GGplot2 Part 2

Seun Odeyemi

2019-03-17

Contents

Load Libraries

```r
library(readr)
library(dplyr)
library(ggplot2)
library(tidyr)
library(skimr)
library(knitr)
library(kableExtra)
library(RColorBrewer)
```

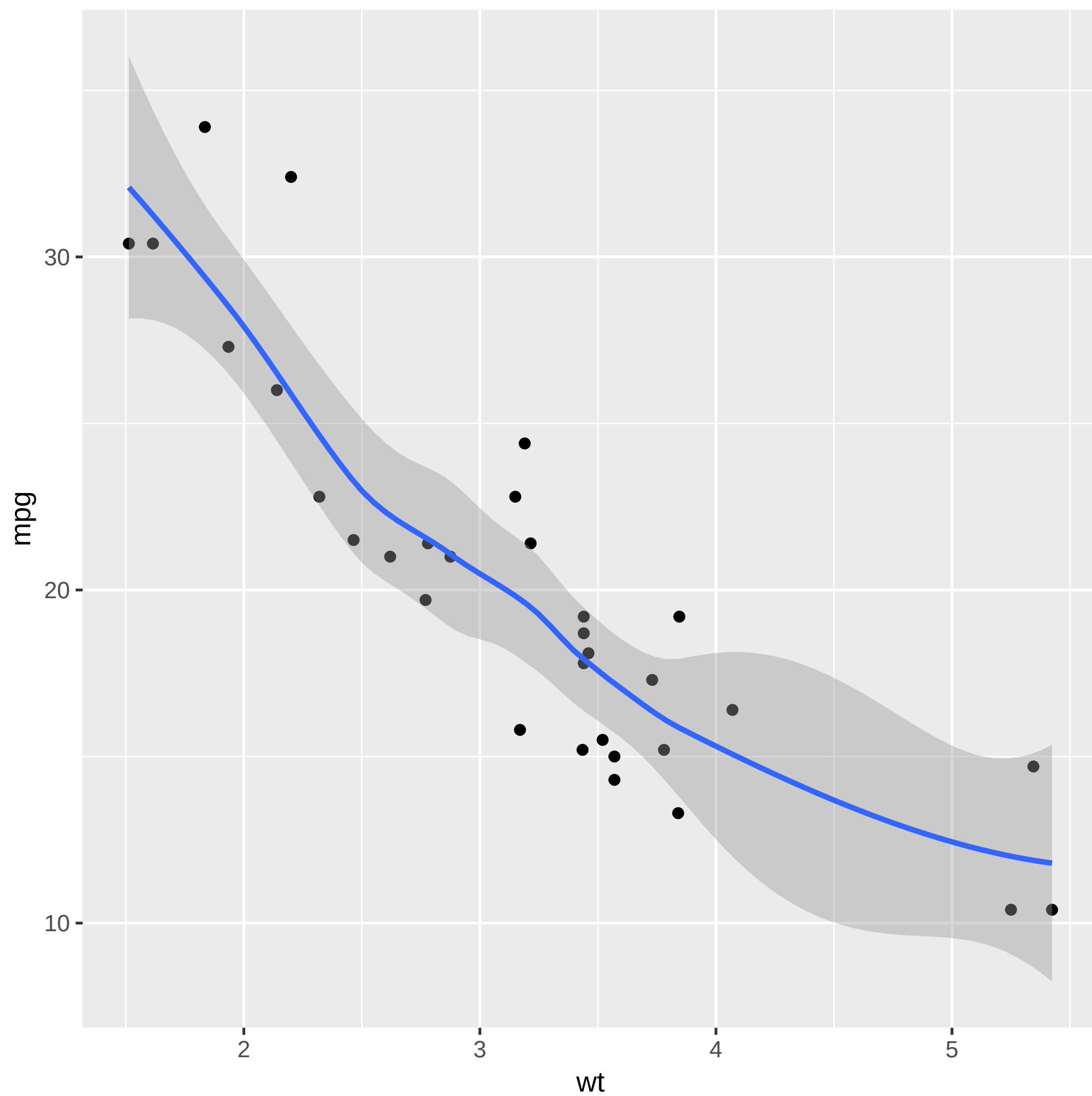
Exploring the mtcars dataset

```r
mtcars_tbl <- as_tibble(mtcars)

glimpse(mtcars_tbl)
#> Observations: 32
#> Variables: 11
#> $ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19....
#> $ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, ...
#> $ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 1...
#> $ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, ...
#> $ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.9...
#> $ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3...
#> $ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 2...
#> $ vs   <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, ...
#> $ am   <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, ...
#> $ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, ...
#> $ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, ...

kable(head(mtcars_tbl)) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)
```

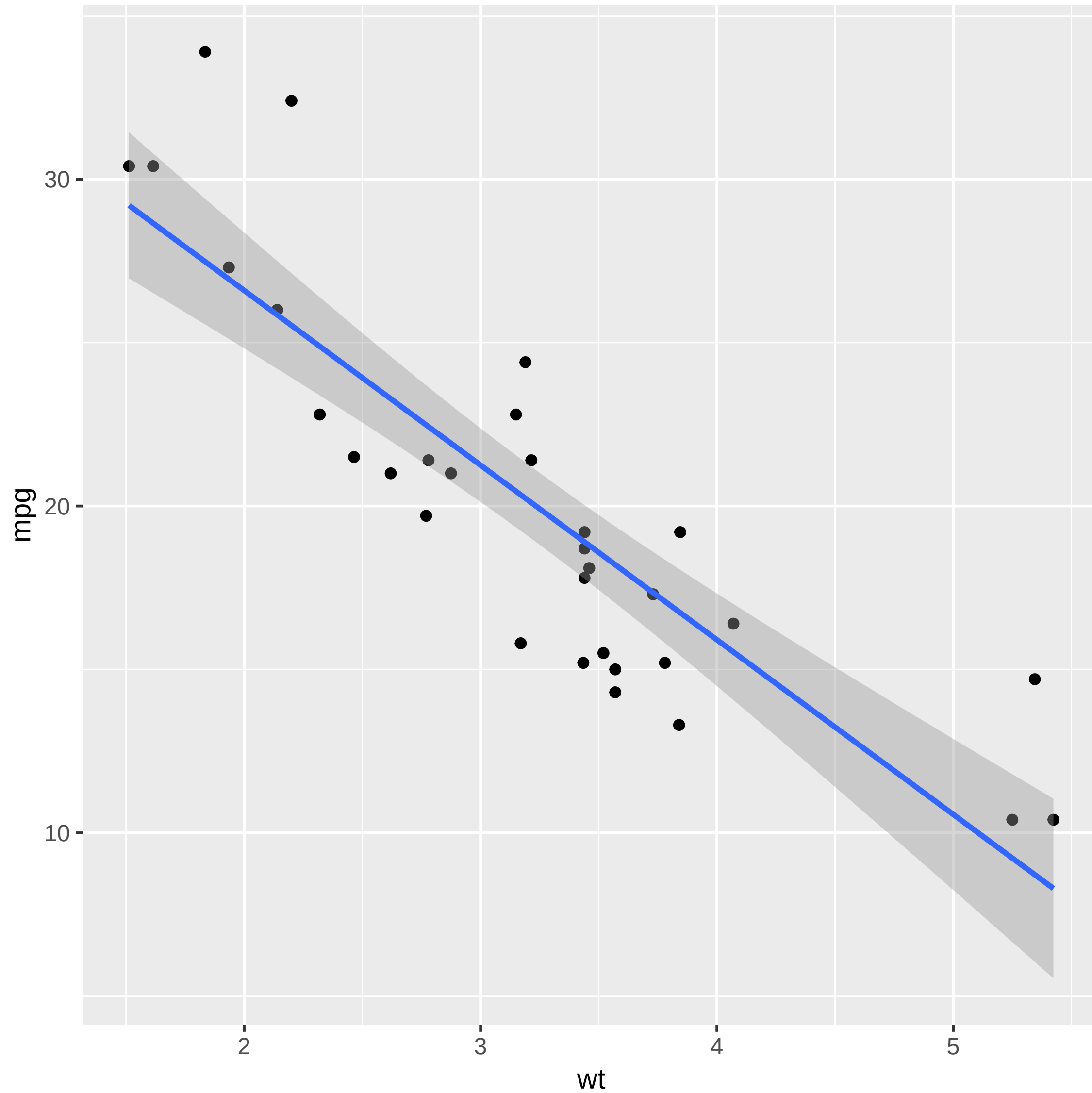
mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Smoothing

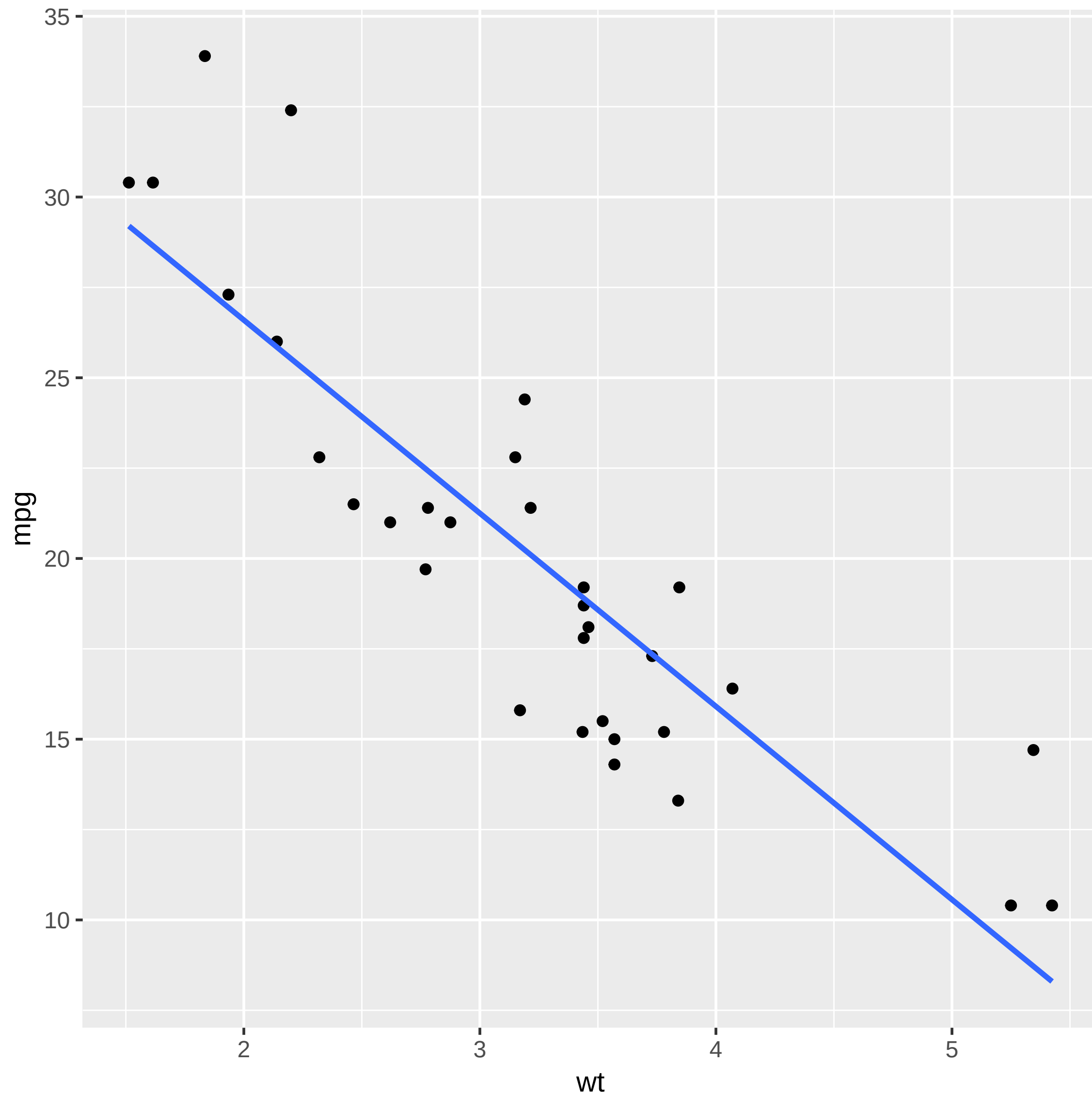
```r
par(mfrow = c(2, 2))

# A scatter plot with LOESS smooth
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "loess")
```



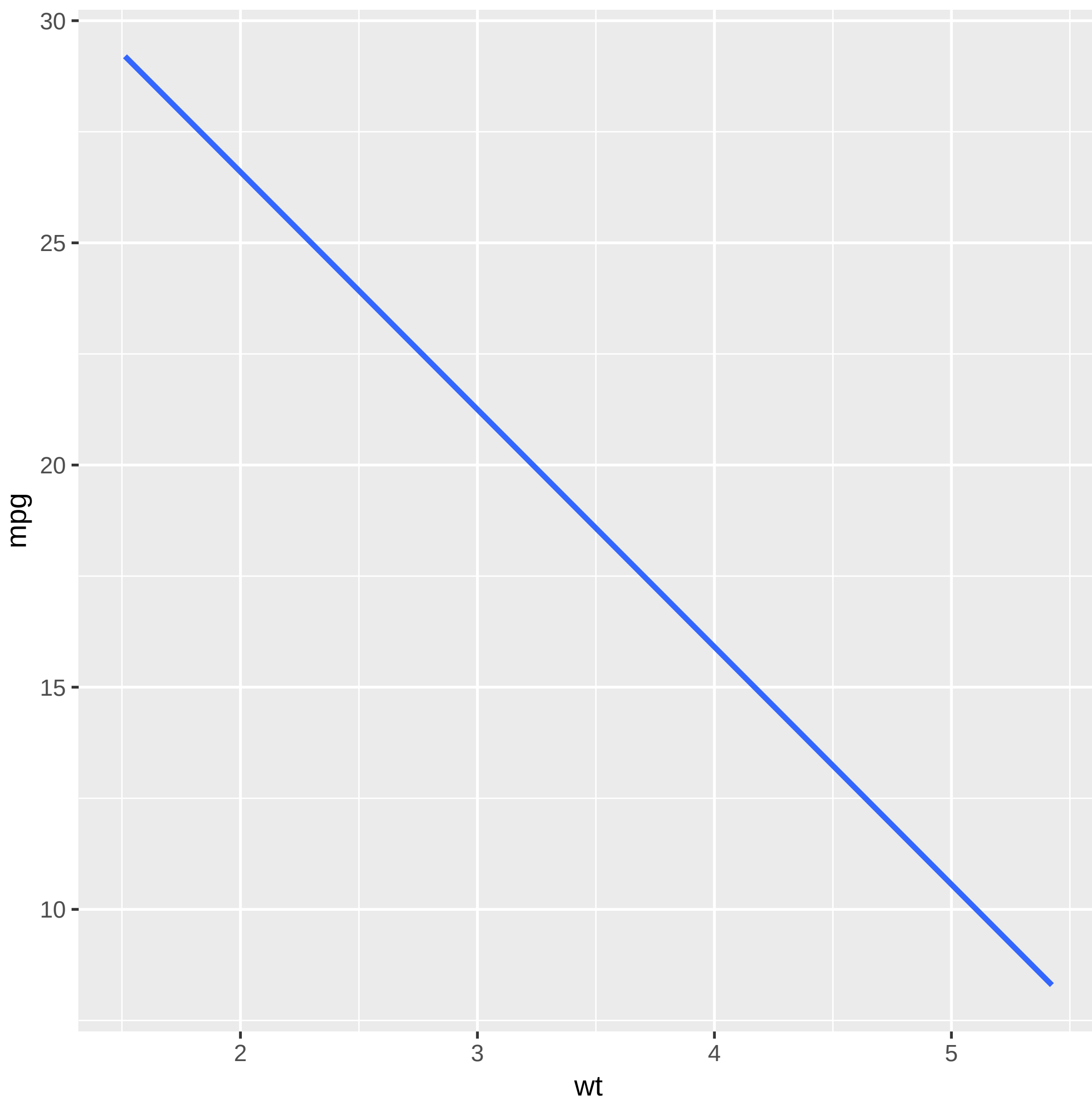

```r
# A scatter plot with an ordinary Least Squares linear model
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm")
```




```
# The previous plot, without CI ribbon
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE)
```



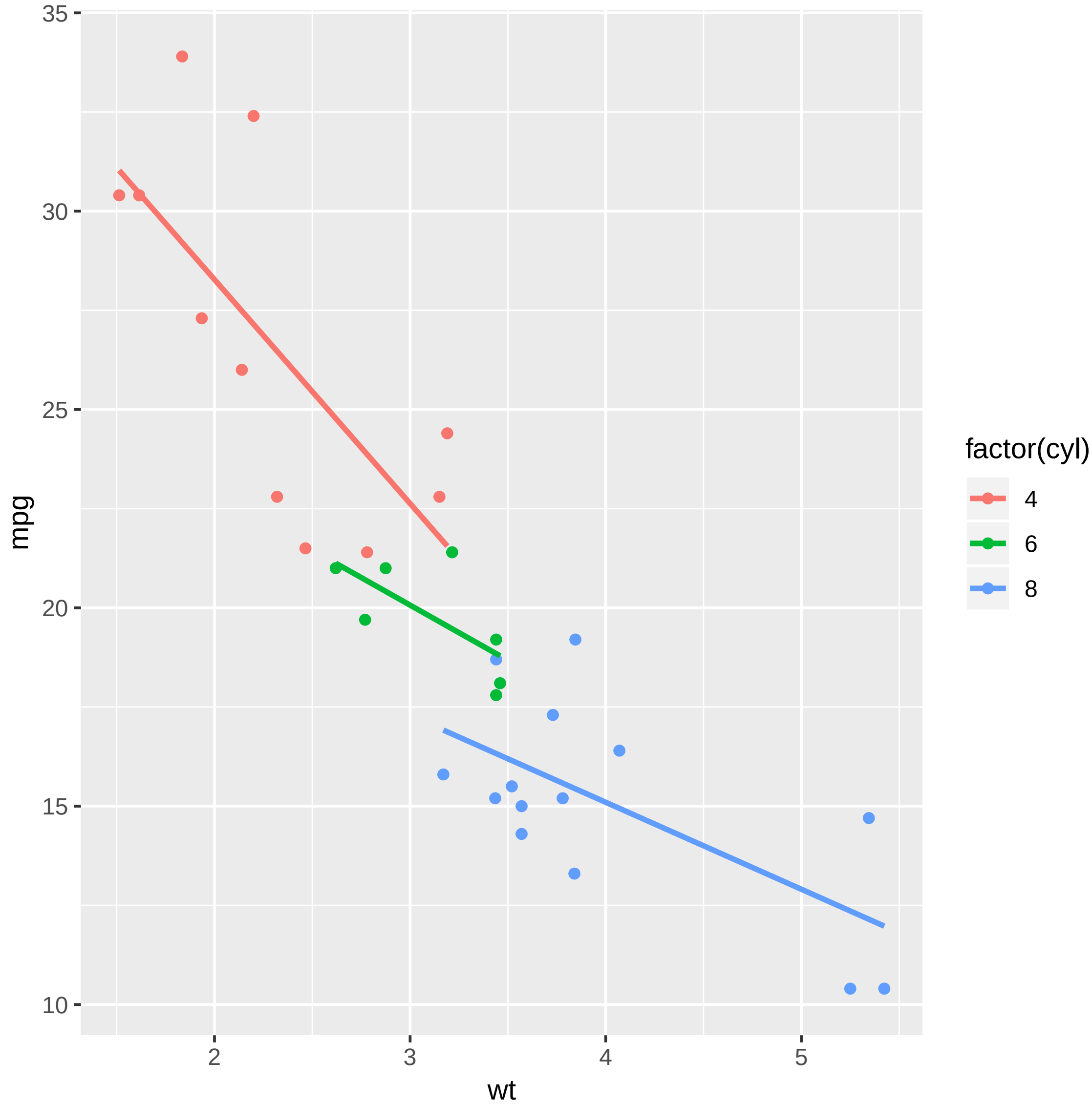

```r
# The previous plot, without points
ggplot(mtcars, aes(x = wt, y = mpg)) +
  stat_smooth(method = "lm", se=FALSE)
```



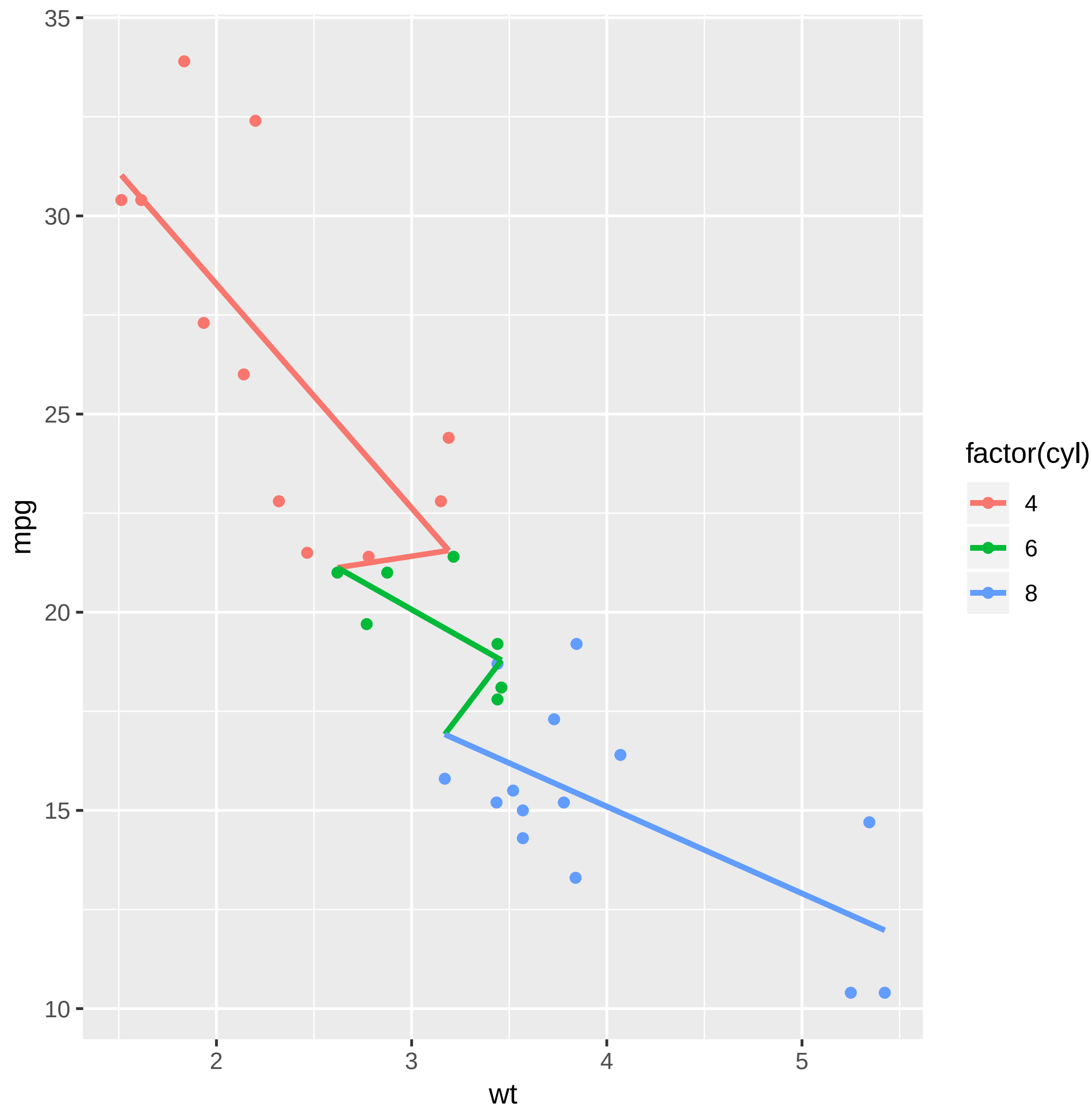
You can use either `stat_smooth()` or `geom_smooth()` to apply a linear model. Remember to always think about how the examples and concepts we discuss throughout the data viz courses can be applied to your own datasets!

Grouping Variables

```
# 1 - Define cyl as a factor variable
ggplot(mtcars, aes(x = wt, y = mpg, col = factor(cyl))) +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE)
```



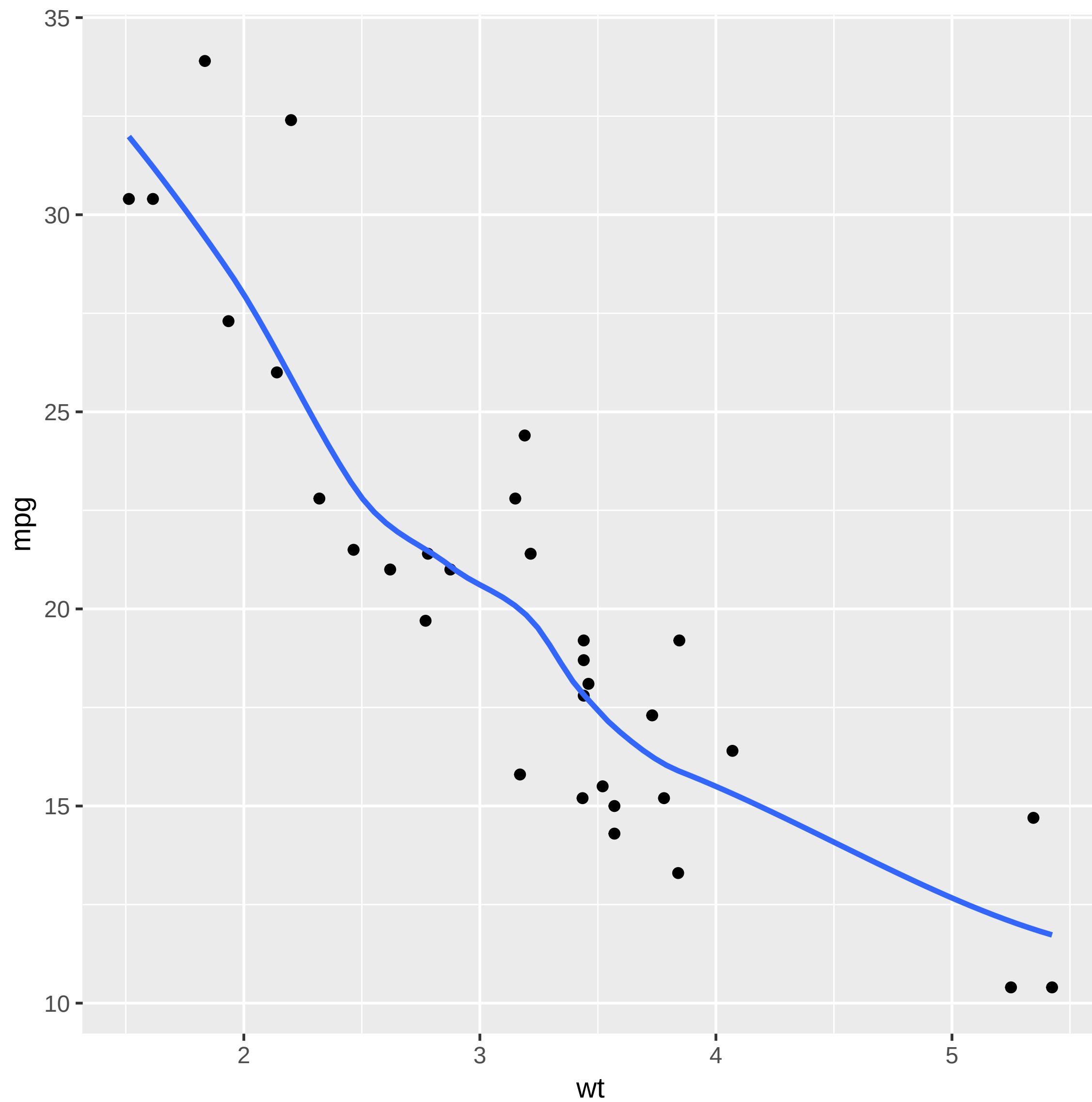

```r
# 2 - Plot 1, plus another stat_smooth() containing a nested aes()
ggplot(mtcars, aes(x = wt, y = mpg, col = factor(cyl))) +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE) +
  stat_smooth(method = "lm", se = FALSE, group = 1)
```



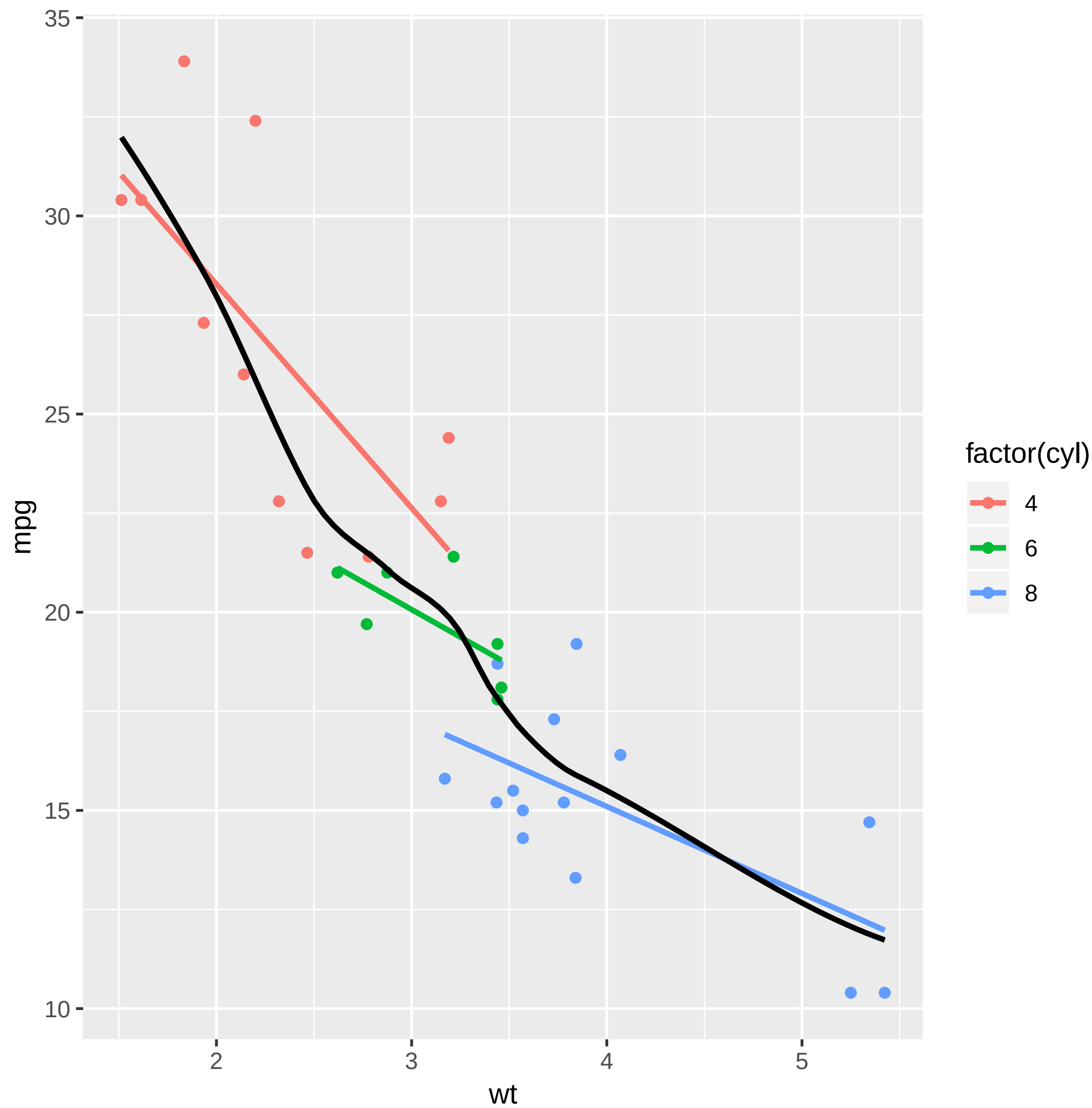
Good job! Notice that we can use multiple aesthetic layers, just like we can use multiple geom layers. Each aesthetic layer can be mapped onto a specific geom.

Modifying stat_smooth (1)

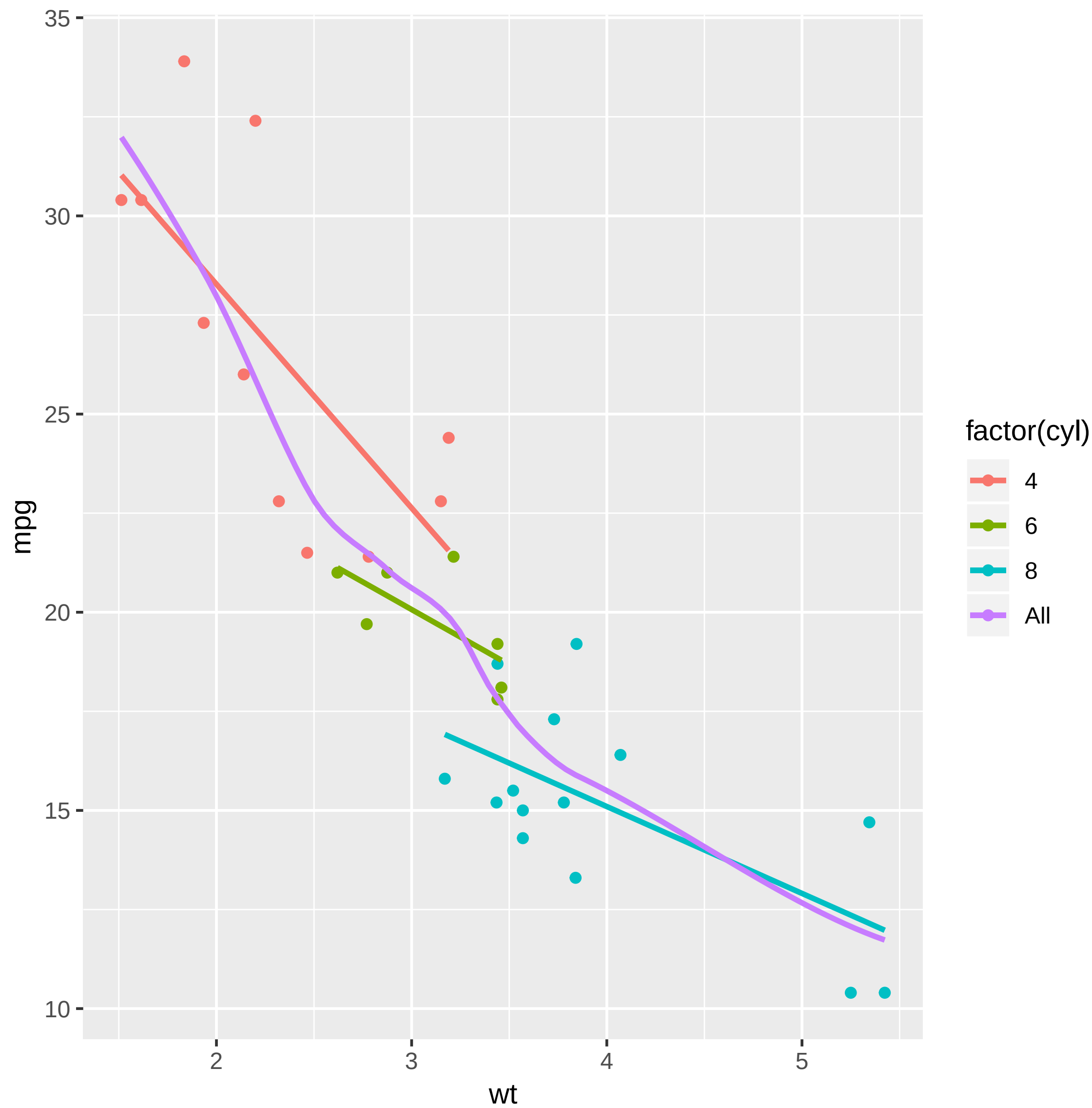
```
par(mfrow = c(2, 2))

# Plot 1: change the LOESS span
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  # Add span below
  geom_smooth(se = FALSE, span = 0.7)
#> `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



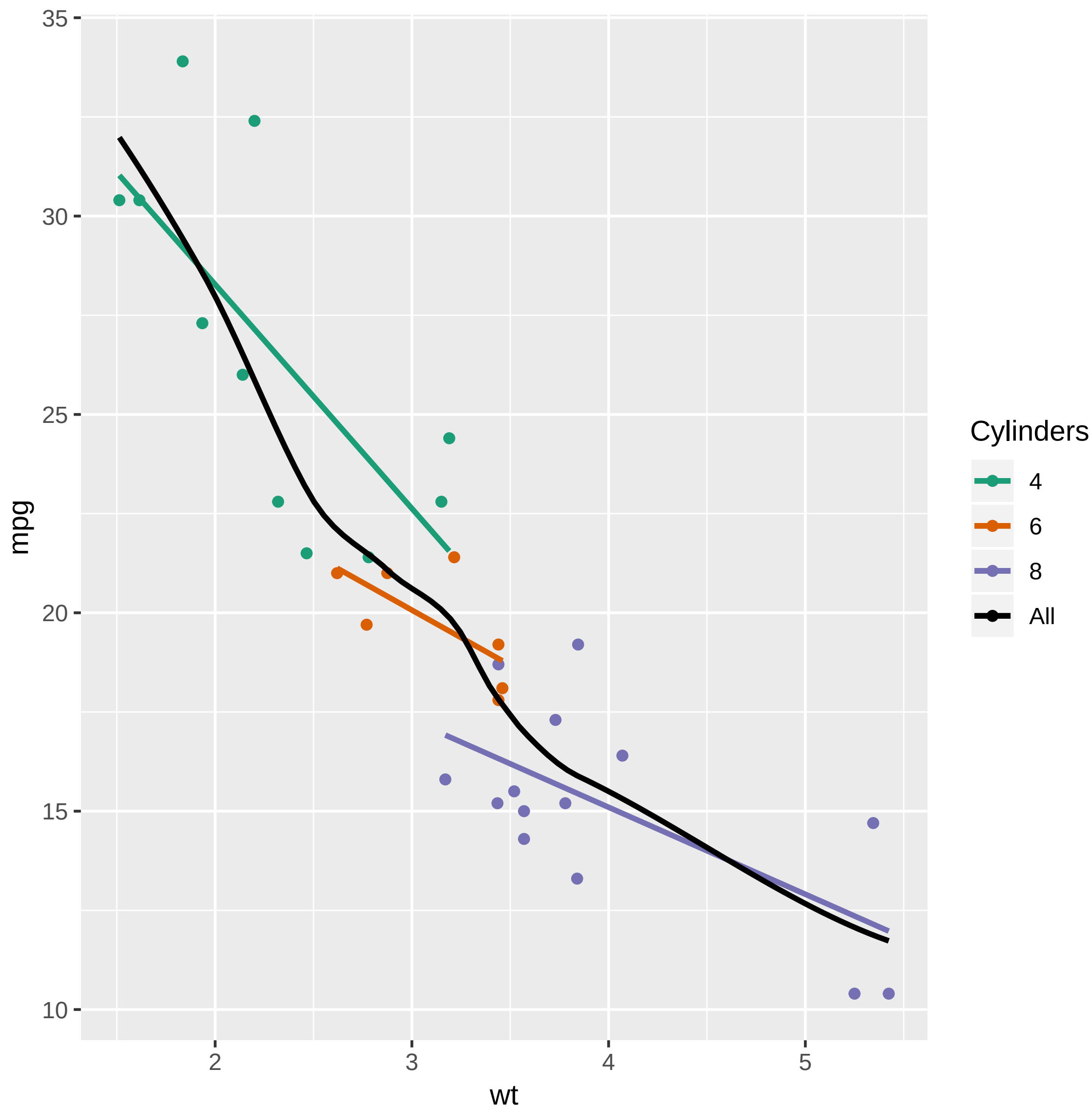

```r
# Plot 2: Set the second stat_smooth() to use LOESS with a span of 0.7
ggplot(mtcars, aes(x = wt, y = mpg, col = factor(cyl))) +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE) +
  # Change method and add span below
  stat_smooth(method = "loess", aes(group = 1),
              se = FALSE, col = "black", span = 0.7)
```




```r
# Plot 3: Set col to "All", inside the aes layer of stat_smooth()
ggplot(mtcars, aes(x = wt, y = mpg, col = factor(cyl))) +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE) +
  stat_smooth(method = "loess",
              # Add col inside aes()
              aes(group = 1, col = "All"),
              # Remove the col argument below
              se = FALSE, span = 0.7)
```




```r
# Plot 4: Add scale_color_manual to change the colors
myColors <- c(brewer.pal(3, "Dark2"), "black")
ggplot(mtcars, aes(x = wt, y = mpg, col = factor(cyl))) +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE, span = 0.7) +
  stat_smooth(method = "loess",
              aes(group = 1, col="All"),
              se = FALSE, span = 0.7) +
  # Add correct arguments to scale_color_manual
  scale_color_manual("Cylinders", values = myColors)
```



Modifying stat_smooth (2)

Calculating Statics

Quantiles

The previous example used the Vocab dataset and applied linear models describing vocabulary by education for different years. Here we'll continue with that example by using stat_quantile() to apply a quantile regression (method rq).

By default, the 1st, 2nd (i.e. median), and 3rd quartiles are modeled as a response to the predictor variable, in this case education. Specific quantiles can be specified with the quantiles argument.

If you want to specify many quantile and color according to year, then things get too busy. We'll explore ways of dealing with this in the next chapter.

Quick quantiles! Quantile regression is a great tool for getting a more detailed overview of a large dataset.

Sum

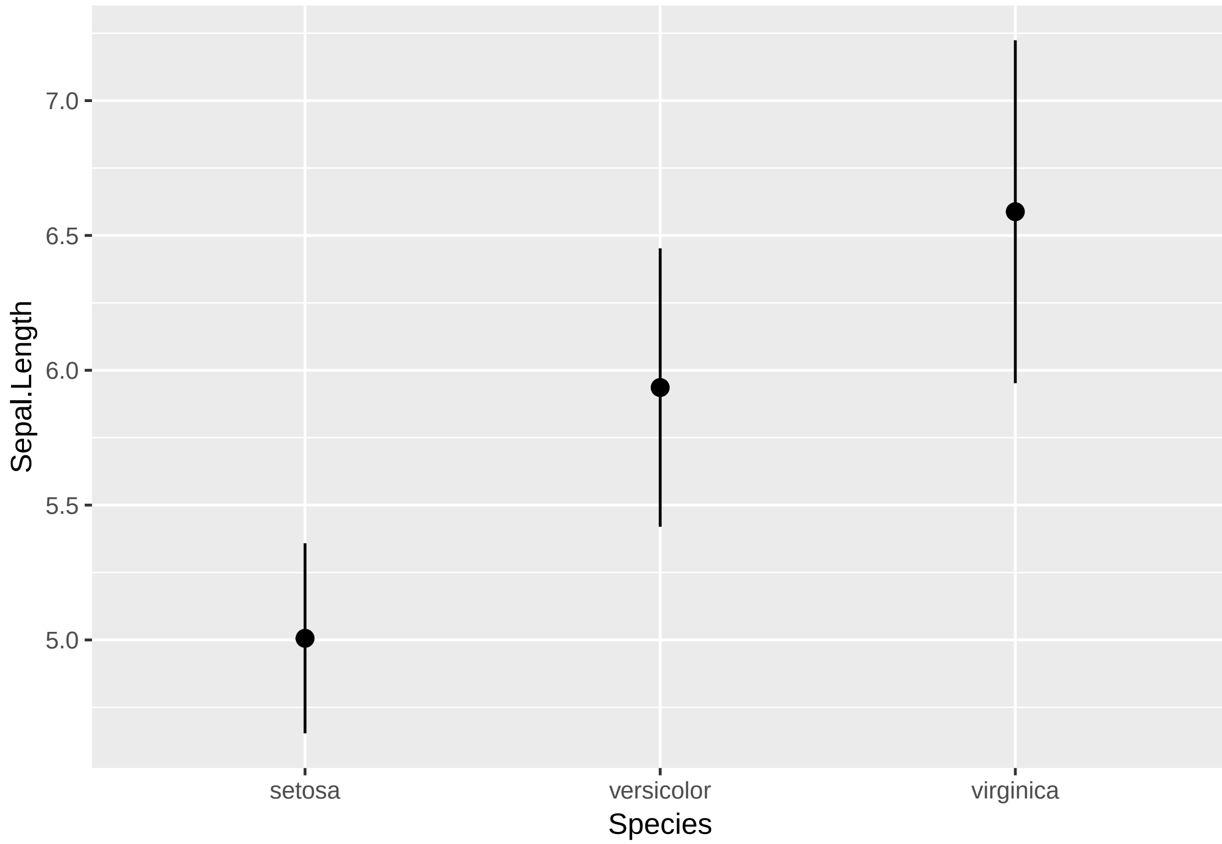
Good job! Remember, typically we'd draw our models on top of the dots, but in this case we didn't so that we could just keep recycling the p object.

Mean & Standard Deviation

```r
set.seed(123)

xx <- rnorm(100)

mean(xx)
#> [1] 0.09040591

mean(xx) + (sd(xx) * c(-1, 1))
#> [1] -0.822410  1.003222

library(Hmisc)
#> Warning: package 'Hmisc' was built under R version 3.5.3
smean.sdl(xx, mult = 1)
#>        Mean       Lower       Upper
#>  0.09040591 -0.82240997  1.00322179

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

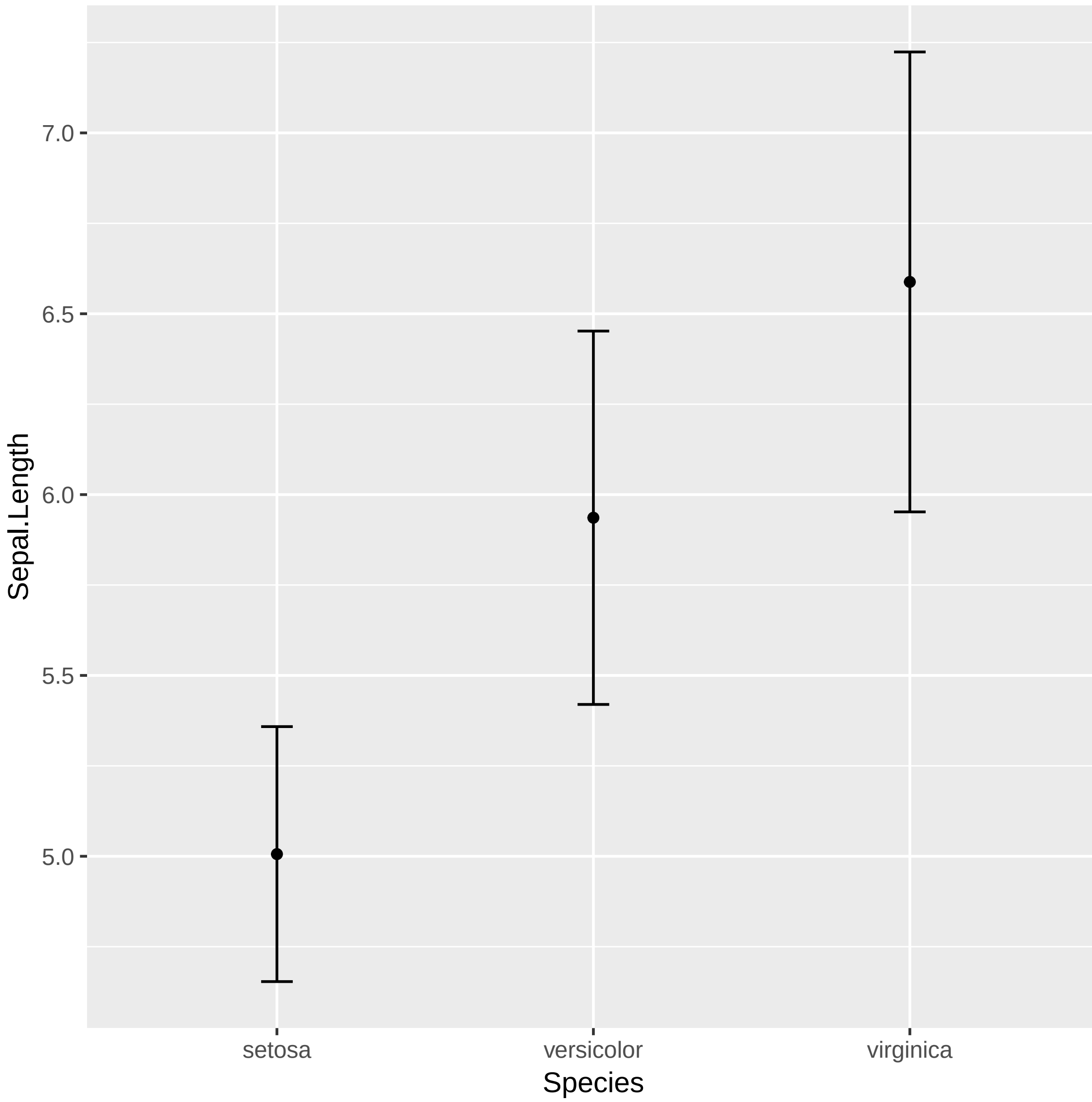
y	ymin	ymax
0.0904059	-0.82241	1.003222

```r
# we can use this in ggplot2 by calling the fun.data function
p <- ggplot(iris, aes(x = Species, y = Sepal.Length))

p + stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1))
```

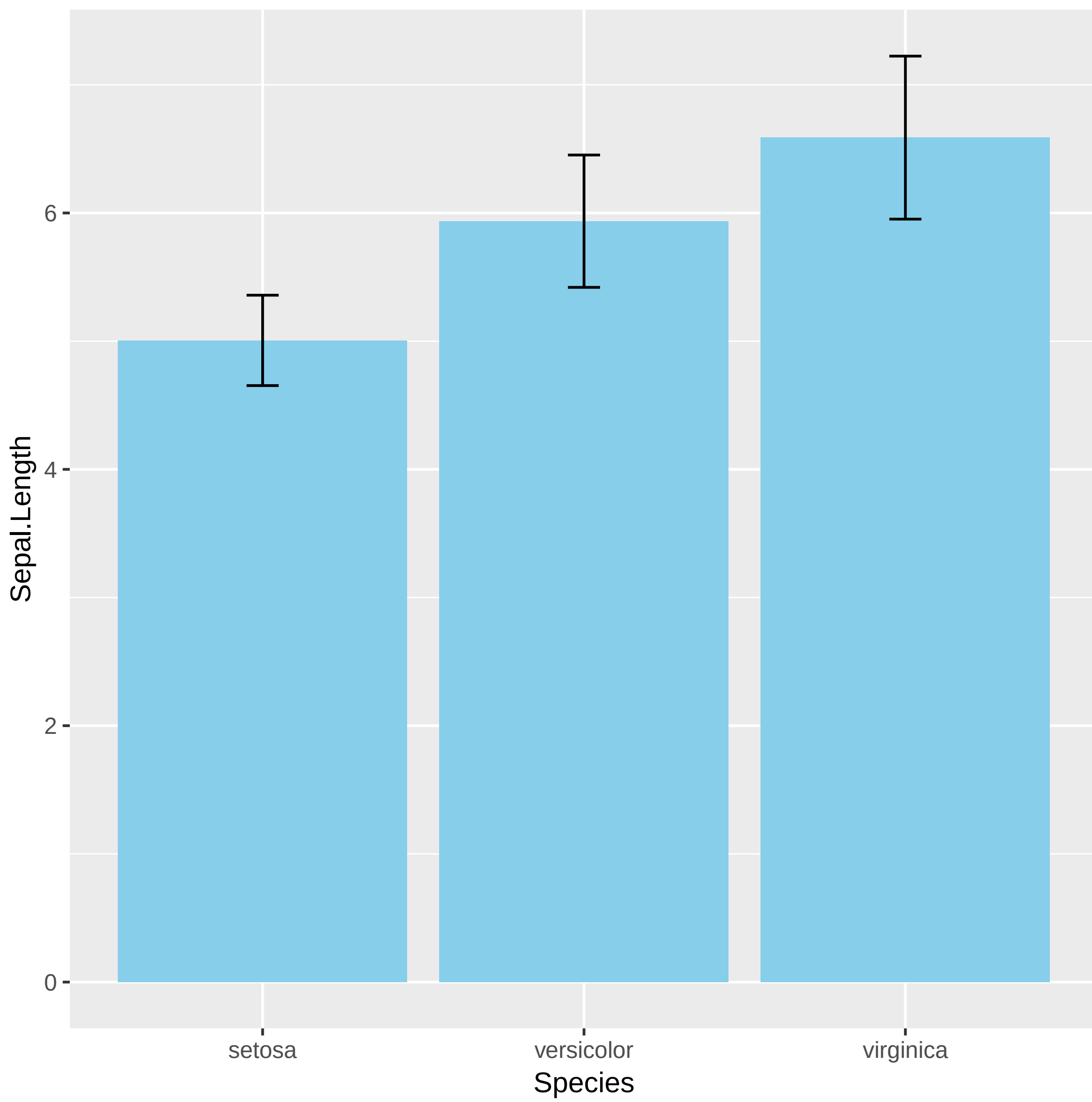


By default the `stat_summary()` function uses the `geom_pointrange()`, which requires `y`, `ymin`, and `ymax` - the exact variables returned by `mean.sdl`. So everything works very well together. If we wanted a more typical errorbar style plot, we can independently plot the mean and the use the `point` argument for the `geom` and again call `mean_sdl`, but this time using the errorbar geom.

```
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)
```




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

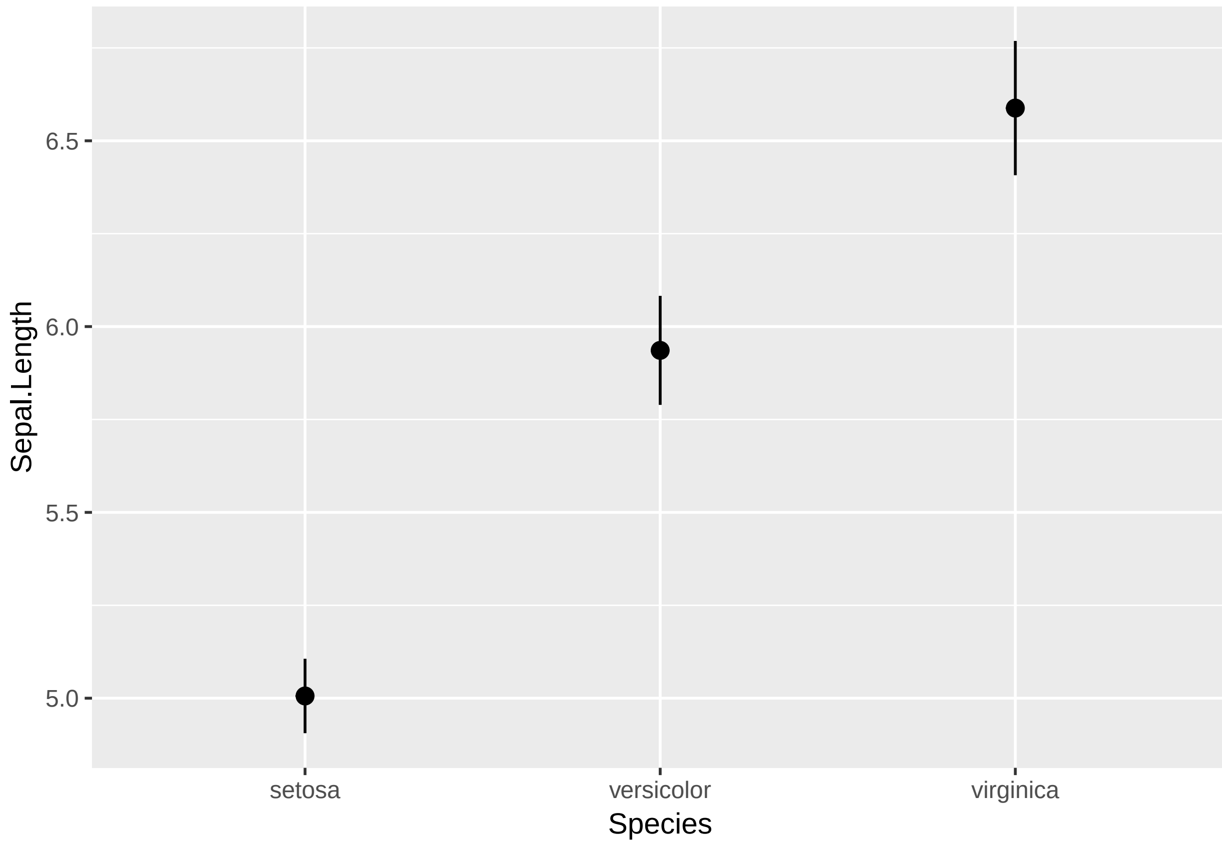


Confidence Interval

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

mean(xx) + (ERR* c(-1, 1))
#> [1] -0.09071657  0.27152838

smean.cl.normal(xx)
#>        Mean       Lower       Upper
#>  0.09040591 -0.09071657  0.27152838

mean_cl_normal(xx)
```

y	ymin	ymax
0.0904059	-0.0907166	0.2715284

```r
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  stat_summary(fun.data = mean_cl_normal, width = 0.1)
#> Warning: Ignoring unknown parameters: width
```



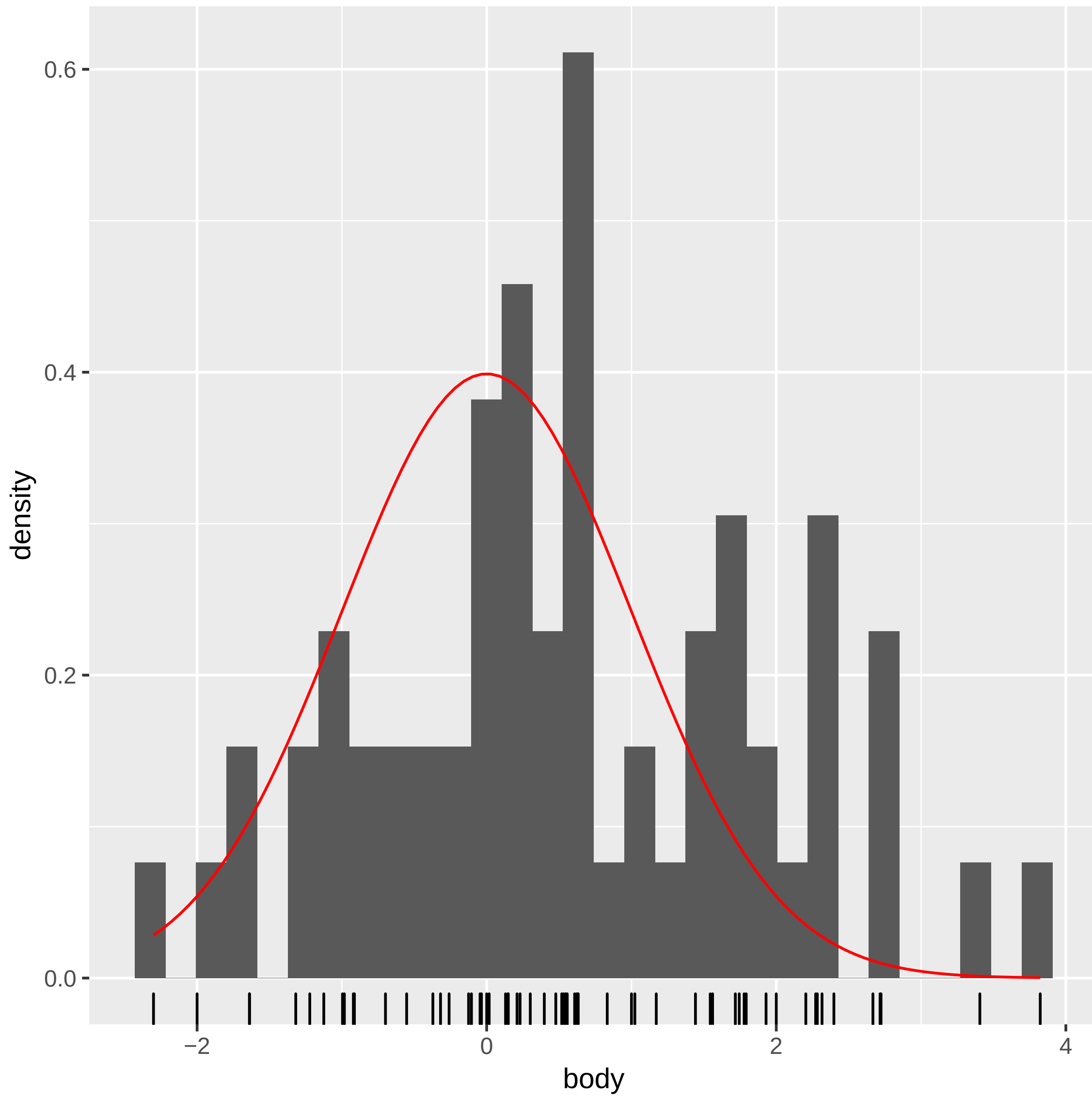
You can use any function in ggplot as long as the output has the expected format. Other useful `stat_` layer functions are `stat_summary`, `stat_function`, and `stat_qq`.

stat	description
stat_summary()	summarize y values at distinct x values
stat_function()	compute y values from a function of x values
stat_qq()	perform calculations for a quantile-quantile plot

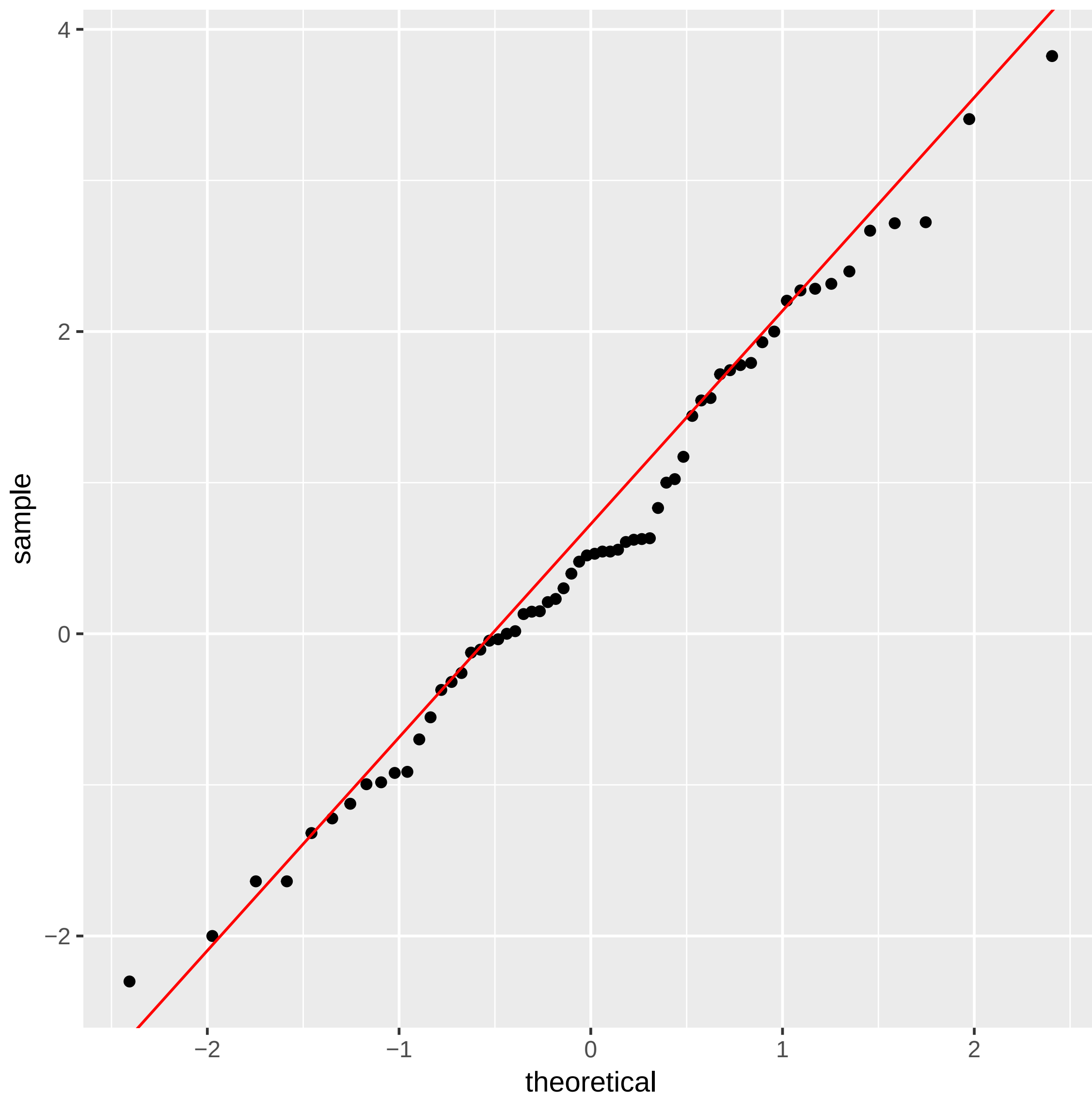
Normal distribution

```r
library(MASS)
#> Warning: package 'MASS' was built under R version 3.5.3
#>
#> Attaching package: 'MASS'
#> The following object is masked from 'package:dplyr':
#>
#>     select
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, colour = "red",
                arg = list(mean = mean(mam.new$body),
                           sd = sd(mam.new$body)))
#> Warning: Ignoring unknown parameters: arg
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



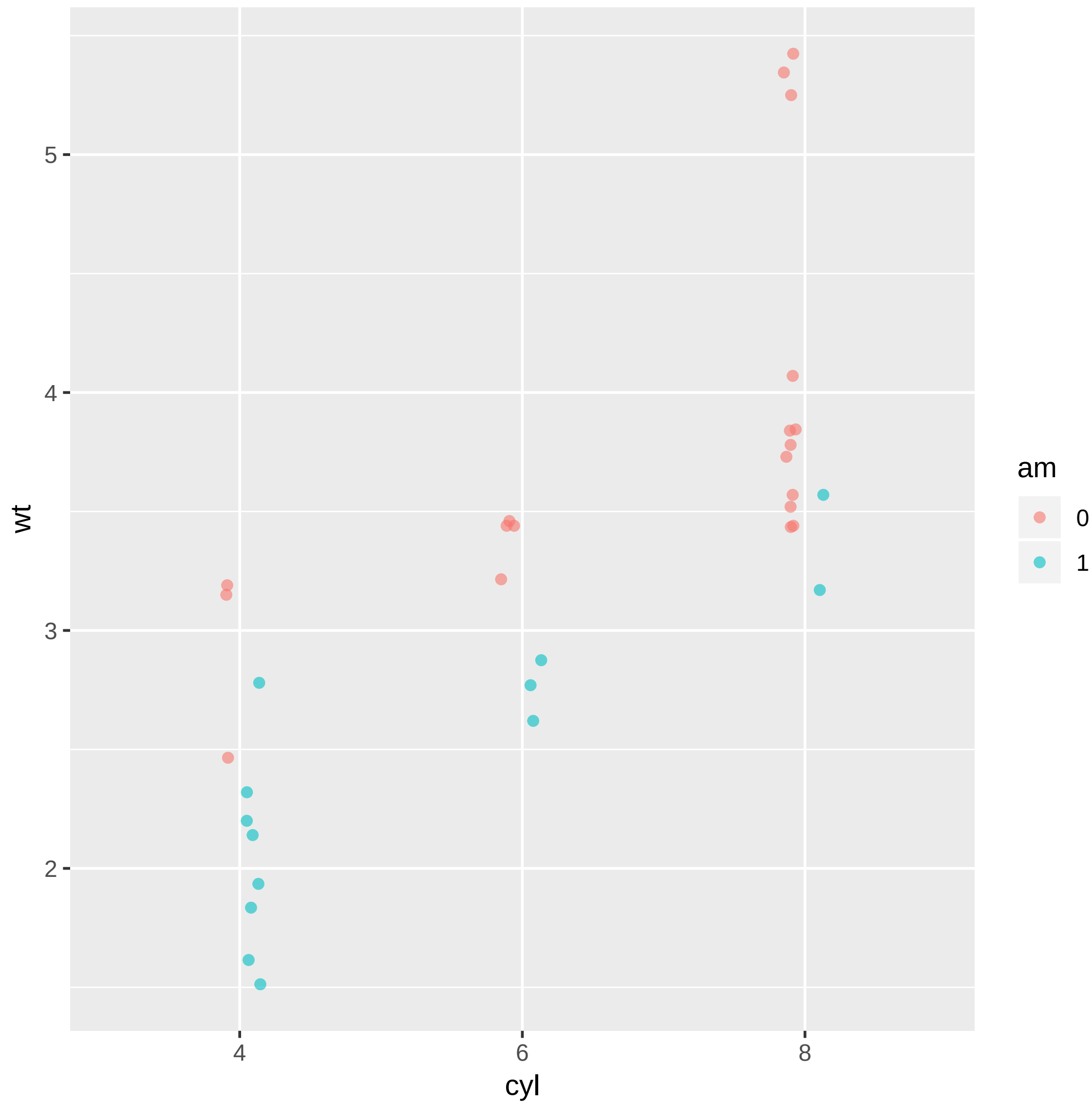

```
# QQplot

mam.new$slope <- diff(quantile(mam.new$body, c(0.25, 0.75))) /
diff(qnorm(c(0.25, 0.75)))
mam.new$int <- quantile(mam.new$body, 0.25) -
mam.new$slope * qnorm(0.25)
ggplot(mam.new, aes(sample = body)) +
  stat_qq() +
  geom_abline(aes(slope = slope, intercept = int), col = "red")
```



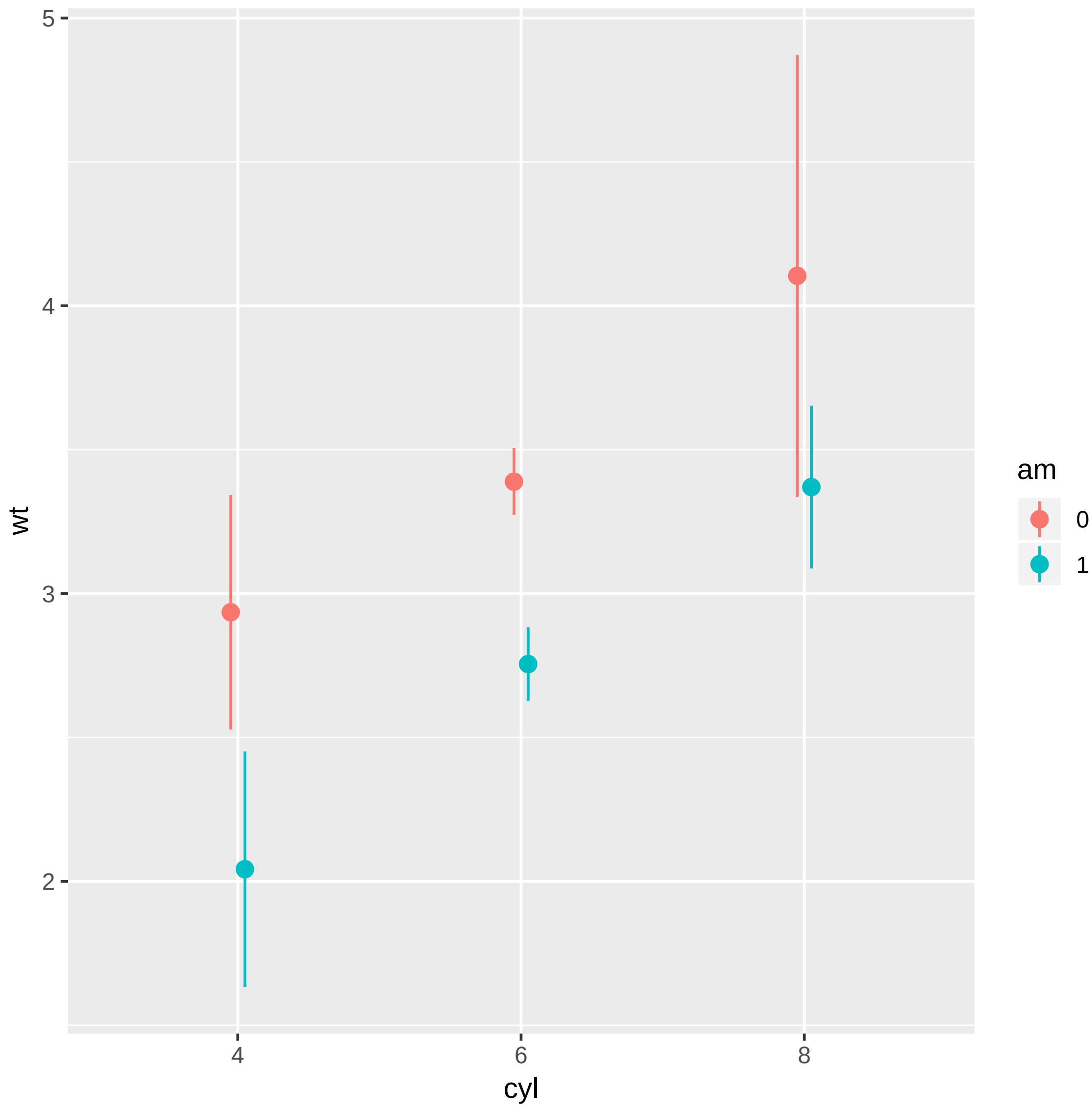
Stat_Summary in Action

```
# Display structure of mtcars
str(mtcars)
#> 'data.frame':	32 obs. of  11 variables:
#>  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
#>  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
#>  $ disp: num  160 160 108 258 360 ...
#>  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
#>  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
#>  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
#>  $ qsec: num  16.5 17 18.6 19.4 17 ...
#>  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
#>  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
#>  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
#>  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

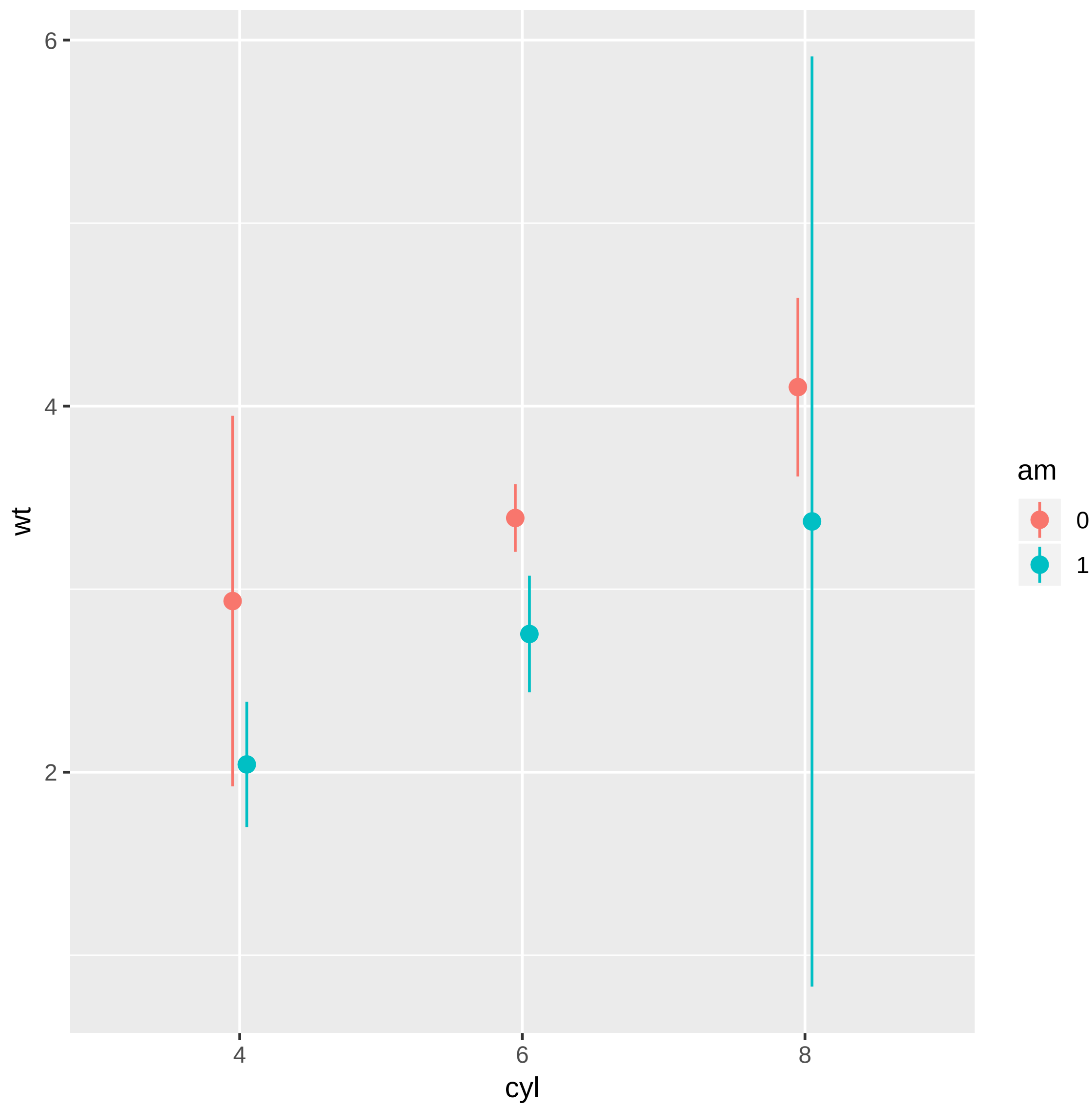
```
# Convert cyl and am to factors
mtcars$cyl <- factor(mtcars$cyl)
mtcars$am <- factor(mtcars$am)

# Define positions
posn.d <- position_dodge(width = 0.1)
posn.jd <- position_jitterdodge(jitter.width = 0.1, dodge.width = 0.2)
posn.j <- position_jitter(width = 0.2)

# Base layers
wt.cyl.am <- ggplot(mtcars, aes(x = cyl, y = wt, col = am, fill = am, group = am))

# Plot 1: Jittered, dodged scatter plot with transparent points
wt.cyl.am +
  geom_point(position = posn.jd, alpha = 0.6)
```



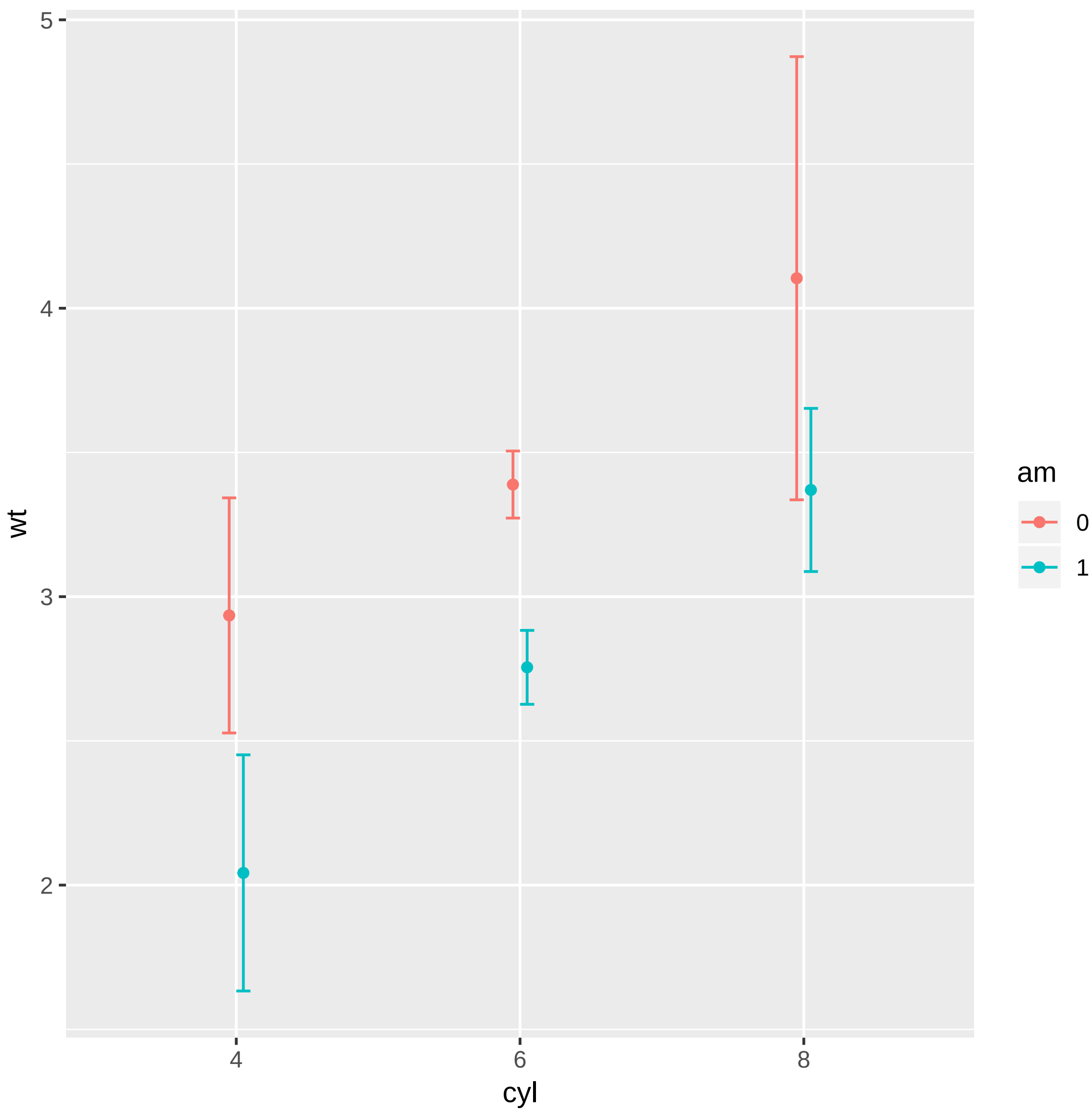
```
# Plot 2: Mean and SD - the easy way
wt.cyl.am +
  stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1), position = posn.d)
```




```
# Plot 3: Mean and 95% CI - the easy way
wt.cyl.am +
  stat_summary(fun.data = mean_cl_normal, position = posn.d)
```




```
# Plot 4: Mean and SD - with T-tipped error bars - fill in ___
wt.cyl.am +
  stat_summary(geom = "point", fun.y = mean,
               position = posn.d) +
  stat_summary(geom = "errorbar", fun.data = mean_sdl,
               position = posn.d, fun.args = list(mult = 1), width = 0.1)
```



Perfect positioning! Although you can set position using e.g. position = "dodge", defining objects promotes consistency between layers.

Good job! Remember that you can always specify your own function to the fun.data argument as long as the variable names match the aesthetics that you will need for the geom layer.

Custom Functions (1)

```
xx <- seq(1, 100, 1)

# Function to save range for use in ggplot
gg_range <- function(x) {
  # Change x below to return the instructed values
  data.frame(ymin = min(x), # Min
             ymax = max(x)) # Max
}
```

```r
gg_range(xx)
```

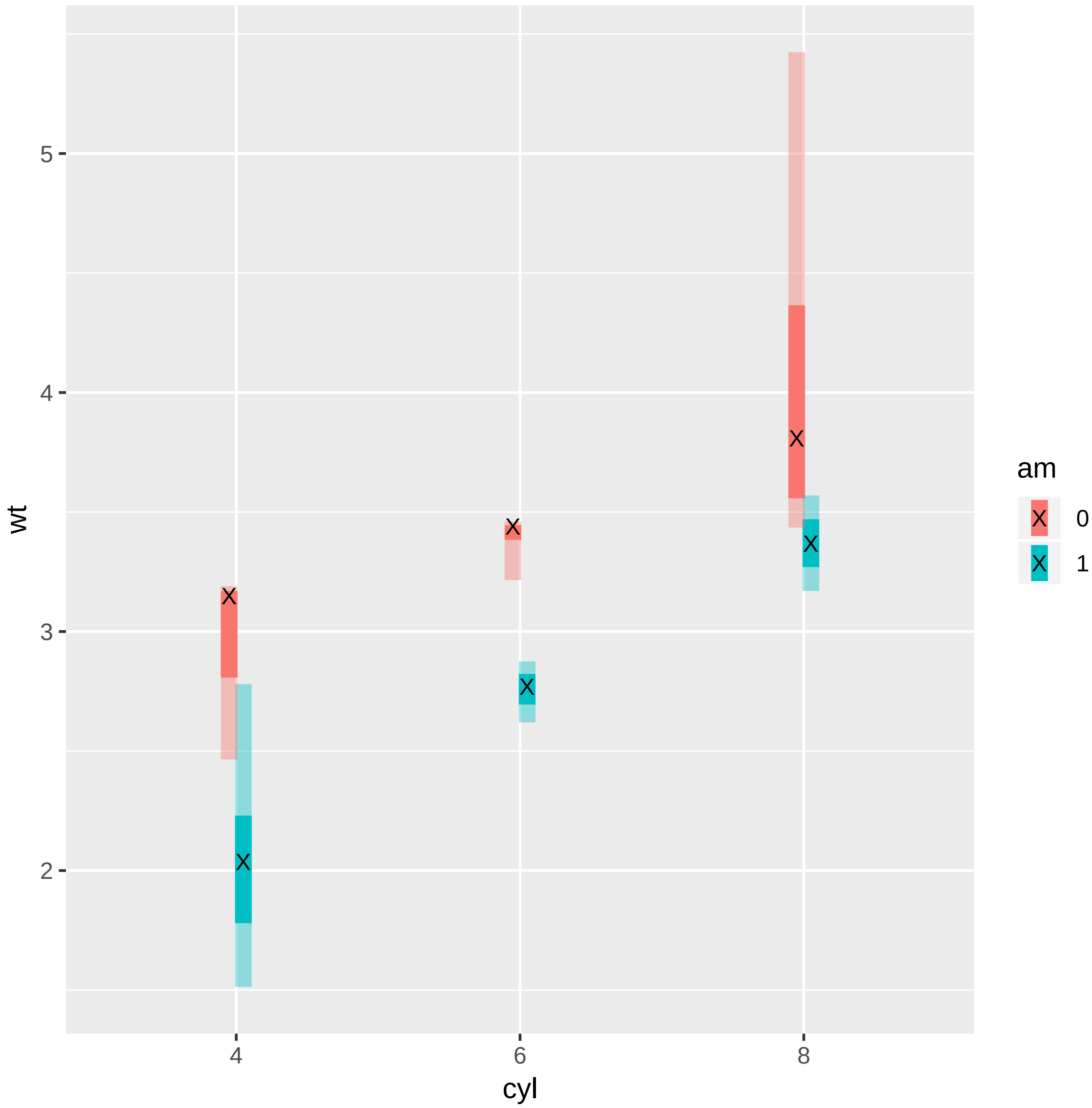
ymin	ymax
1	100

```r
# Function to Custom function
med_IQR <- function(x) {
  # Change x below to return the instructed values
  data.frame(y = median(x), # Median
             ymin = quantile(x)[2], # 1st quartile
             ymax = quantile(x)[4])  # 3rd quartile
}

med_IQR(xx)
```

	y	ymin	ymax
25%	50.5	25.75	75.25

Custom Functions (2)

```r
wt.cyl.am +
  stat_summary(geom = "linerange", fun.data = med_IQR,
               position = posn.d, size = 3) +
  stat_summary(geom = "linerange", fun.data = gg_range,
               position = posn.d, size = 3,
               alpha = 0.4) +
  stat_summary(geom = "point", fun.y = median,
               position = posn.d, size = 3,
               col = "black", shape = "X")
```


wt
5
4
3
2
4
6
8
cyl
am
0
1