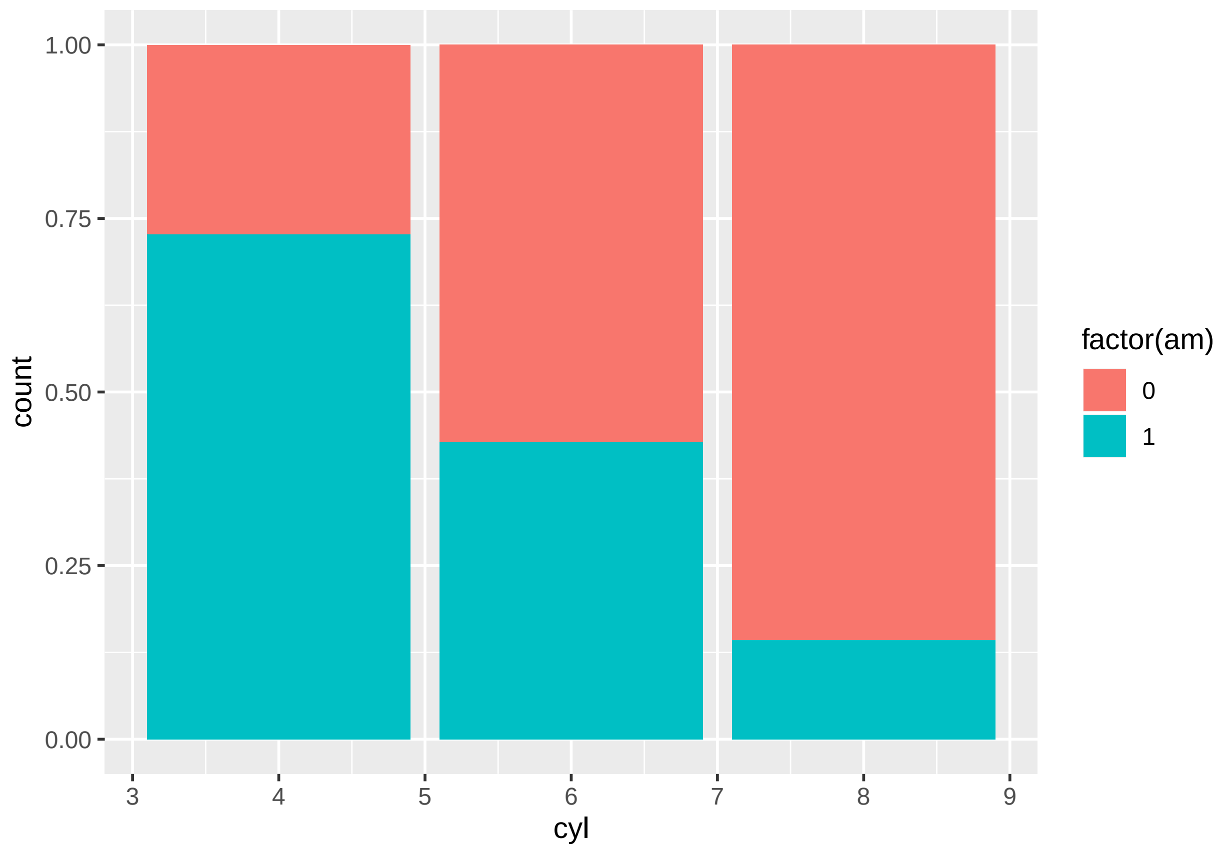

```r
ggplot(mtcars, aes(x = factor(1), fill = factor(cyl))) +
  geom_bar(width = 1) +
  coord_polar(theta = "y")
```



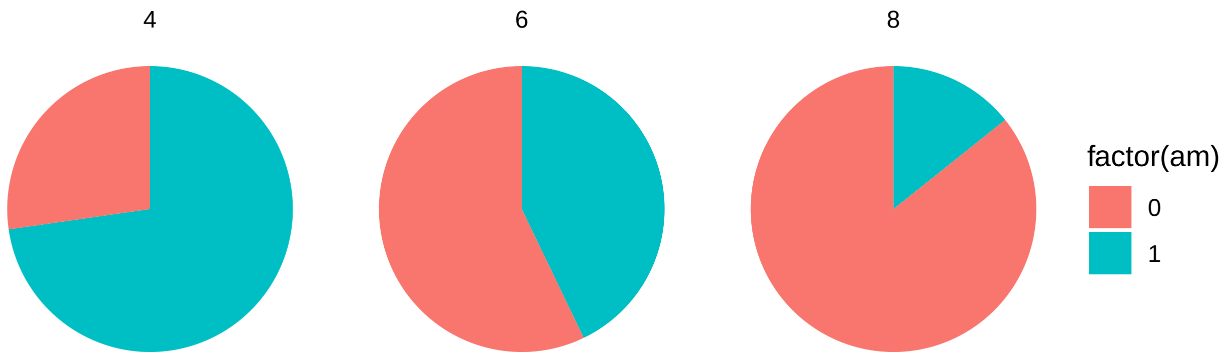
Practice

Pie Charts(1)

```
# Bar chart
ggplot(mtcars, aes(x = cyl, fill = factor(am))) +
  geom_bar(position = "fill")
```




```r
# Convert bar chart to pie chart
ggplot(mtcars, aes(x = factor(1), fill = factor(am))) +
  geom_bar(position = "fill") +
  facet_grid(. ~ cyl) + # Facets
  coord_polar(theta = "y") + # Coordinates
  theme_void() # theme
```



Pie Charts (2)

In the previous example, we looked at one categorical variable (am) as a proportion of another (cyl). Here, we're interested in two or more categorical variables, independent of each other. The many pie charts in the viewer is an unsatisfactory visualization. We're interested in the relationship between all these variables (e.g. where are 8 cylinder cars represented on the Transmission, Gear and Carburetor variables?) Perhaps we also want continuous variables, such as weight. How can we combine all this information?

The trick is to use a parallel coordinates plot, like this one. Each variable is plotted on its own parallel axis. Individual observations are connected with lines, colored according to a variable of interest. This is a surprisingly useful visualization since we can combine many variables, even if they are on entirely different scales.

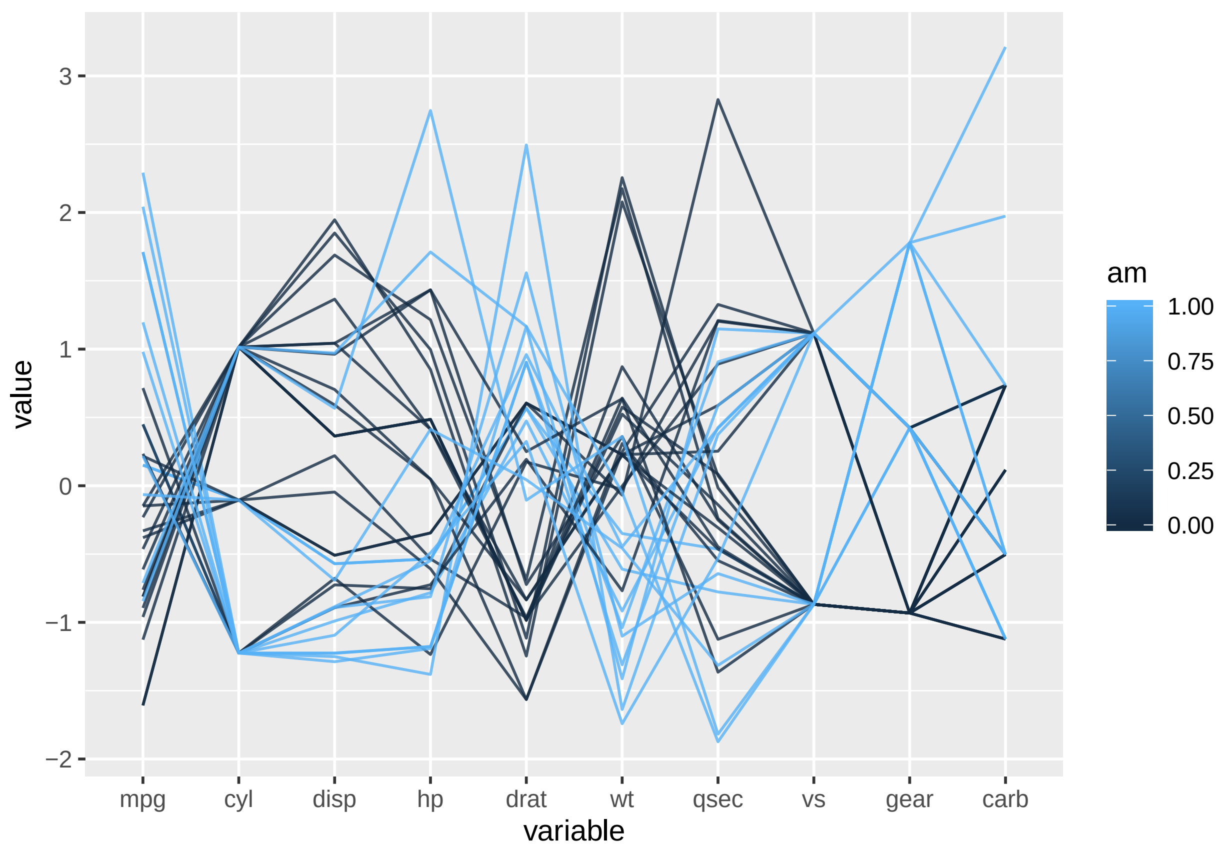
A word of caution though: typically it is very taboo to draw lines in this way. It's the reason why we don't draw lines across levels of a nominal variable - the order, and thus the slope of the line, is meaningless. Parallel plots are a (very useful) exception to the rule!

1. am is variable 9 in the mtcars data frame. Assign this number to group_by_am. The object my_names_am will contain a numeric vector from 1 - 11 excluding the column with am. These will be our parallel axes.
2. Fill in the ggparcoord() function.
3. The first argument is the data frame you're using. mtcars in our case.
4. The second argument is the number of the columns to plot (use my_names_am),
5. groupColumn specifies the column number of the grouping variable (use group_by_am)
6. alpha, the opacity, should be set to 0.8

```r
# Parallel coordinates plot using GGally
# install.packages("GGally")
library("GGally")
```

```
## Warning: package 'GGally' was built under R version 3.6.3

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

##
## Attaching package: 'GGally'

## The following object is masked from 'package:dplyr':
##
##     nasa
```

```r
# All columns except am
group_by_am <- 9
my_names_am <- (1:11)[-group_by_am]

# Basic parallel plot - each variable plotted as a z-score transformation
ggparcoord(mtcars, my_names_am, groupColumn = group_by_am, alpha = 0.8)
```

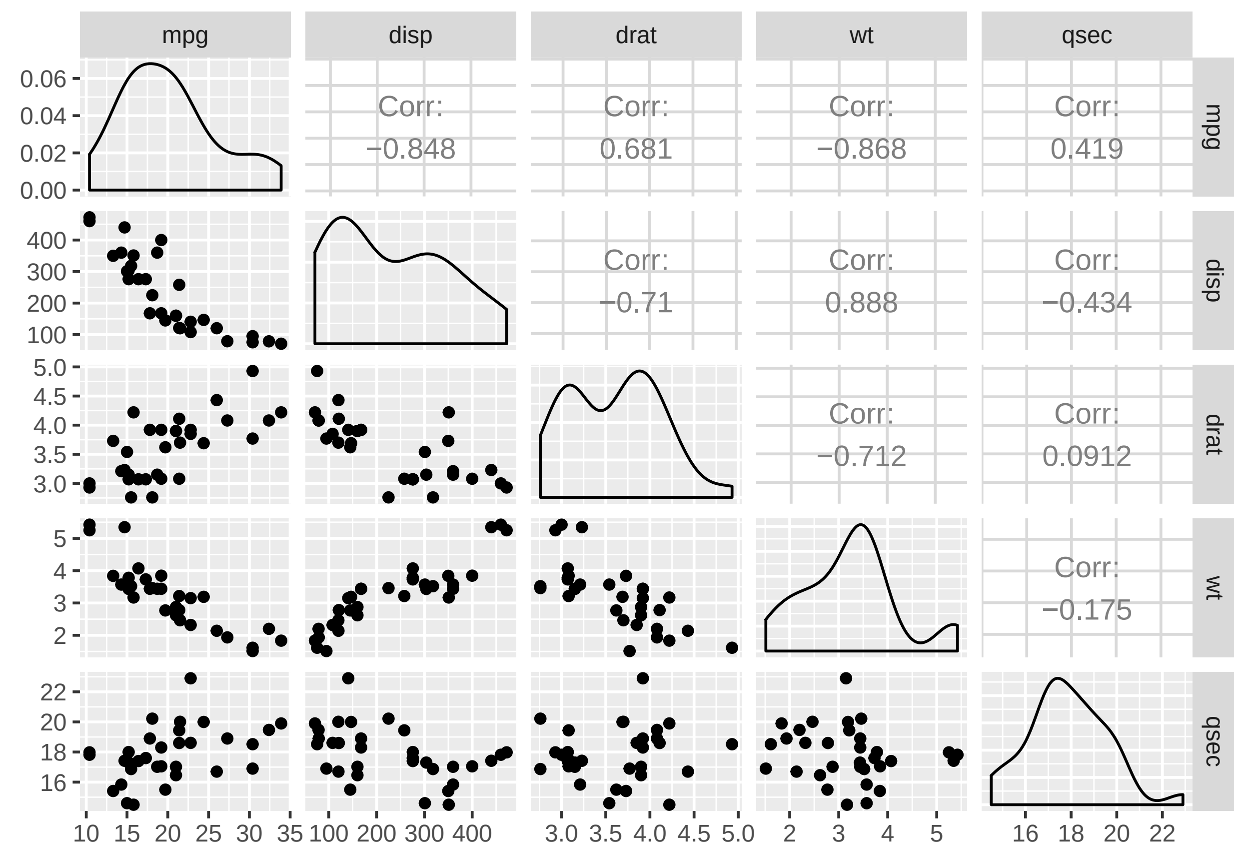


Plot Matrix (1)

The parallel coordinate plot from the last exercise is an excellent example of an exploratory plot. It presents a massive amount of information and allows the specialist to explore many relationships all at once. Another great example is a plot matrix (a SPLOM, from scatter plot matrix).

GGally::ggpairs(mtcars2) will produce the plot of a selection of the mtcars dataset, mtcars2, in the viewer. Depending on the nature of the dataset a specific plot type will be produced and if both variables are continuous the correlation (rho) will also be calculated.

```r
mtcars2 <- mtcars[,c("mpg","disp","drat","wt","qsec")]
GGally::ggpairs(mtcars2)
```



Heat maps

```r
library(lattice)
library(tidyr)
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse 1.2.1 --
```

```
## v tibble  2.1.3     v purrr   0.3.2
## v readr   1.3.1     v stringr 1.4.0
## v tibble  2.1.3     v forcats 0.4.0
```

```
## -- Conflicts -------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
head(barley)
```

```
##      yield   variety year            site
## 1 27.00000 Manchuria 1931 University Farm
## 2 48.86667 Manchuria 1931          Waseca
## 3 27.43334 Manchuria 1931          Morris
## 4 39.93333 Manchuria 1931       Crookston
## 5 32.96667 Manchuria 1931    Grand Rapids
## 6 28.96667 Manchuria 1931          Duluth
```

```r
barley.s <- barley %>% spread(year,yield)
head(barley.s)
```

```
##    variety            site     1932     1931
## 1 Svansota    Grand Rapids 16.63333 29.66667
## 2 Svansota          Duluth 22.23333 25.70000
## 3 Svansota University Farm 27.43334 35.13333
## 4 Svansota          Morris 35.03333 25.76667
## 5 Svansota       Crookston 20.63333 40.46667
## 6 Svansota          Waseca 38.50000 47.33333
```

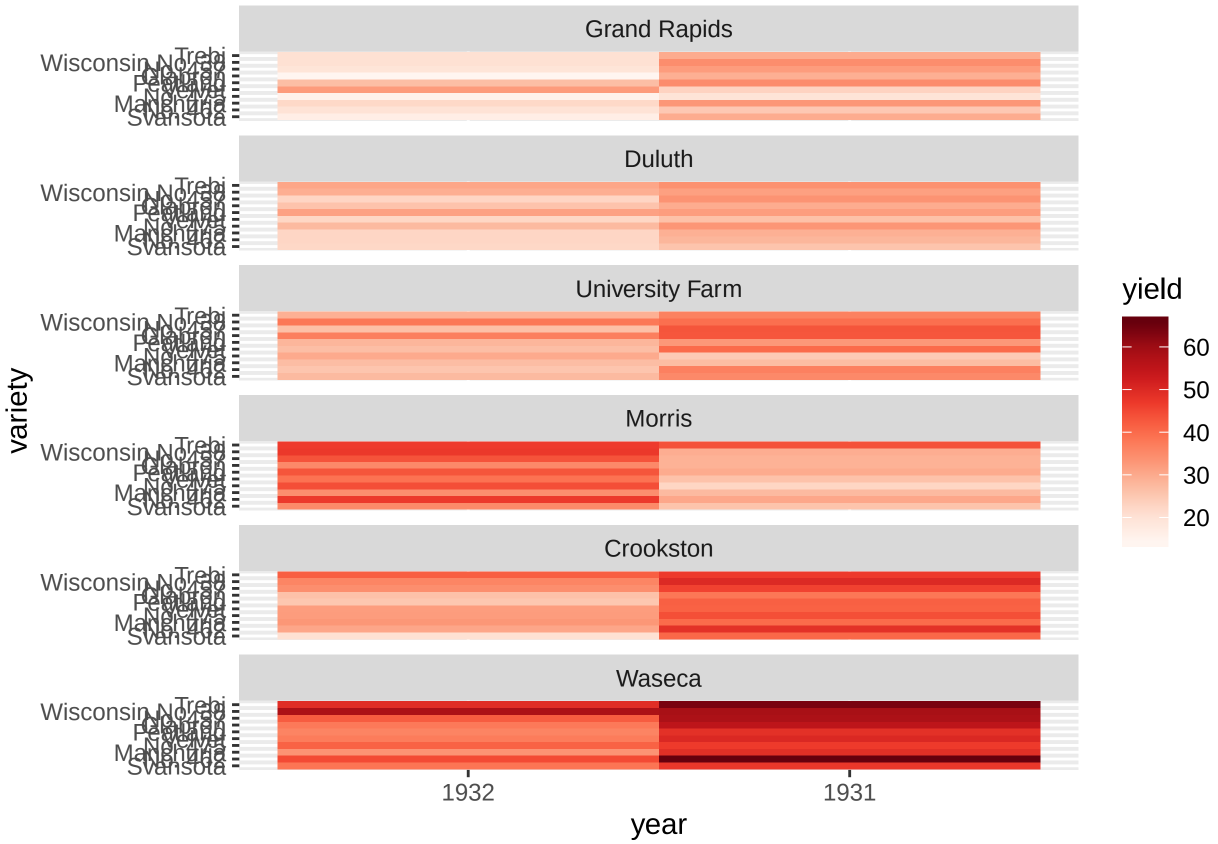
Practice

Add a geom_tile() to build the heat maps.

So far the entire dataset is plotted on one heat map. Add a facet_wrap() function to get a facetted plot. Use the formula ~ site (without the dot!) and set ncol = 1. By default, the names of the farms will be above the panels, not to the side (as we get with facet_grid()).

brewer.pal() from the RColorBrewer package has been used to create a "Reds" color palette. The hexadecimal color codes are stored in the myColors object. Add the scale_fill_gradientn() function and specify the colors argument correctly to give the heat maps a reddish look.

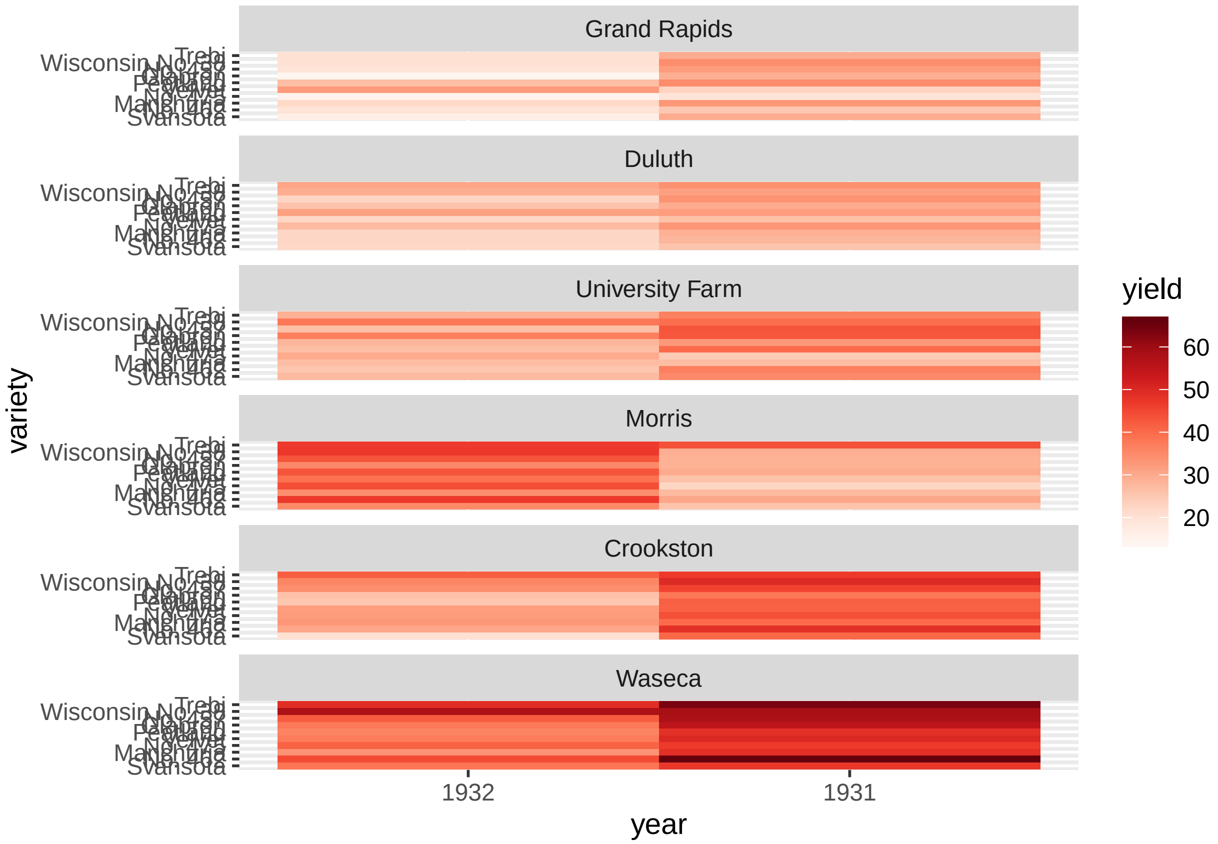
```r
library("RColorBrewer")
# Create color palette
myColors <- brewer.pal(9, "Reds")

# Build the heat map from scratch
ggplot(barley, aes(x = year, y = variety, fill = yield)) +
  geom_tile() + # Geom layer
  facet_wrap( ~ site, ncol = 1) + # Facet layer
  scale_fill_gradientn(colors = myColors) # Adjust colors
```



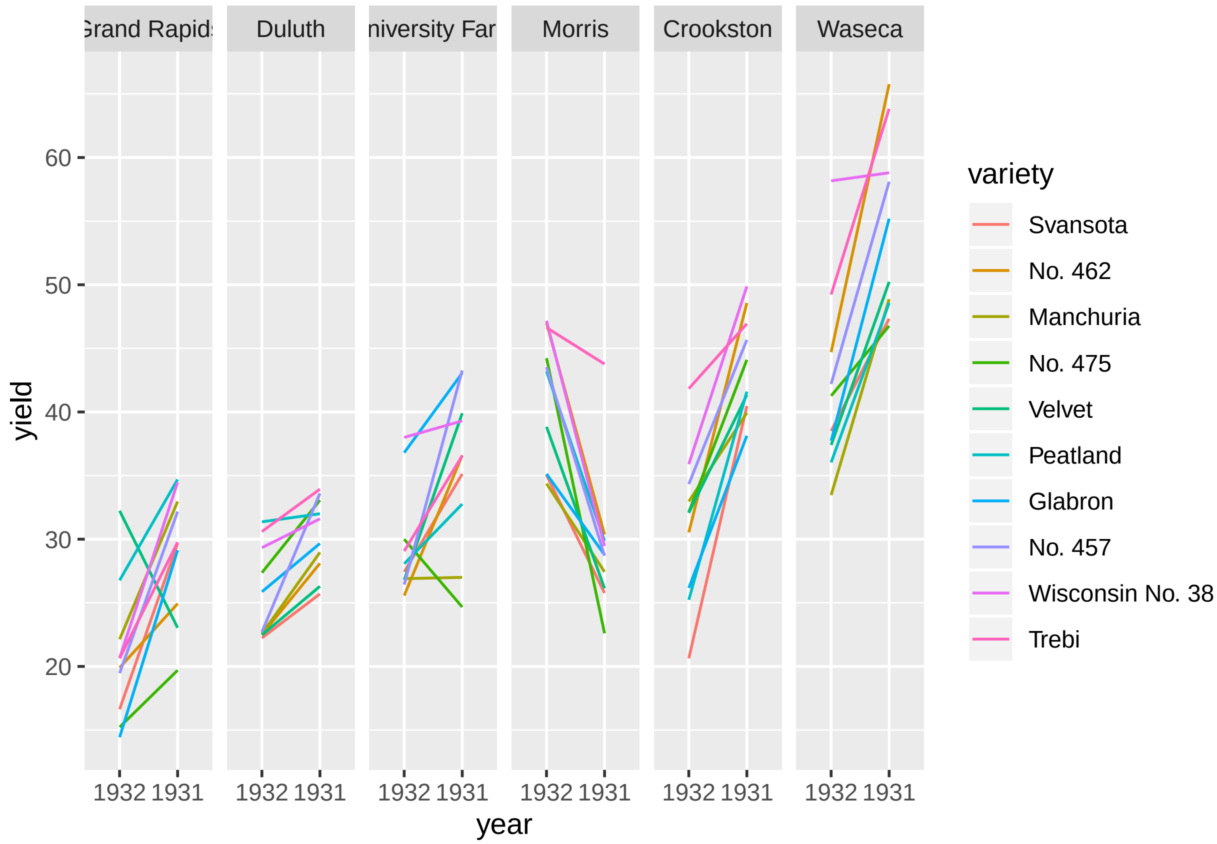
Heat Maps Alternatives (1)

There are several alternatives to heat maps. The best choice really depends on the data and the story you want to tell with this data. If there is a time component, the most obvious choice is a line plot.

```r
# The heat map we want to replace
# Don't remove, it's here to help you!
myColors <- brewer.pal(9, "Reds")
ggplot(barley, aes(x = year, y = variety, fill = yield)) +
  geom_tile() +
  facet_wrap( ~ site, ncol = 1) +
  scale_fill_gradientn(colors = myColors)
```




```r
# Line plot; set the aes, geom and facet
ggplot(barley, aes(x = year, y = yield, col = variety, group = variety)) +
  geom_line() +
  facet_wrap(~ site, nrow = 1)
```

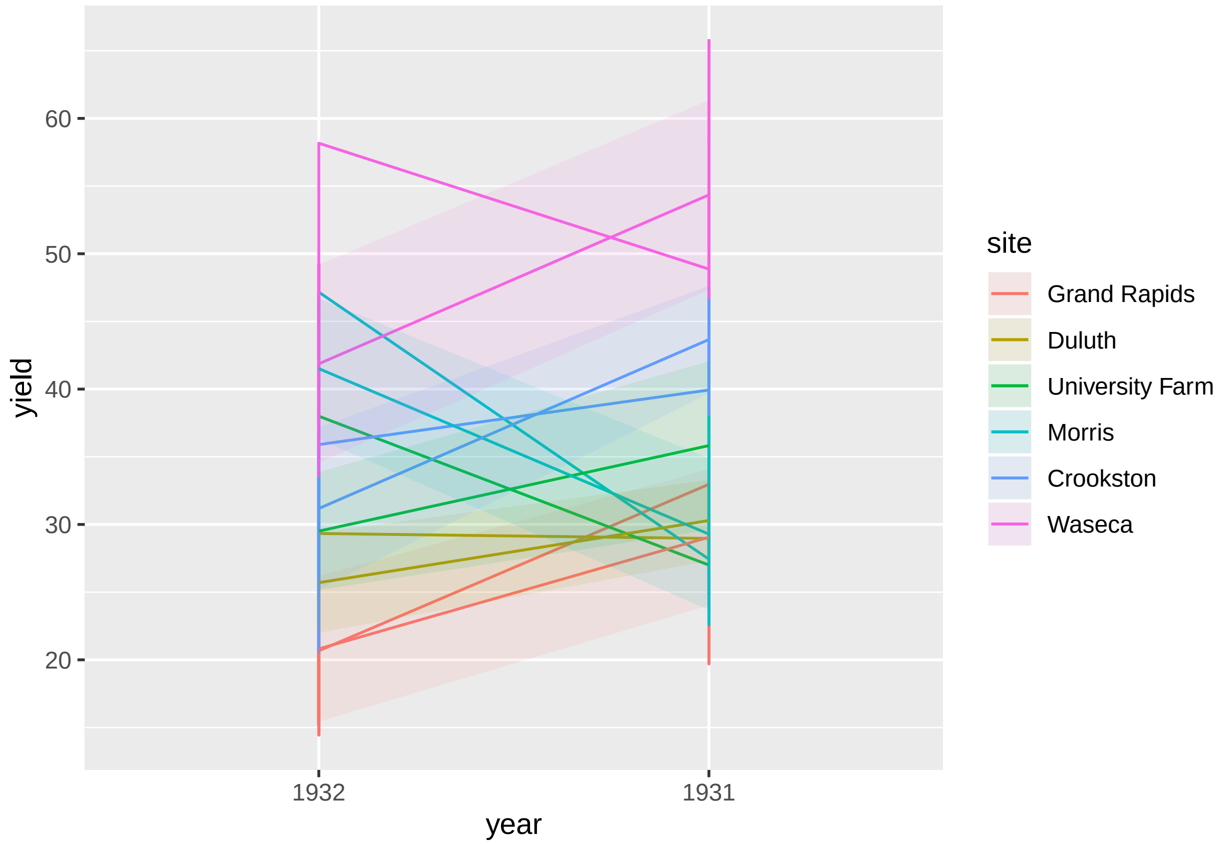


Heat Maps Alternatives (2)

There are two methods for depicting overlapping measurements of spread. You can use dodged error bars or you can use overlapping transparent ribbons (shown in the viewer). In this exercise we'll try to recreate the second option, the transparent ribbons.

1. Base layer: use the barley dataset. Try to come up with the correct mappings for x, y, col, group and fill.
2. Add a stat_summary() function for the mean. Specify fun.y to be mean and set geom to "line".
3. Add a stat_summary() function for the ribbons. Set fun.data = mean_sdl and fun.args = list(mult = 1) to have a ribbon that spans over one standard deviation in both directions. Use geom = "ribbon" and set col = NA and alpha = 0.1.

```
# Create overlapping ribbon plot from scratch

ggplot(barley, aes(x = year, y = yield, col = site, group = site, fill = site)) +
  geom_line() +
  stat_summary(fun.y = mean, geom = "line") +
  stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1),geom = "ribbon", col = NA, alpha = 0.1)
```


60
50
40
30
20
yield
1932
1931
year
site
Grand Rapids
Duluth
University Farm
Morris
Crookston
Waseca