ggplot-lecture

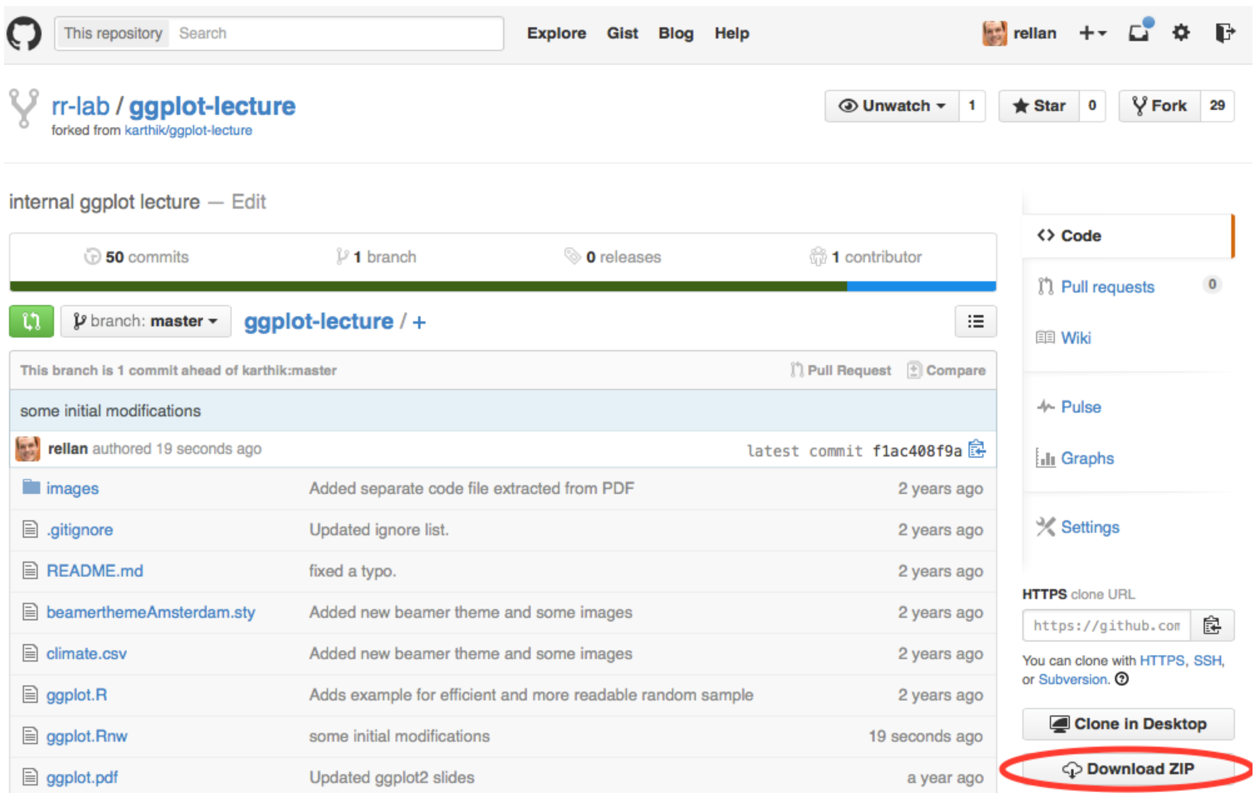
Rubén Rellán Álvarez, Karthik Ram

May 30, 2015

Data Visualization with R and ggplot

First go here:

http://github.com/rrlab/ggplot-lecture}{github.com/rrlab/ggplot-lecture



This branch is 1 commit ahead of karthik:master

some initial modifications

rellan authored 19 seconds ago — latest commit f1ac408f9a

images	Added separate code file extracted from PDF	2 years ago
.gitignore	Updated ignore list.	2 years ago
README.md	fixed a typo.	2 years ago
beamerthemeAmsterdam.sty	Added new beamer theme and some images	2 years ago
climate.csv	Added new beamer theme and some images	2 years ago
ggplot.R	Adds example for efficient and more readable random sample	2 years ago
ggplot.Rnw	some initial modifications	19 seconds ago
ggplot.pdf	Updated ggplot2 slides	a year ago

Then install the following packages:

```r
install.packages("ggplot2", repos="http://cran.rstudio.com/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/g5/s26yfnqs4dj12pn6w88g__cc0000gn/T//RtmpAsixXD/downloaded_packages
```

```r
install.packages("dplyr", repos="http://cran.rstudio.com/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/g5/s26yfnqs4dj12pn6w88g__cc0000gn/T//RtmpAsixXD/downloaded_packages
```

```r
install.packages("tidyr", repos="http://cran.rstudio.com/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/g5/s26yfnqs4dj12pn6w88g__cc0000gn/T//RtmpAsixXD/downloaded_packages
```

```r
install.packages("magrittr", repos="http://cran.rstudio.com/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/g5/s26yfnqs4dj12pn6w88g__cc0000gn/T//RtmpAsixXD/downloaded_packages
```

```r
install.packages("RColorBrewer", repos="http://cran.rstudio.com/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/g5/s26yfnqs4dj12pn6w88g__cc0000gn/T//RtmpAsixXD/downloaded_packages
```

And then load them:

```r
library("ggplot2")
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library("tidyr")
library("magrittr")
```

```
##
## Attaching package: 'magrittr'
##
## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
library("RColorBrewer")
```

Base graphics

- Ugly, laborious, and verbose
- There are better ways to describe statistical visualizations

ggplot2

- Follows a grammar, just like any language.
- It defines basic components that make up a sentence. In this case, the grammar defines components in a plot.
- Grammar of graphics originally coined by Lee Wilkinson ggplot : **g**ramar of **g**raphics plot
- Implemented in R by Hadley Wickham

Why ggplot2?

- Supports a continuum of expertise.
- Get started right away but with practice you can effortless build complex, publication quality figures.

Basics

Some terminology

- **ggplot** The main function where you specify the dataset and variables to plot
- **geoms** geometric objects
- geom_point(), geom_bar(), geom_density(), geom_line(), geom_area()
- **aes** aesthetics
- shape, transparency (alpha), color, fill, linetype.
- **scales** Define how your data will be plotted -continuous, discrete, log

Assembling your first ggplot

The iris dataset

```r
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
tail(iris)
```

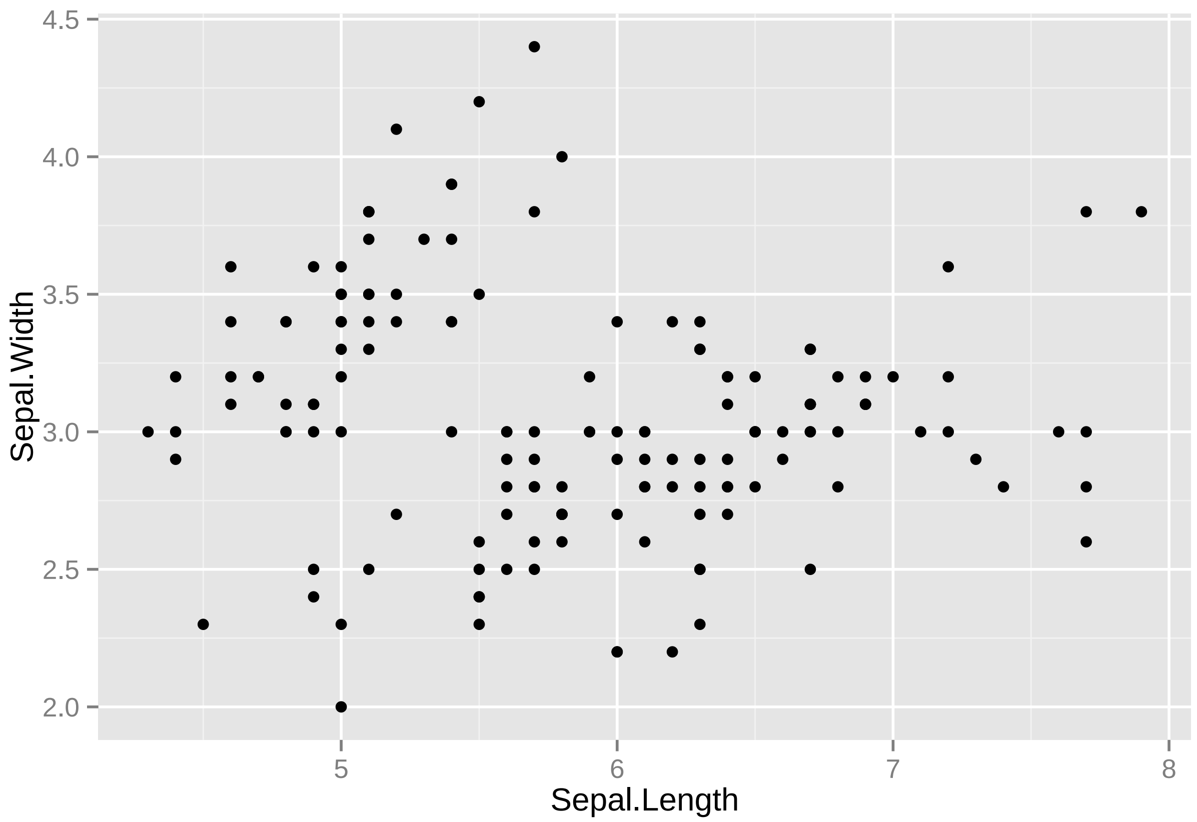
```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

```
glimpse(iris)
```

```
## Observations: 150
## Variables:
## $ Sepal.Length (dbl) 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9,...
## $ Sepal.Width  (dbl) 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1,...
## $ Petal.Length (dbl) 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5,...
## $ Petal.Width  (dbl) 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1,...
## $ Species      (fctr) setosa, setosa, setosa, setosa, setosa, setosa, ...
```

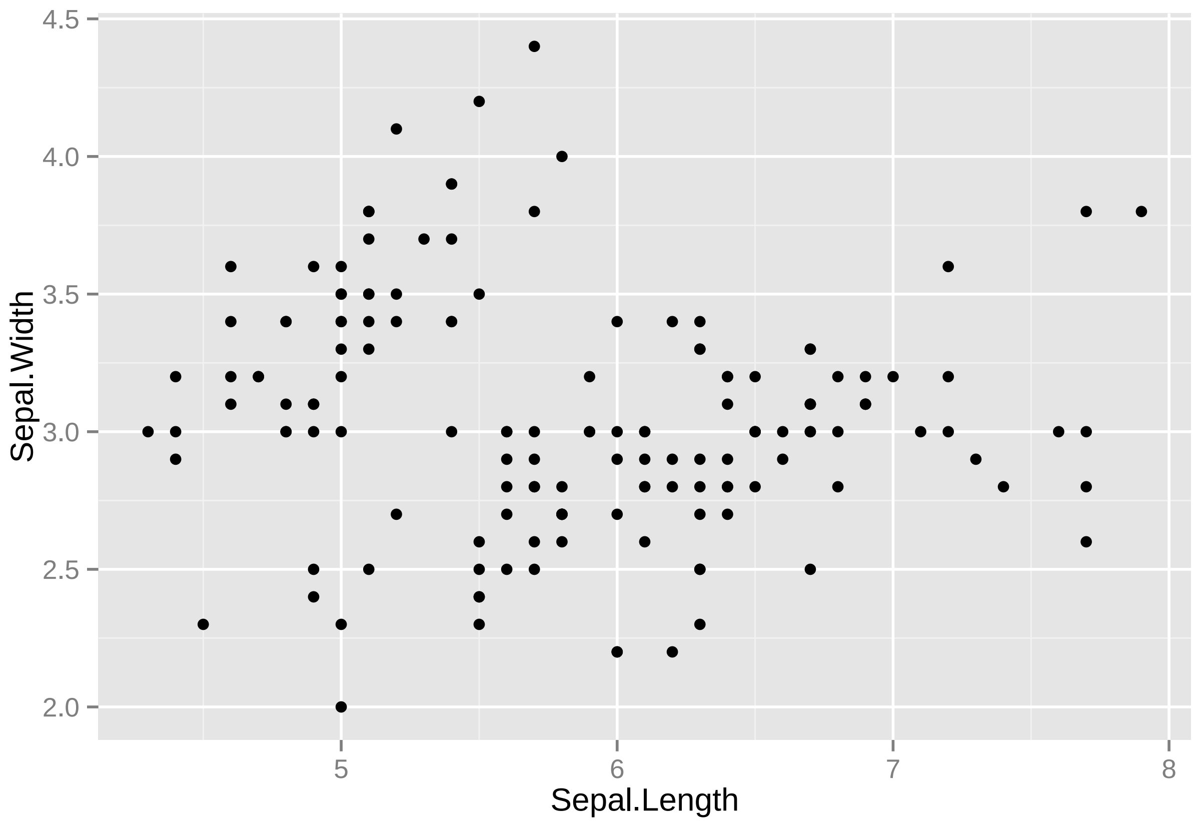
Let's try an example

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point()
```

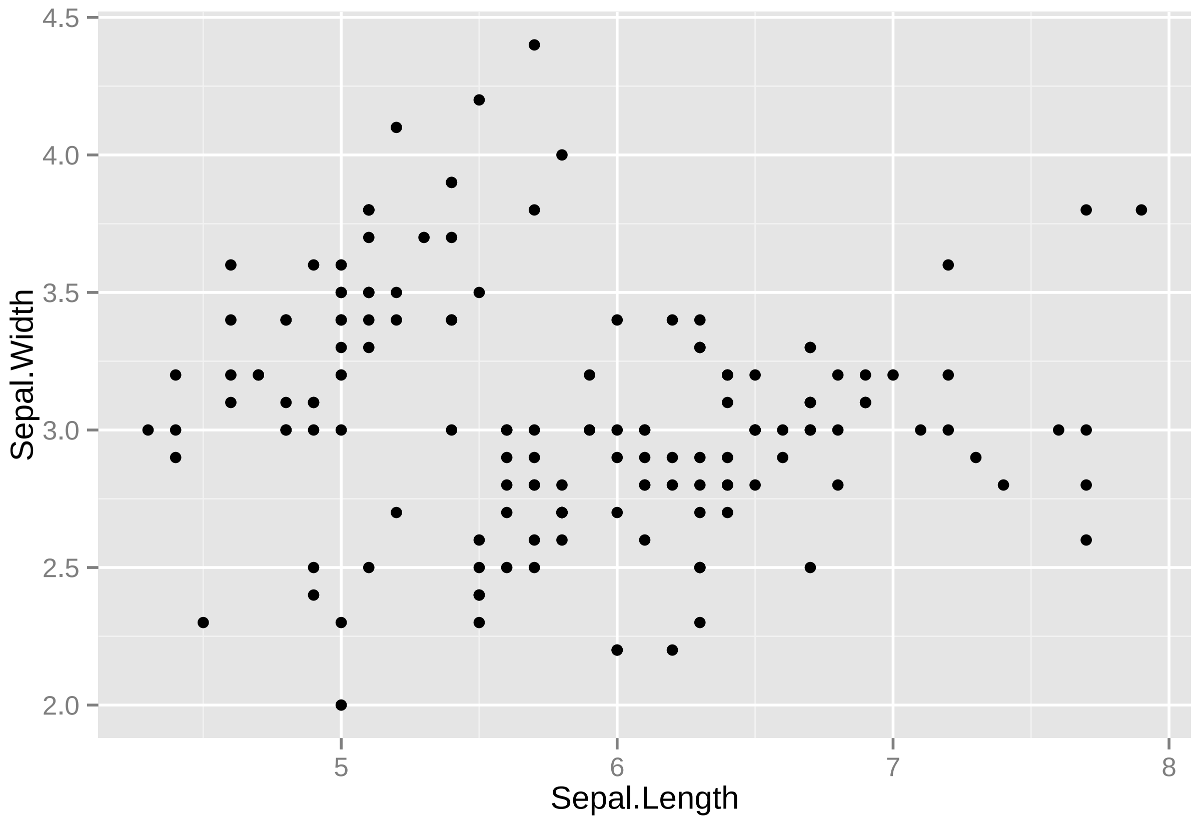


Basic structure

```r
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point()
```



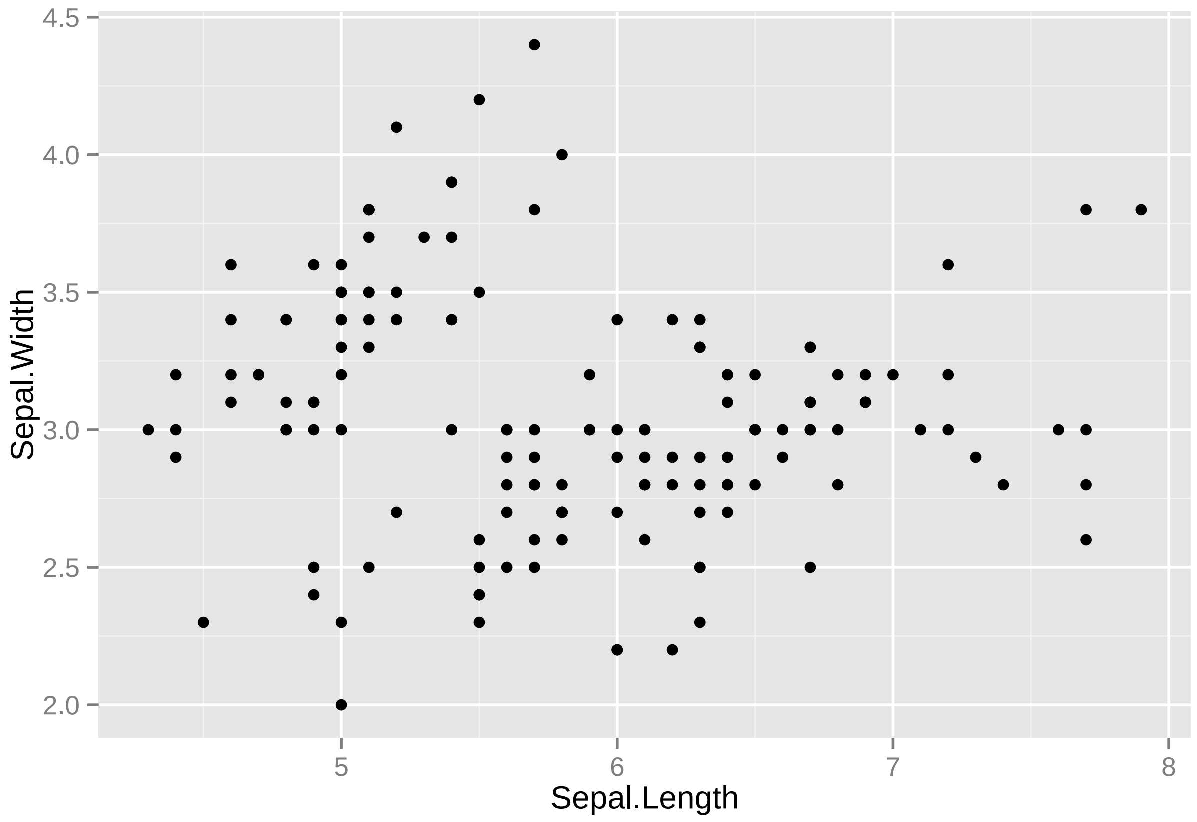
```r
myplot <- ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width))
myplot + geom_point()
```



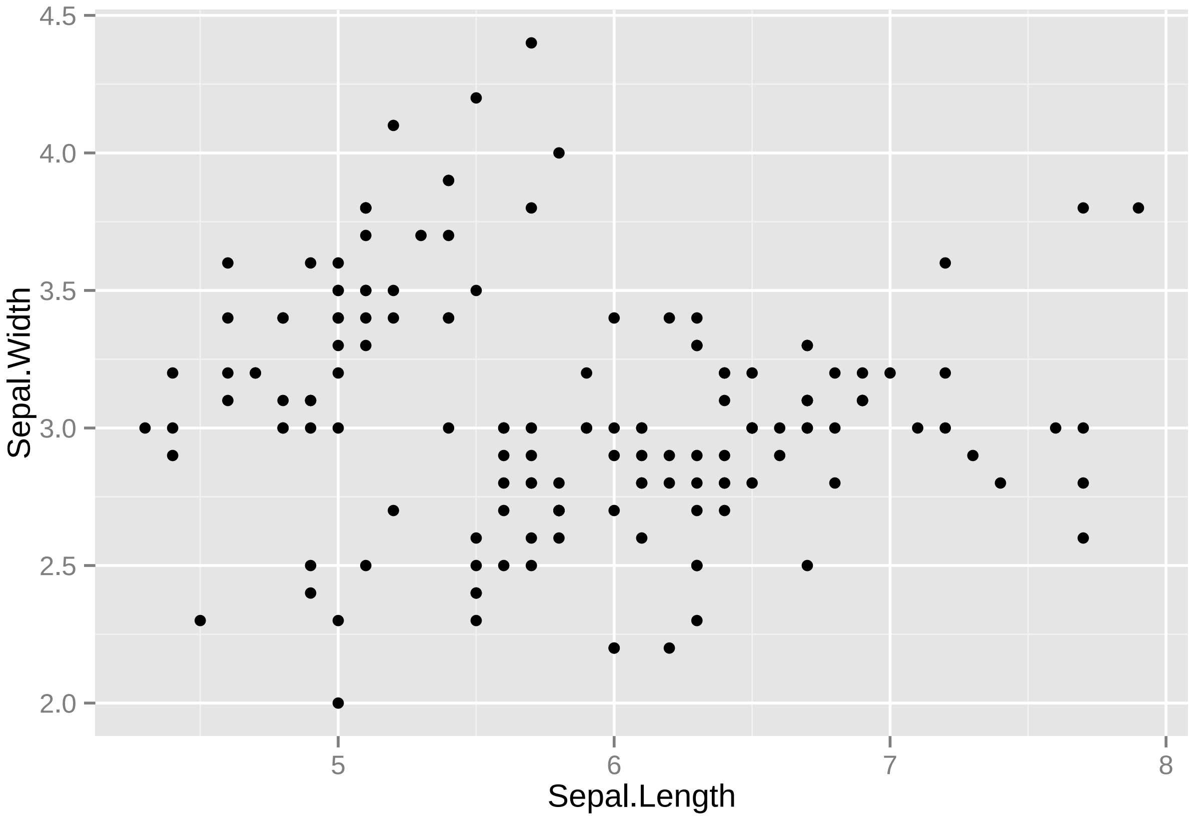
- Specify the data and variables inside the **ggplot** function.
- Anything else that goes in here becomes a global setting.
- Then add layers of geometric objects, statistical models, and panels.

Alternative modes to say the same thing

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



```r
iris %>%
ggplot(aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point()
```



Quick notes

- Never use *qplot* - short for quick plot.
- You'll end up unlearning and relearning a good bit.
- Check the name of your variables Sepal.length is not sepal.length is not SepalLength
- Be consistent with your variable naming. I like sepal_length
- Have you closed all your parenthesis? This:

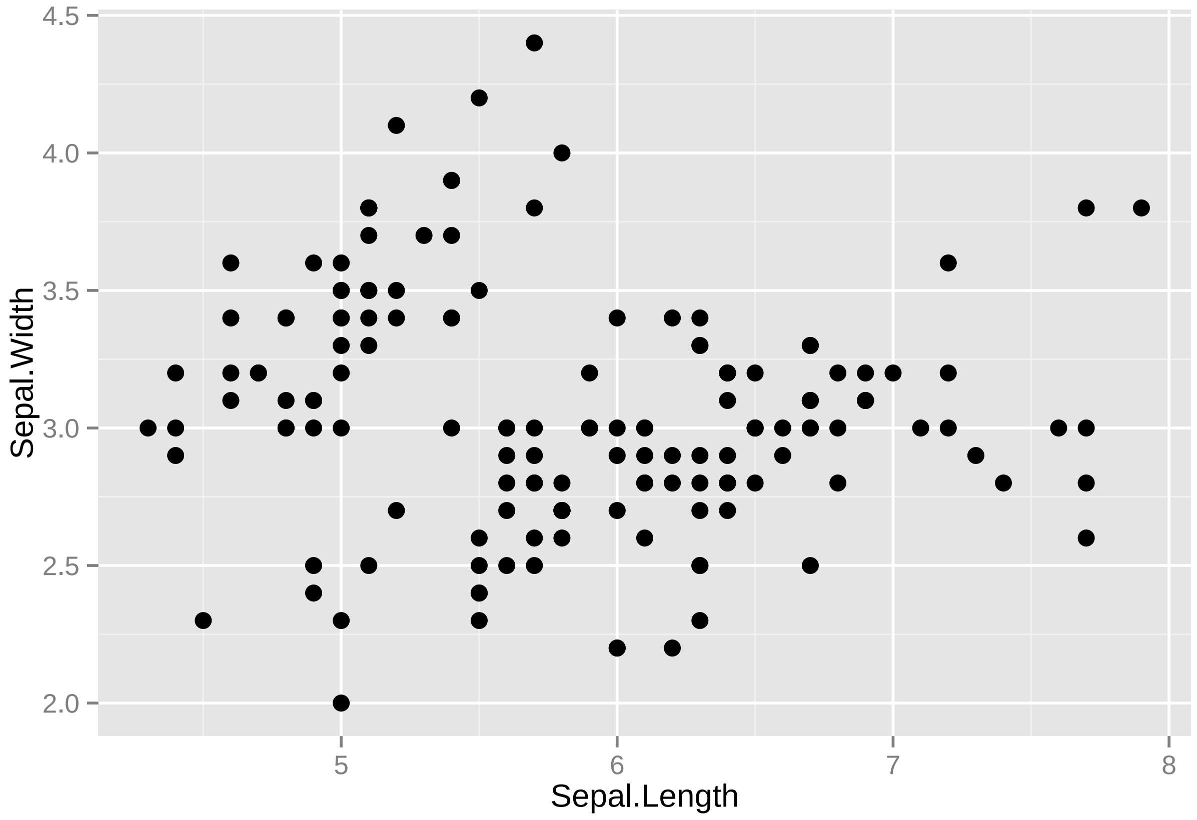
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width) + geom_point()

won´t work.

Let´s do some modifications

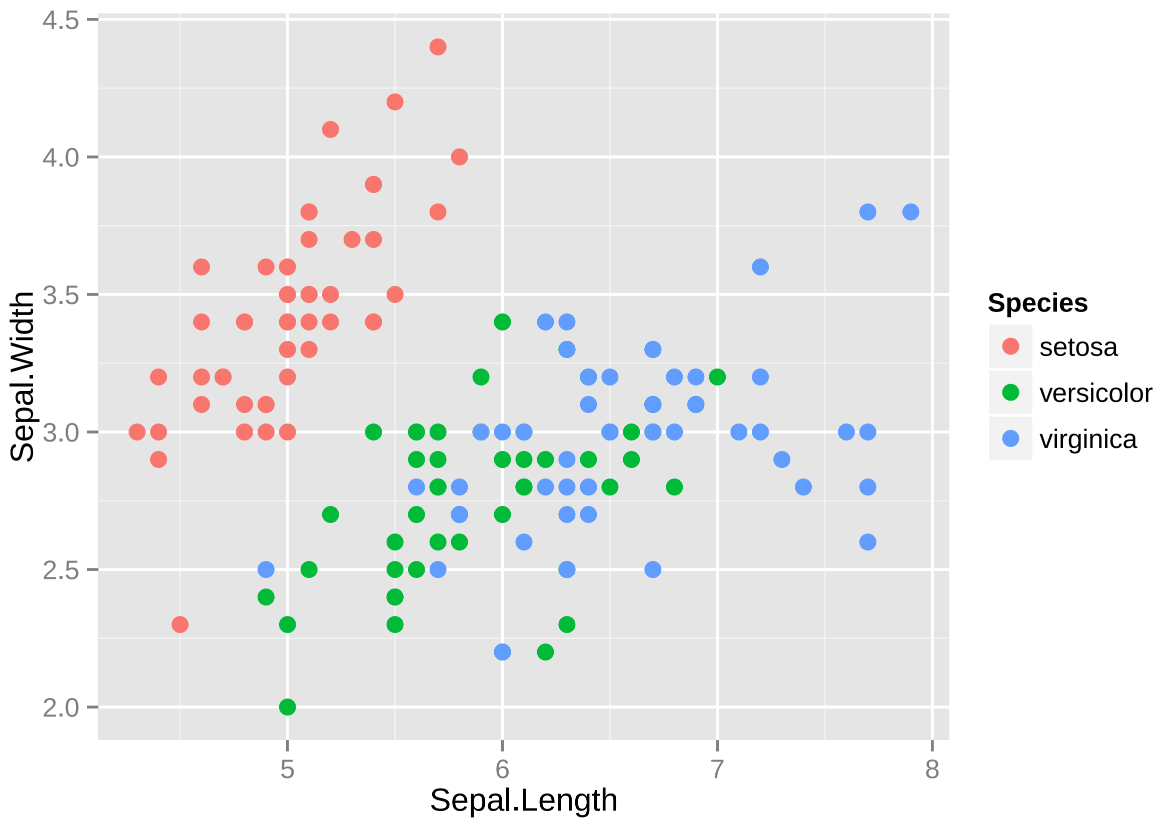
Increase the size of points

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point(size = 3)
```



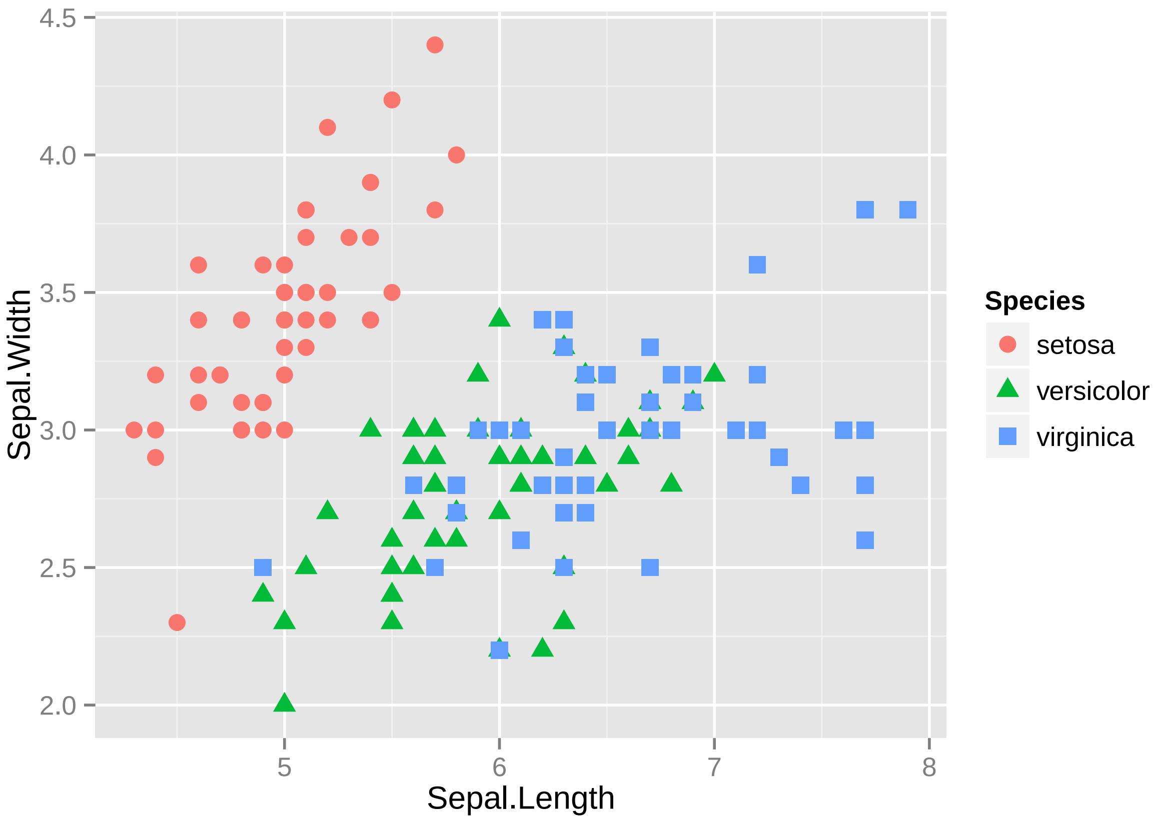
Add some color

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 3)
```



Differentiate points by shape

```r
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(aes(shape = Species), size = 3)
```

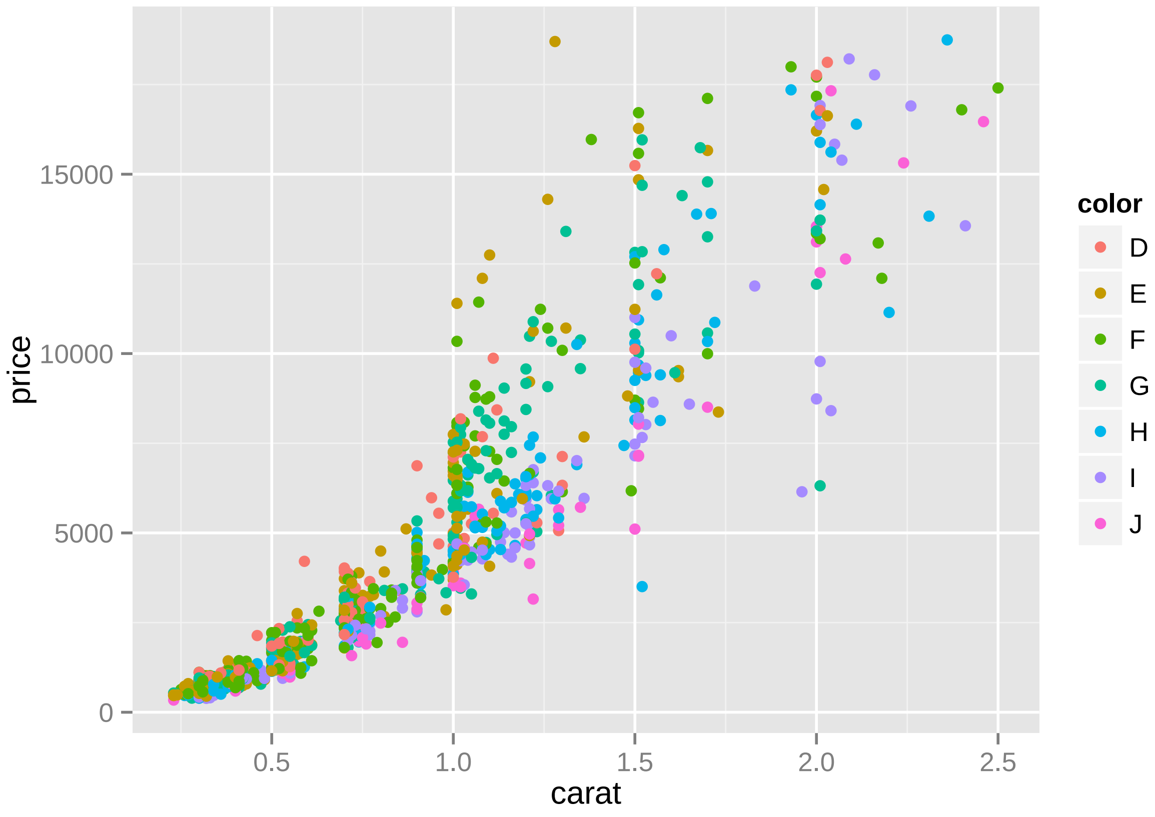


Exercise 1

Make a small sample of the diamonds dataset

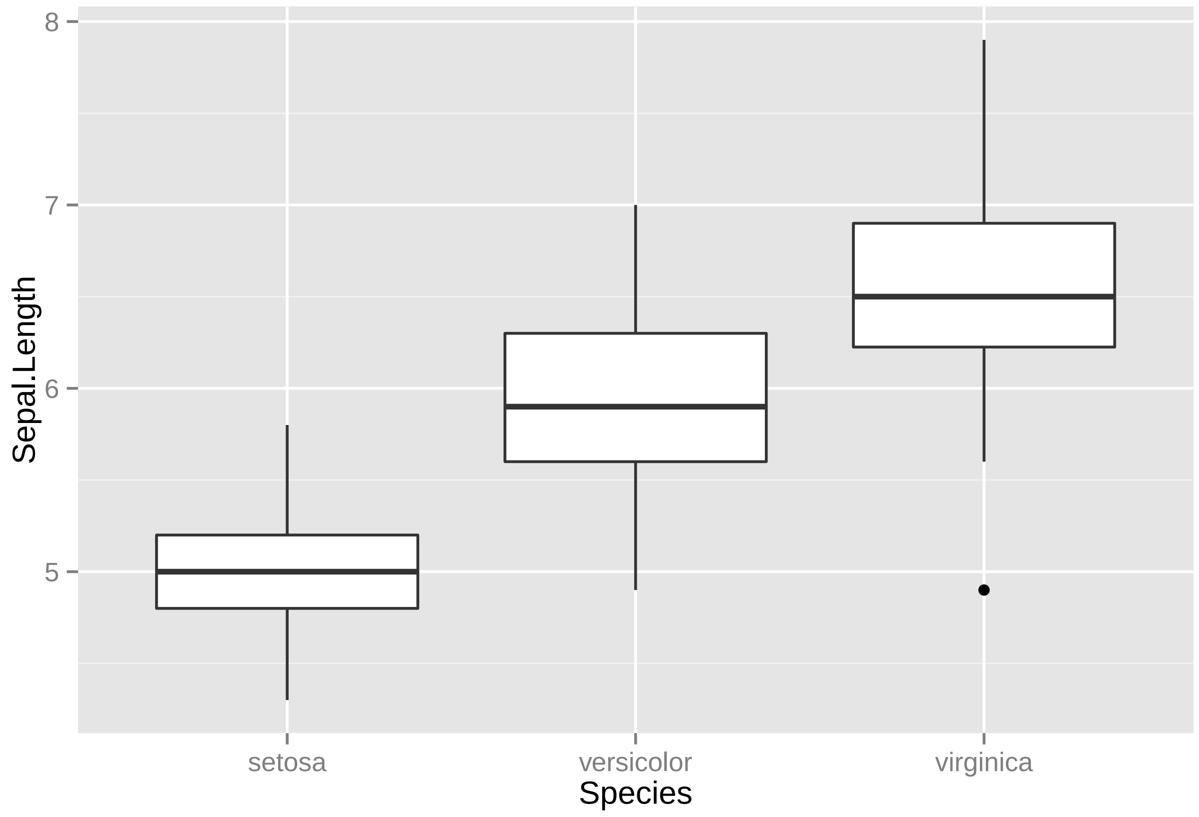
```
d2 <- diamonds[sample(1:dim(diamonds)[1], 1000), ]
```

Then generate the graph below:



Box plots

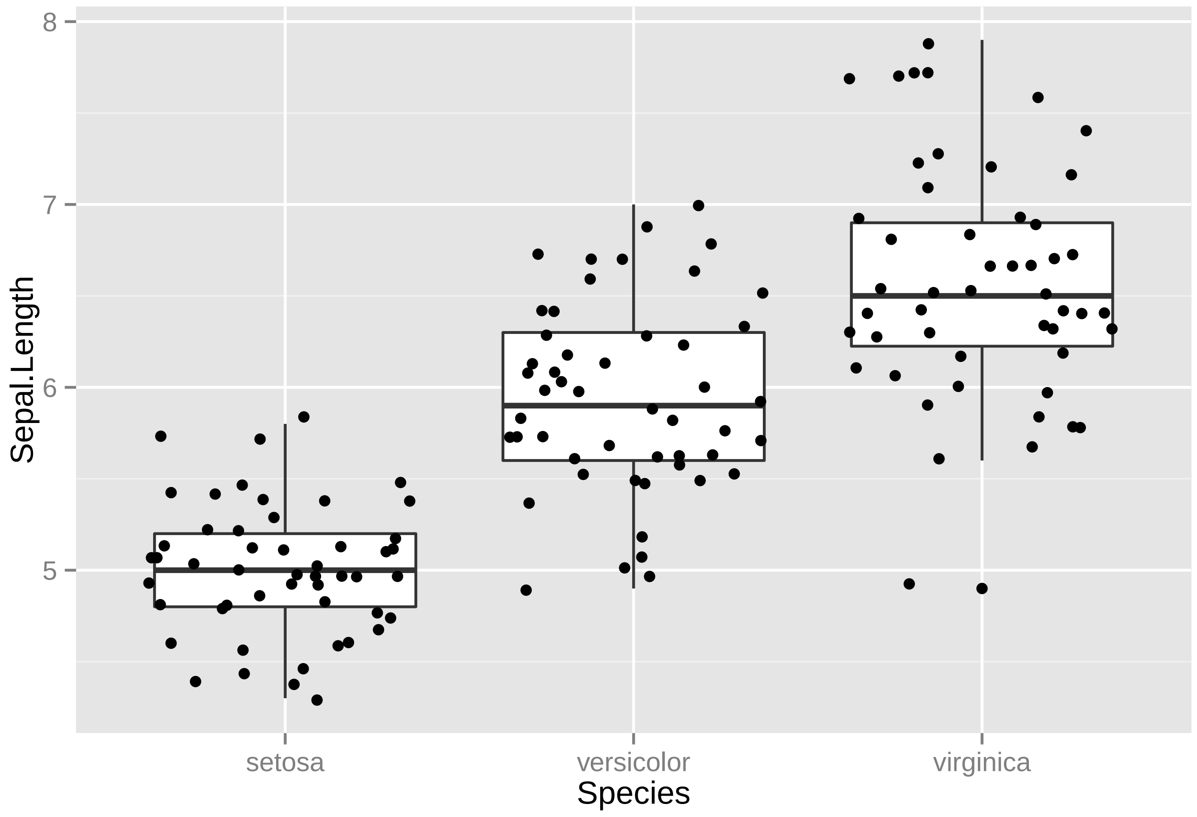
```r
ggplot(iris, aes(Species, Sepal.Length)) +
  geom_boxplot()
```



What if you also want to see all the individual points?

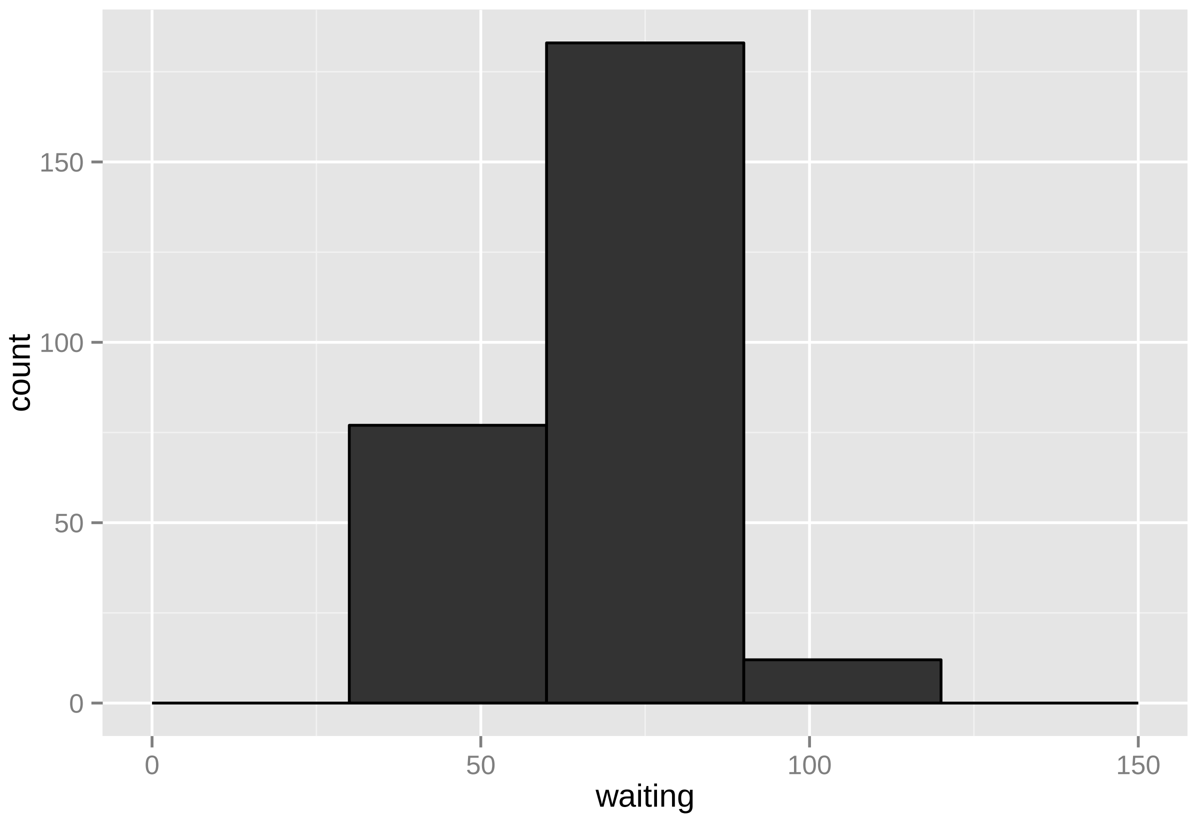
This is where the ggplot excels.

```
ggplot(iris, aes(Species, Sepal.Length)) +
geom_boxplot() +
geom_point(position = "jitter")
```

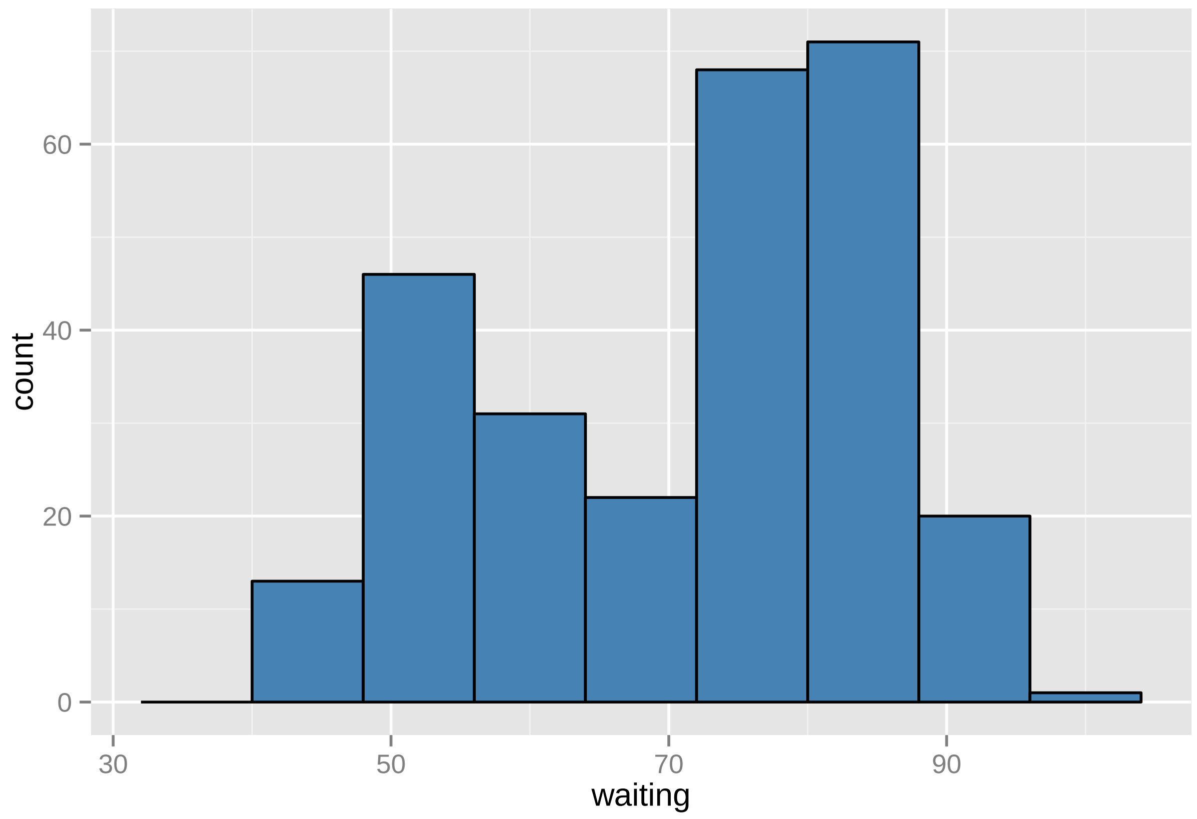


Histograms

```r
h <- ggplot(faithful, aes(x = waiting))
h + geom_histogram(binwidth = 30, colour = "black")
```

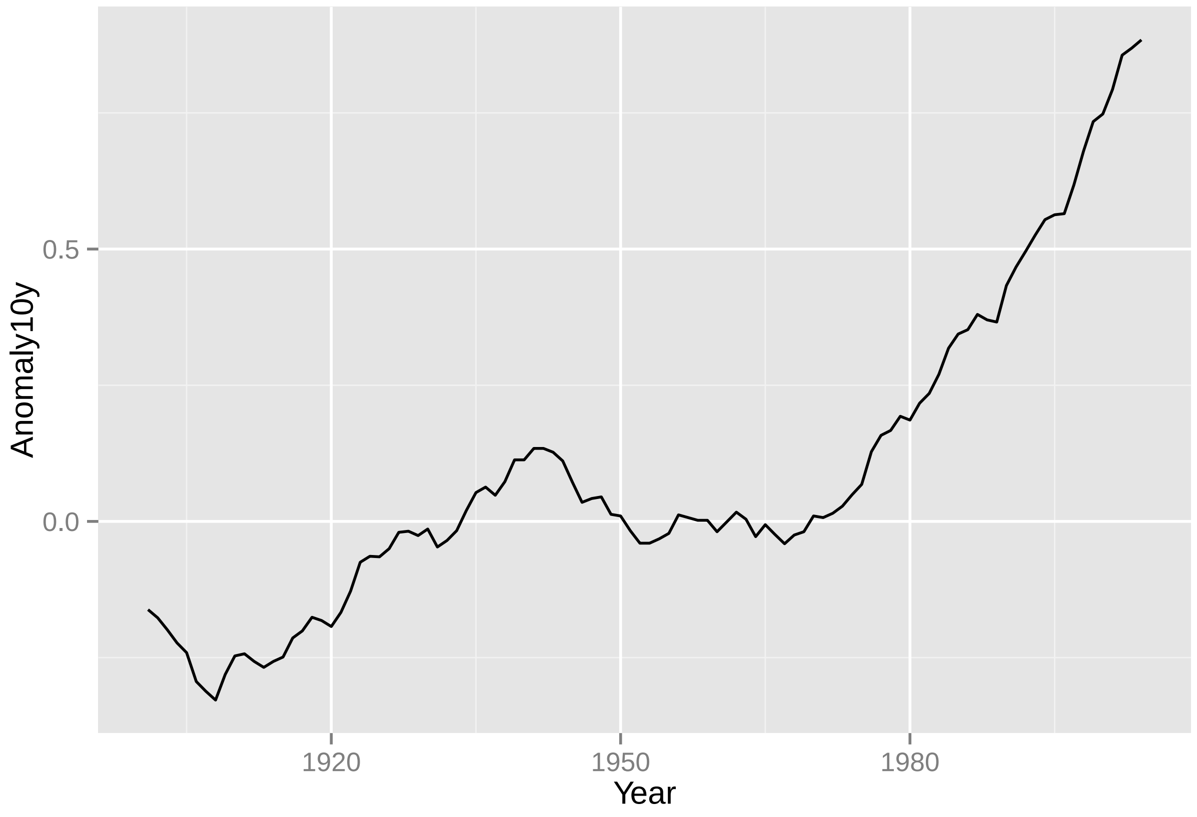


```
h + geom_histogram(binwidth = 8, fill = "steelblue",
colour = "black")
```



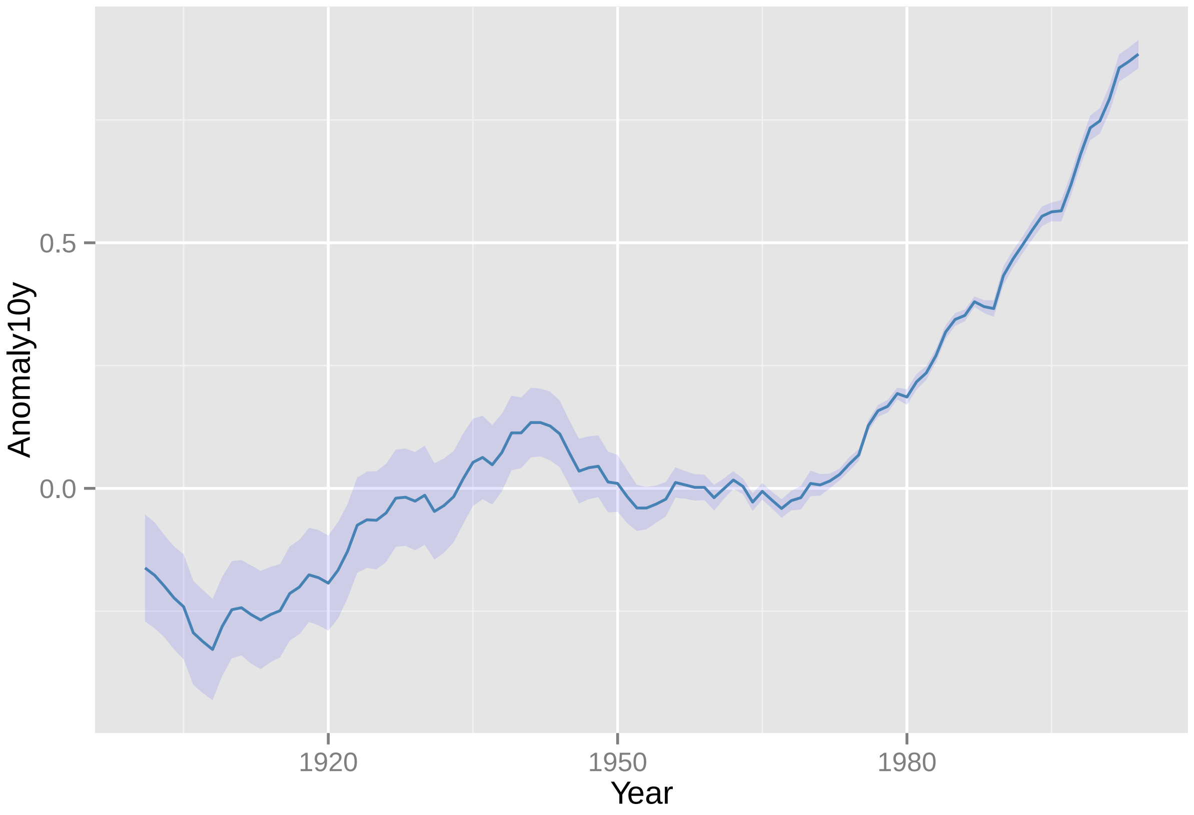
Line plots

```
climate <- read.csv("climate.csv", header = T)
ggplot(climate, aes(Year, Anomaly10y)) +
geom_line()
```



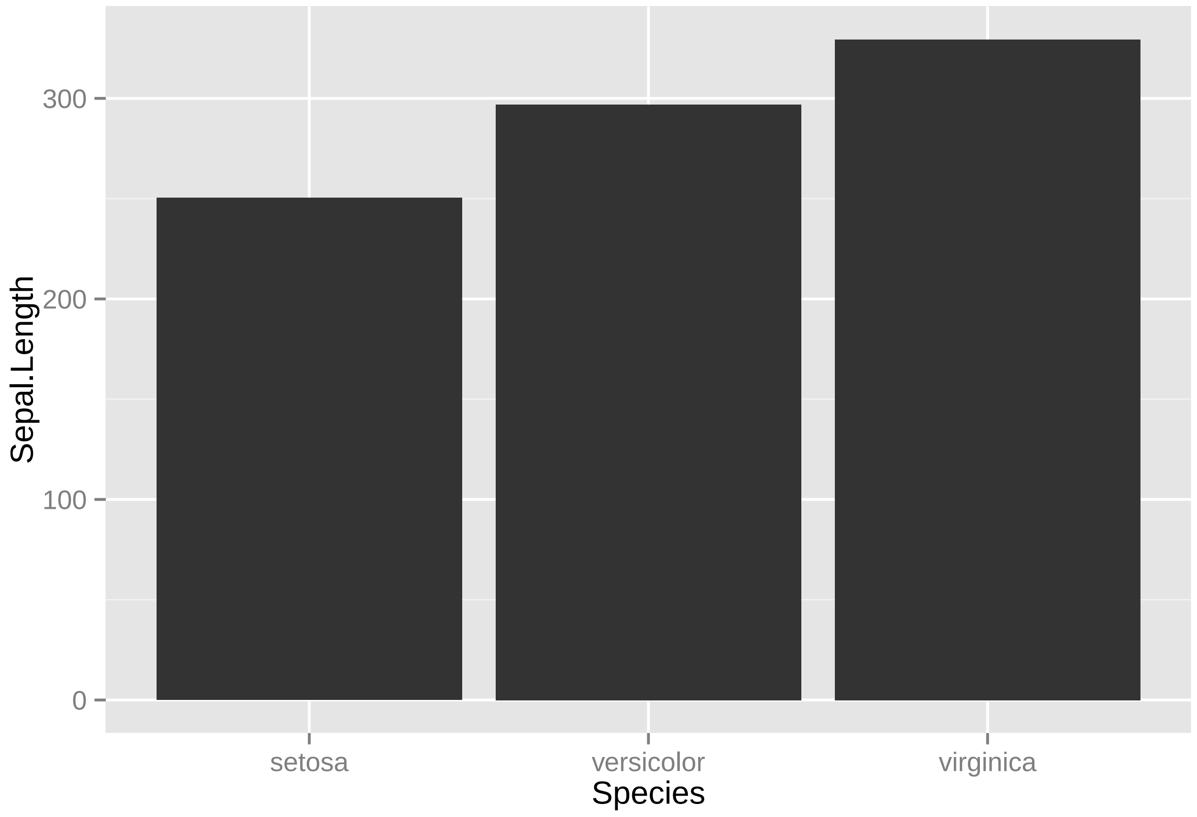
We can also plot confidence regions

```
ggplot(climate, aes(Year, Anomaly10y)) +
  geom_ribbon(aes(ymin = Anomaly10y - Unc10y,
                  ymax = Anomaly10y + Unc10y),
              fill = "blue", alpha = .1) +
  geom_line(color = "steelblue")
```



Bar plots

```
ggplot(iris, aes(Species, Sepal.Length)) +
geom_bar(stat = "identity")
```



tidyr and dplyr are key for preparing data into ggplot friendly format.

- dplyr
- select
- filter
- mutate
- summarise
- arrange
- tidyr}.
- gather
- separate
- spread

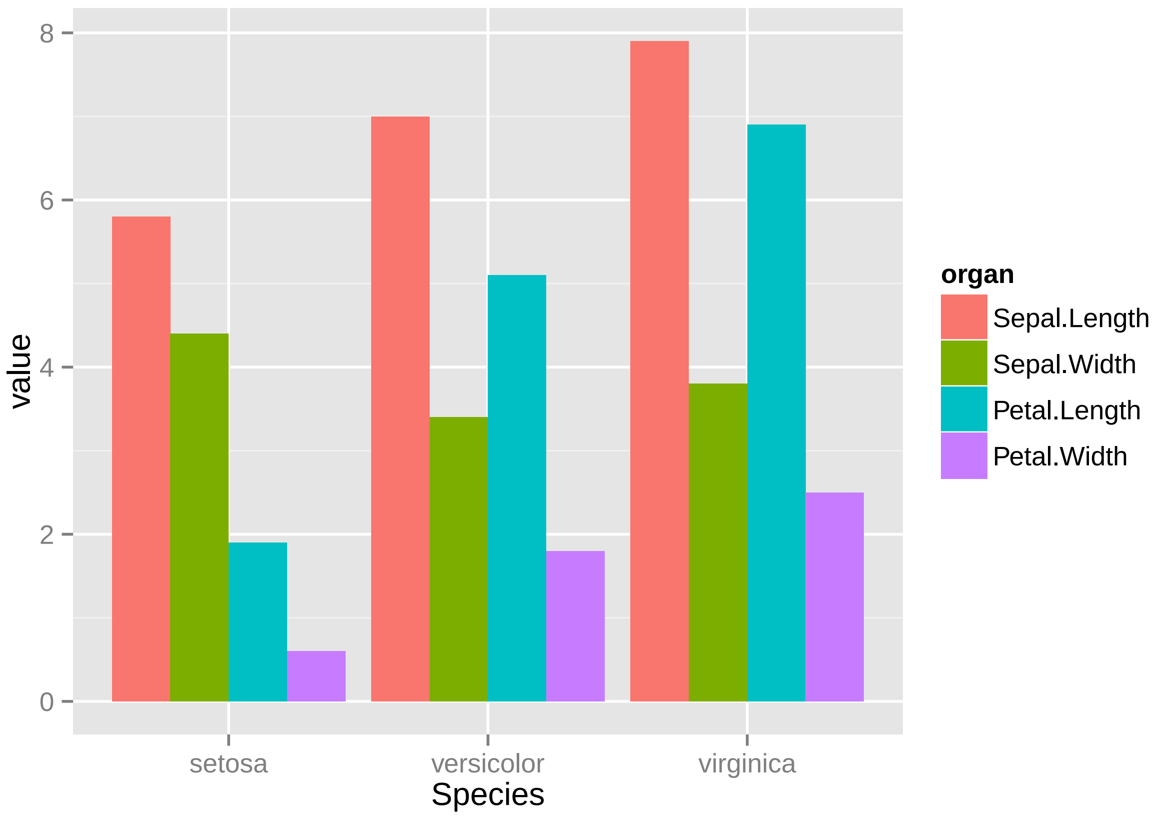
```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
df <- gather(iris, organ, value, -Species)
head(iris)
```

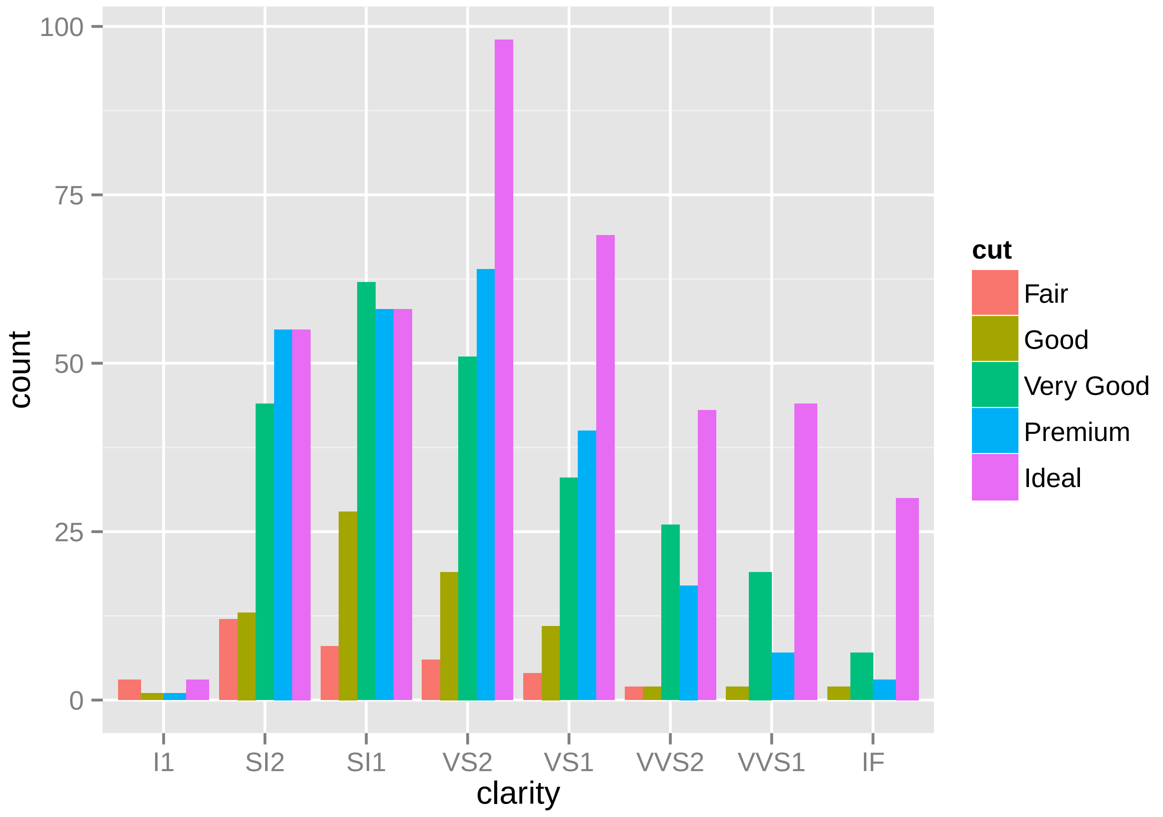
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
ggplot(df, aes(Species, value, fill = organ)) +
geom_bar(stat = "identity", position = "dodge")
```



Exercise

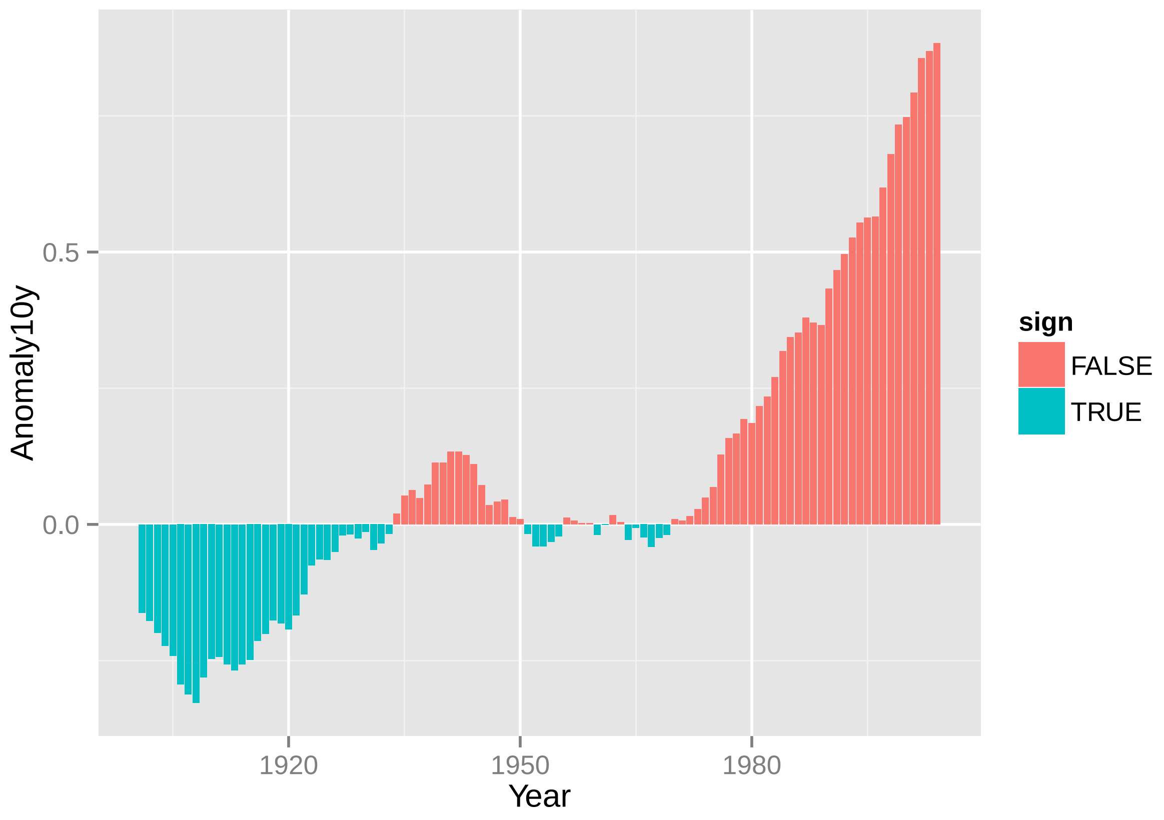
Using the d2 dataset you created earlier, generate this plot below. Take a quick look at the data first to see if it needs to be binned.



Exercise

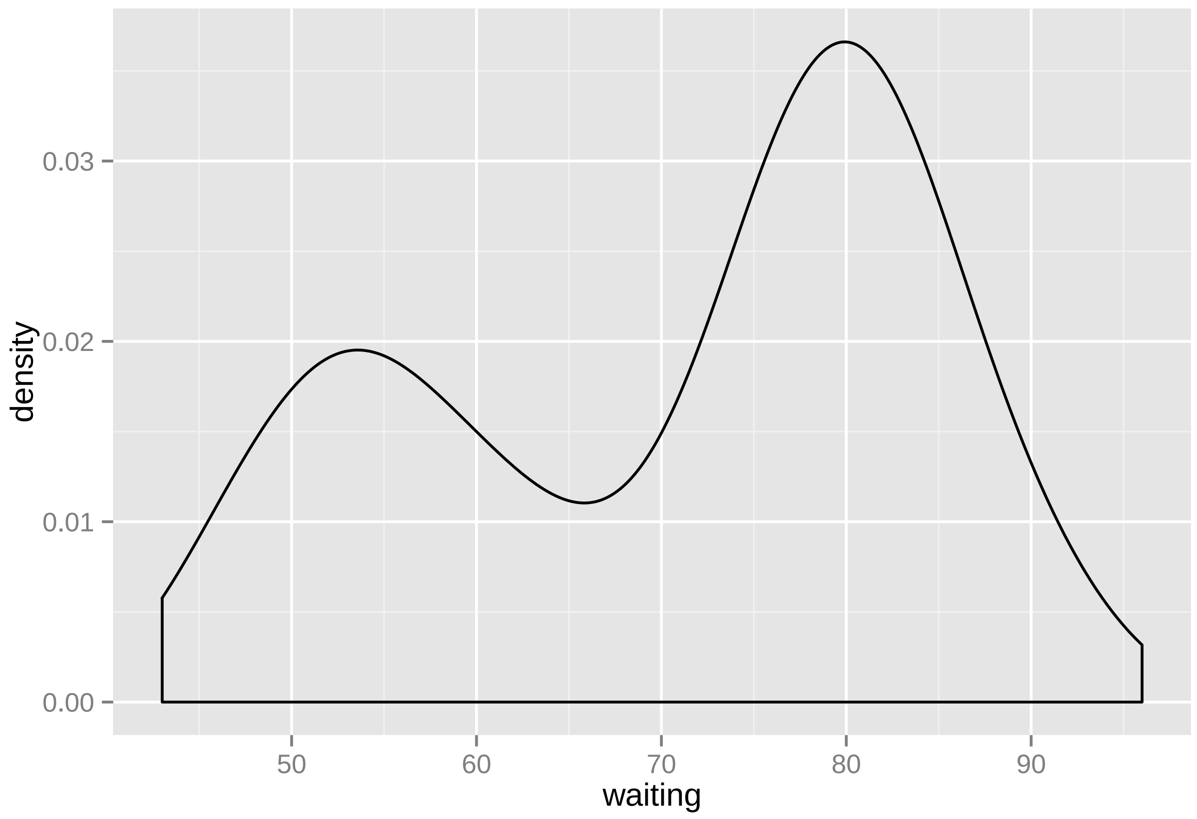
Using the climate dataset, create a new variable called sign. Make it logical (true/false) based on the sign of Anomaly10y.

```
## Warning in loop_apply(n, do.ply): Stacking not well defined when ymin != 0
```



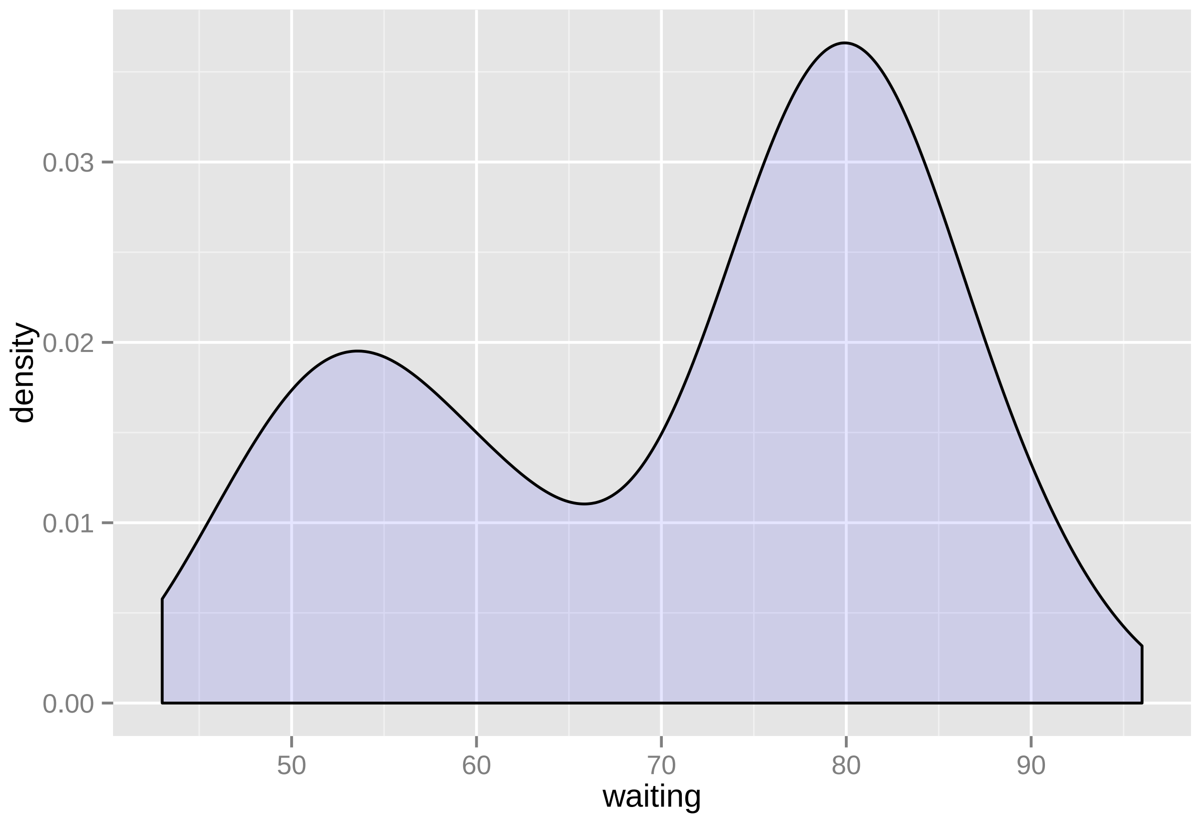
Density Plots

```
ggplot(faithful, aes(waiting)) +
  geom_density()
```



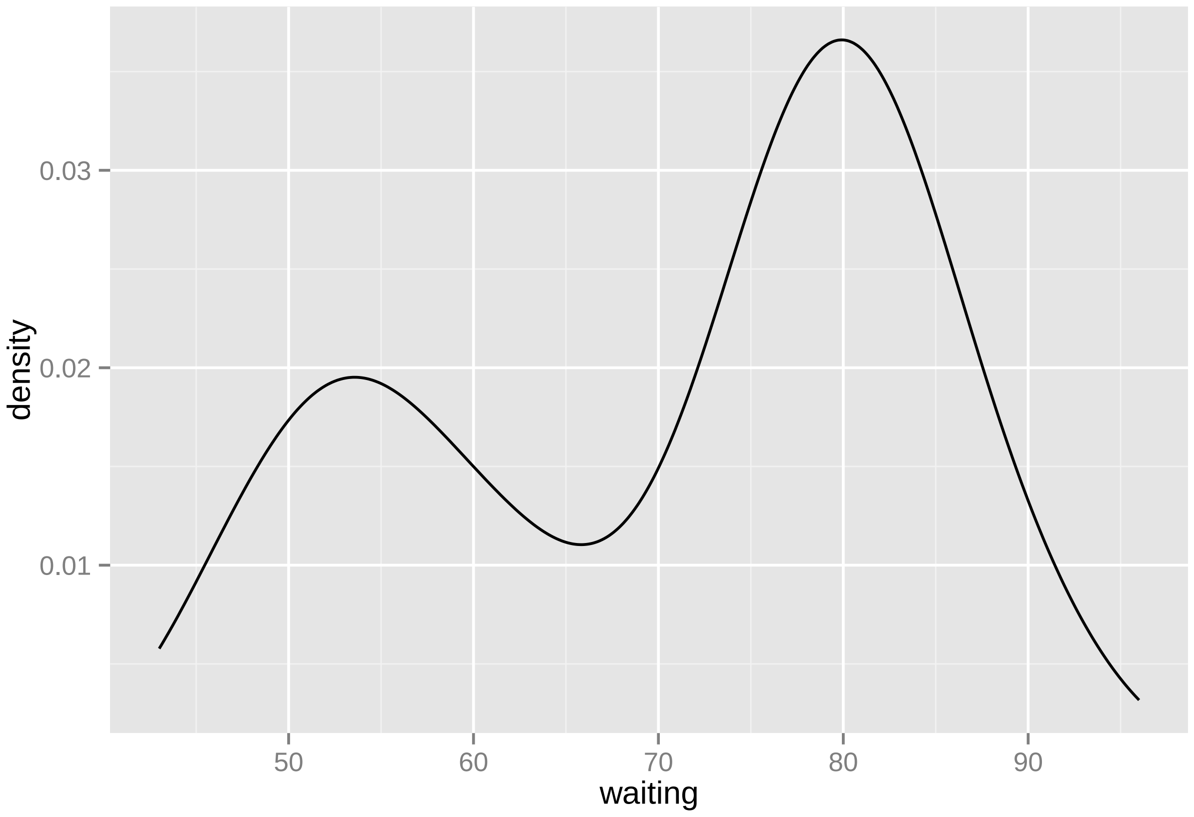
some polishing

```
ggplot(faithful, aes(waiting)) +
  geom_density(fill = "blue", alpha = 0.1)
```



another way to do the same

```r
ggplot(faithful, aes(waiting)) +
  geom_line(stat = "density")
```



Mapping Variables to colors

Different ways to specify colors:

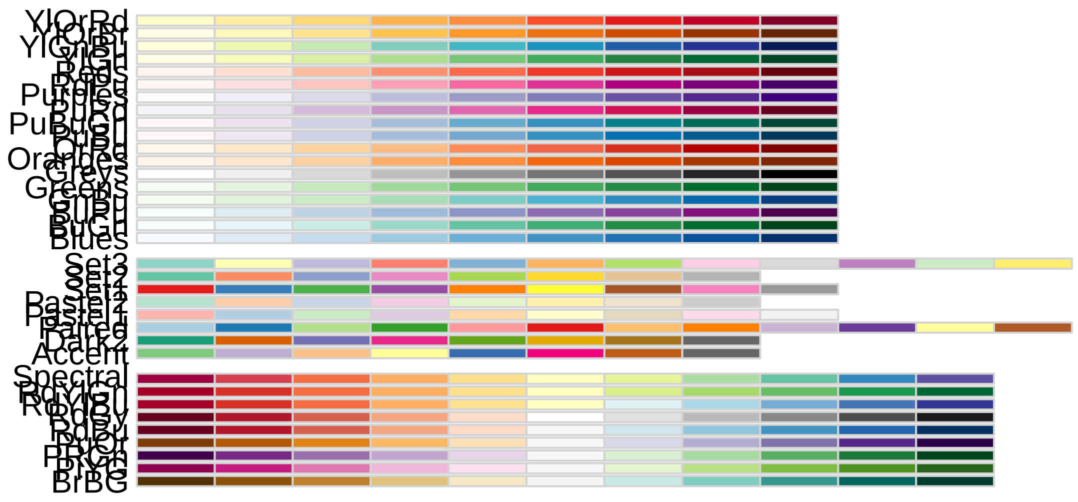
- aes(color = variable)
- aes(color = “black”)

Or add it as a scale

- scale_fill_manual(values = c(“color1”, “color2”))

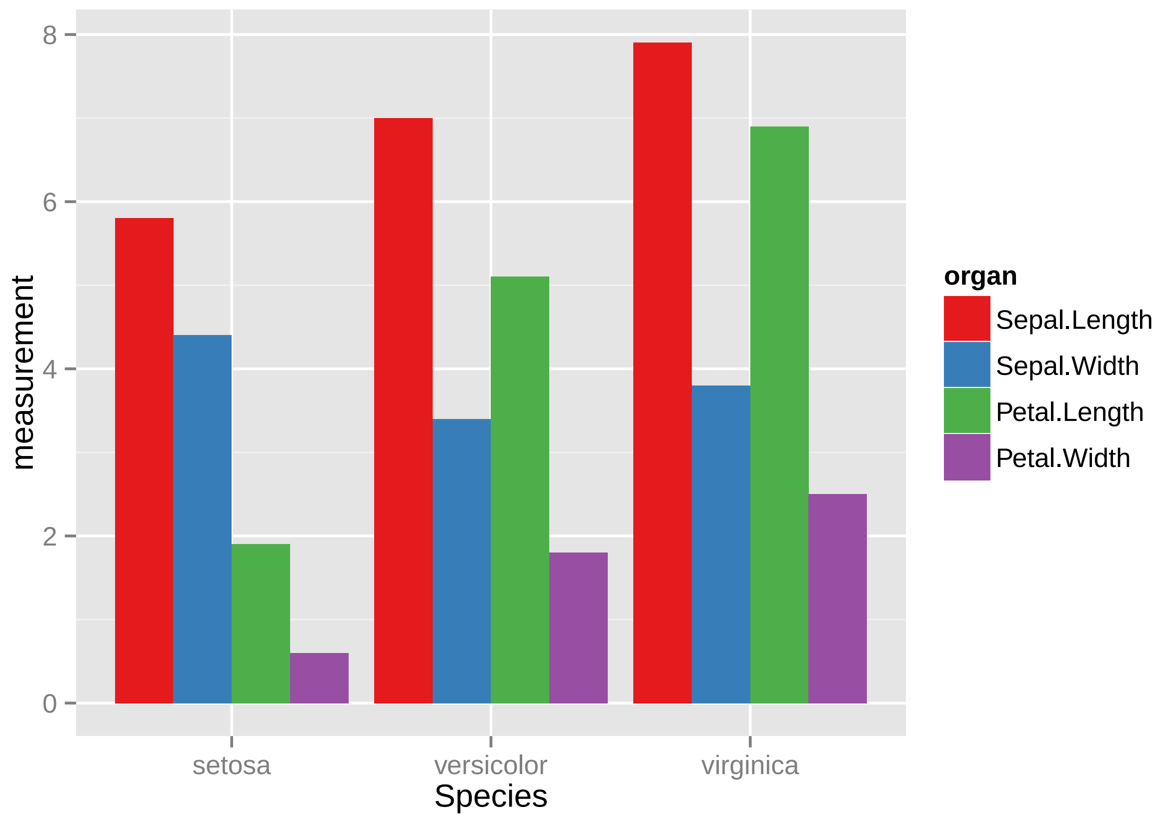
The RColorBrewer package

```
display.brewer.all()
```



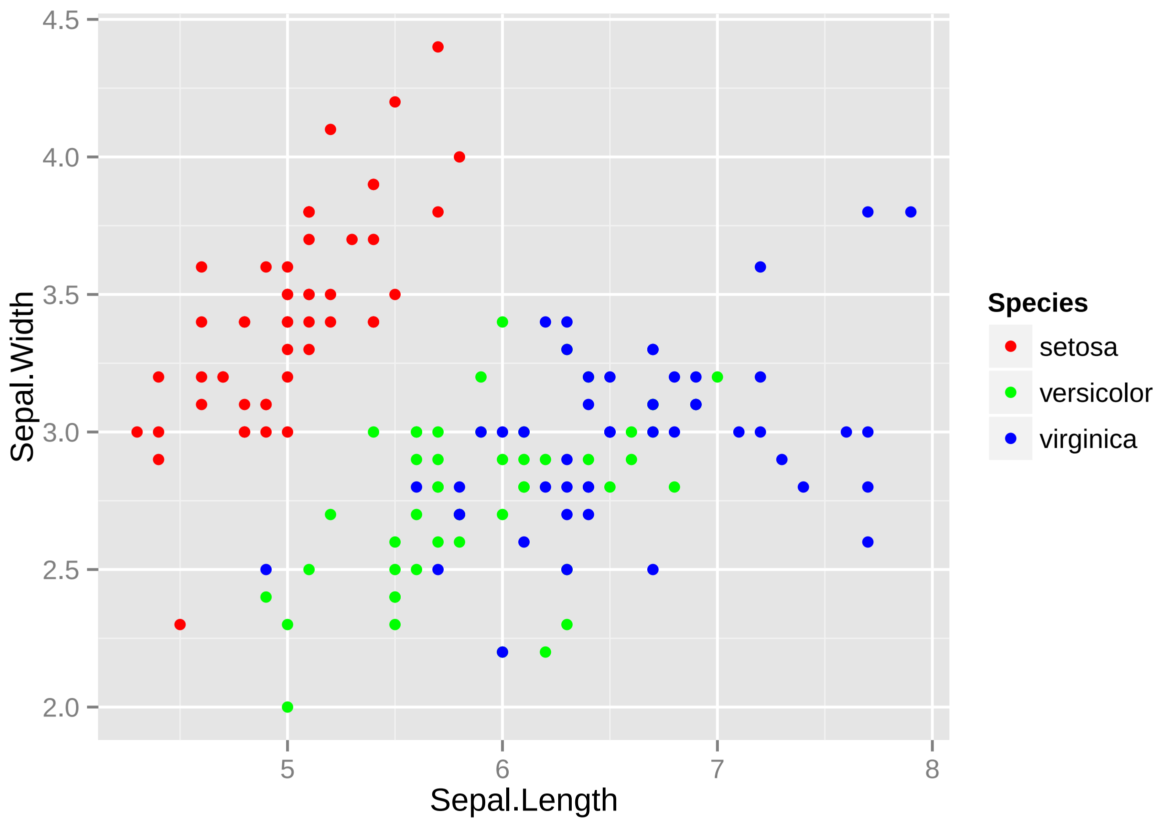
Using a color brewer palette

```
df  <- gather(iris, organ, measurement, -Species)
ggplot(df, aes(Species, measurement, fill = organ)) +
geom_bar(stat = "identity", position = "dodge") +
scale_fill_brewer(palette = "Set1")
```



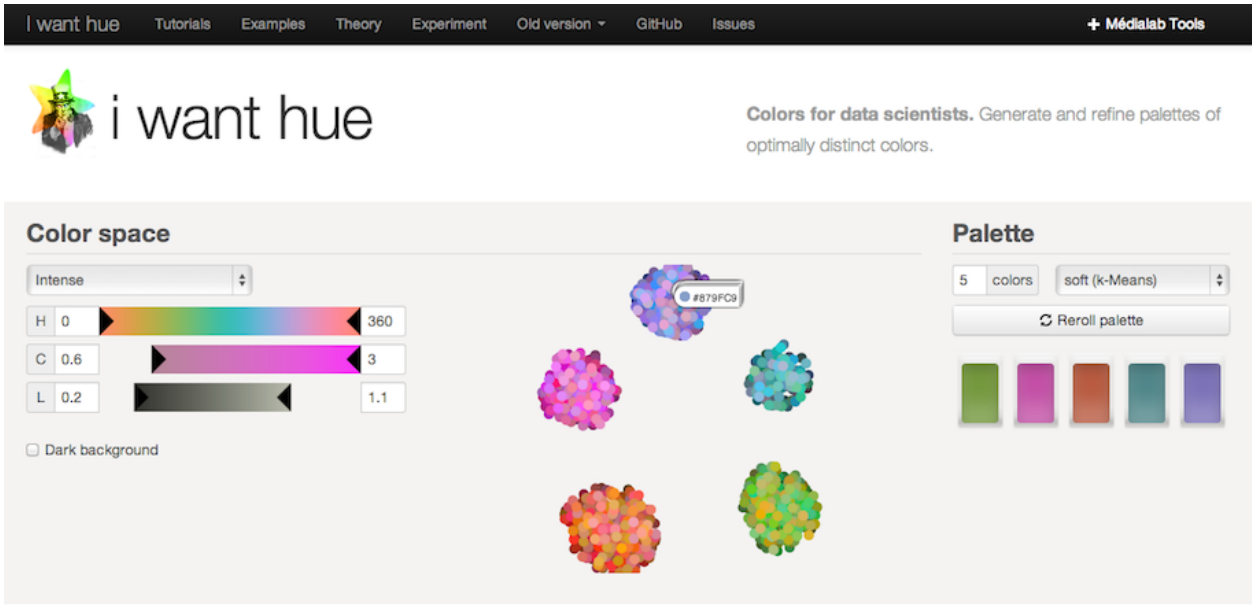
Manual color scale

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point() +
  scale_color_manual(values = c("red", "green", "blue"))
```



Refer to a color chart for beautiful visualizations

I want hue



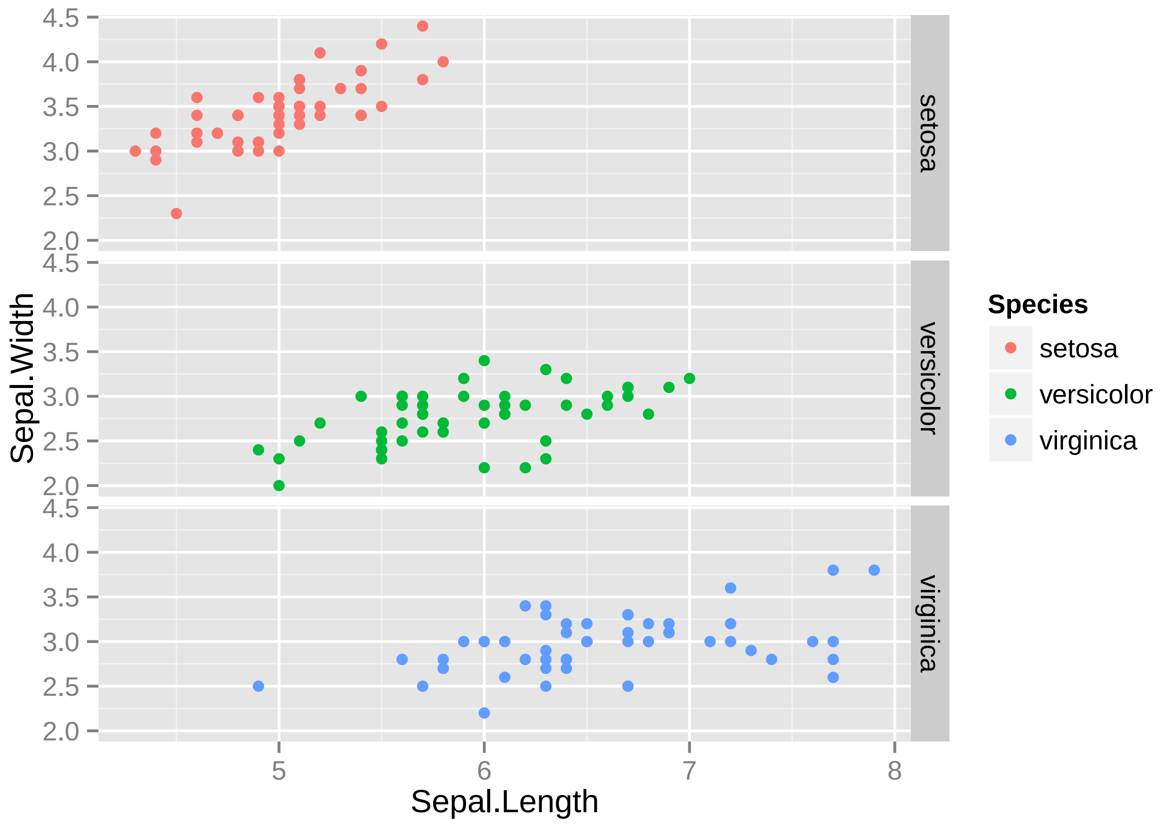
Colors for data scientists. Generate and refine palettes of optimally distinct colors.

Faceting

Facets allow you to keep your graphs clean and easy to read. Very useful in time series data.

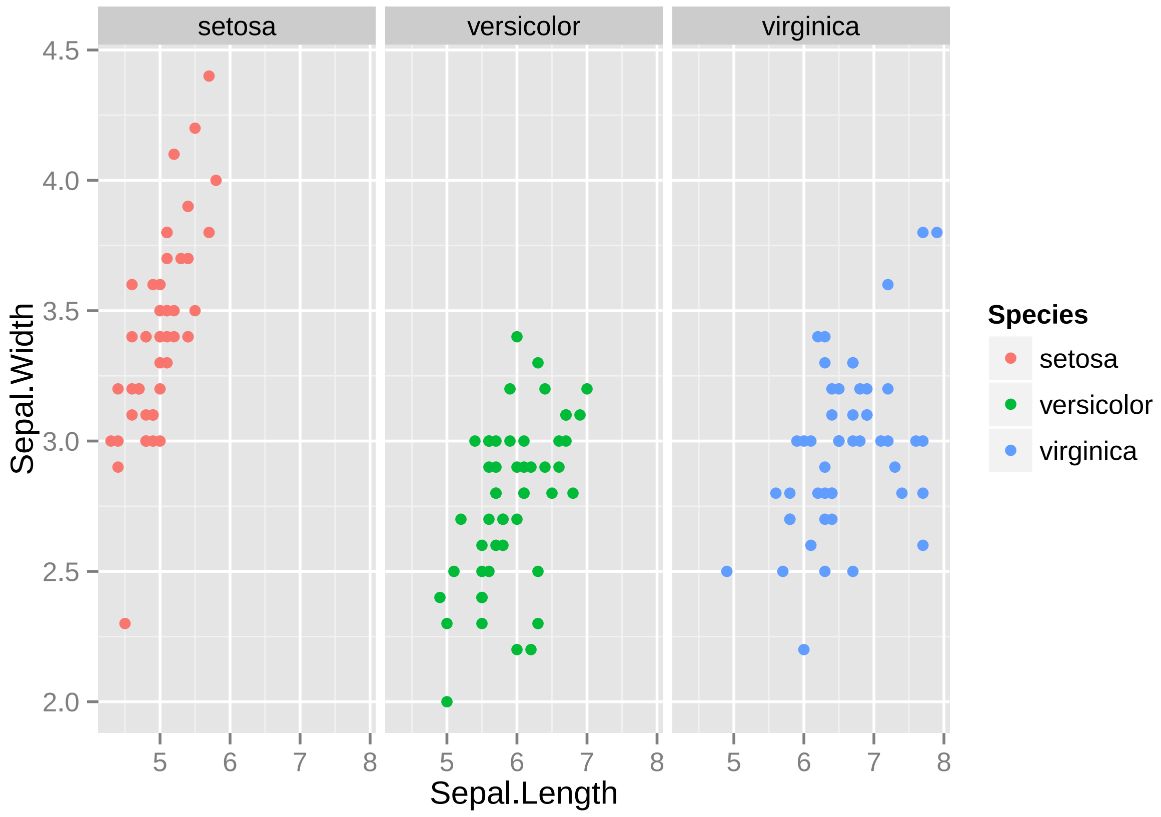
Faceting along columns

```r
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
geom_point() +
facet_grid(Species ~ .)
```



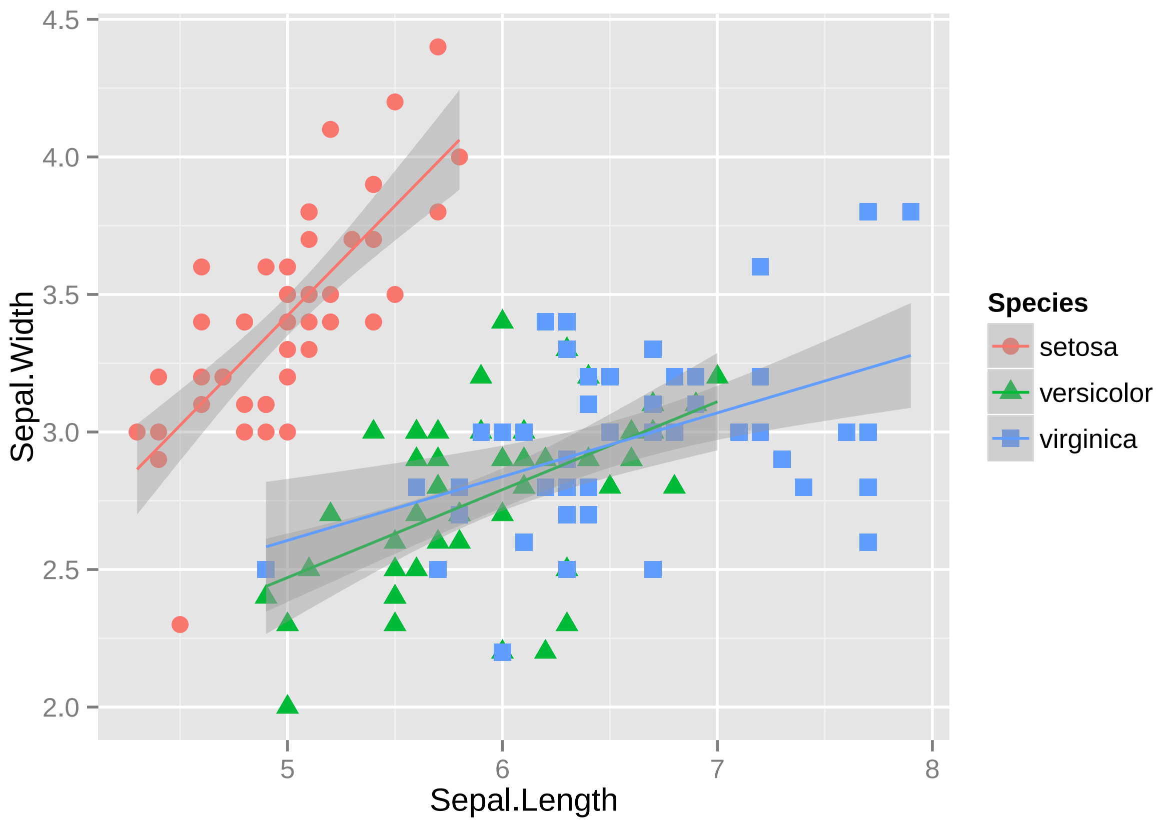
and along rows

```r
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point() +
  facet_grid(. ~ Species)
```



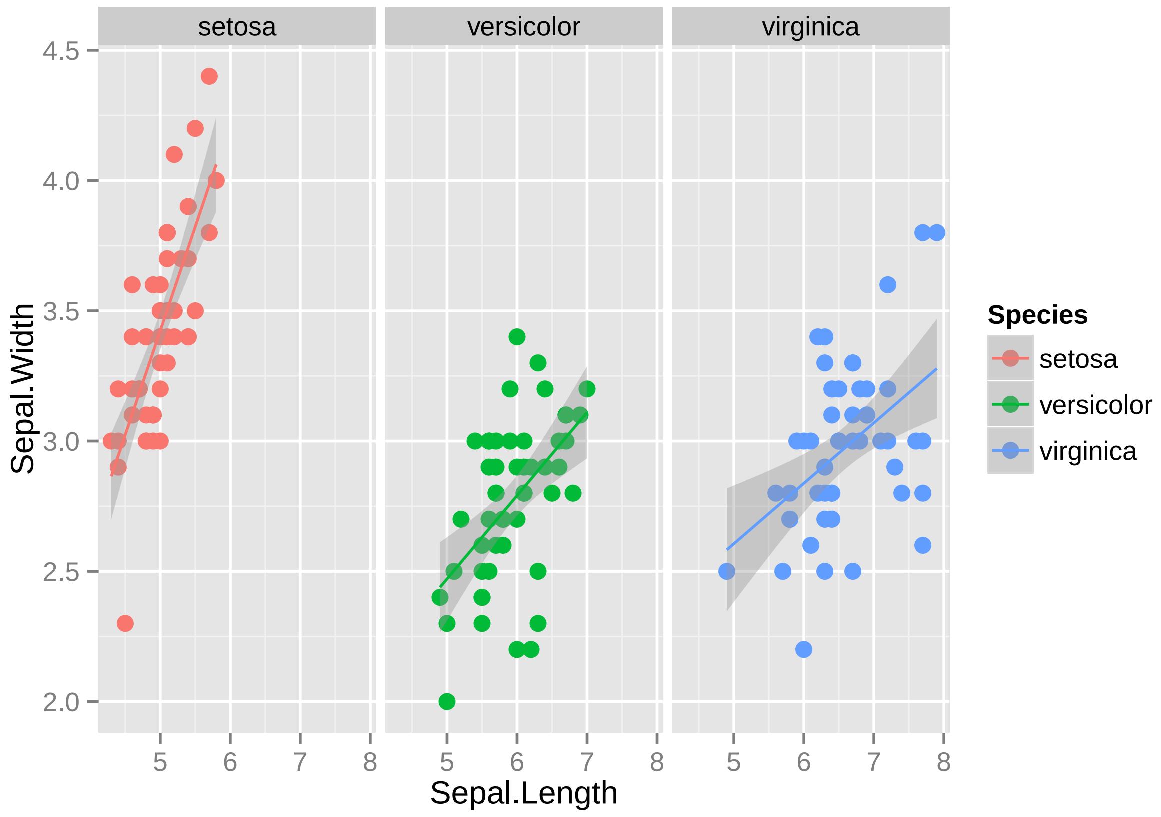
Adding smoothers

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(aes(shape = Species), size = 3) +
  geom_smooth(method = "lm")
```



here we have a typical example where facetting will be usefull

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 3) +
  geom_smooth(method = "lm") +
  facet_grid(. ~ Species)
```

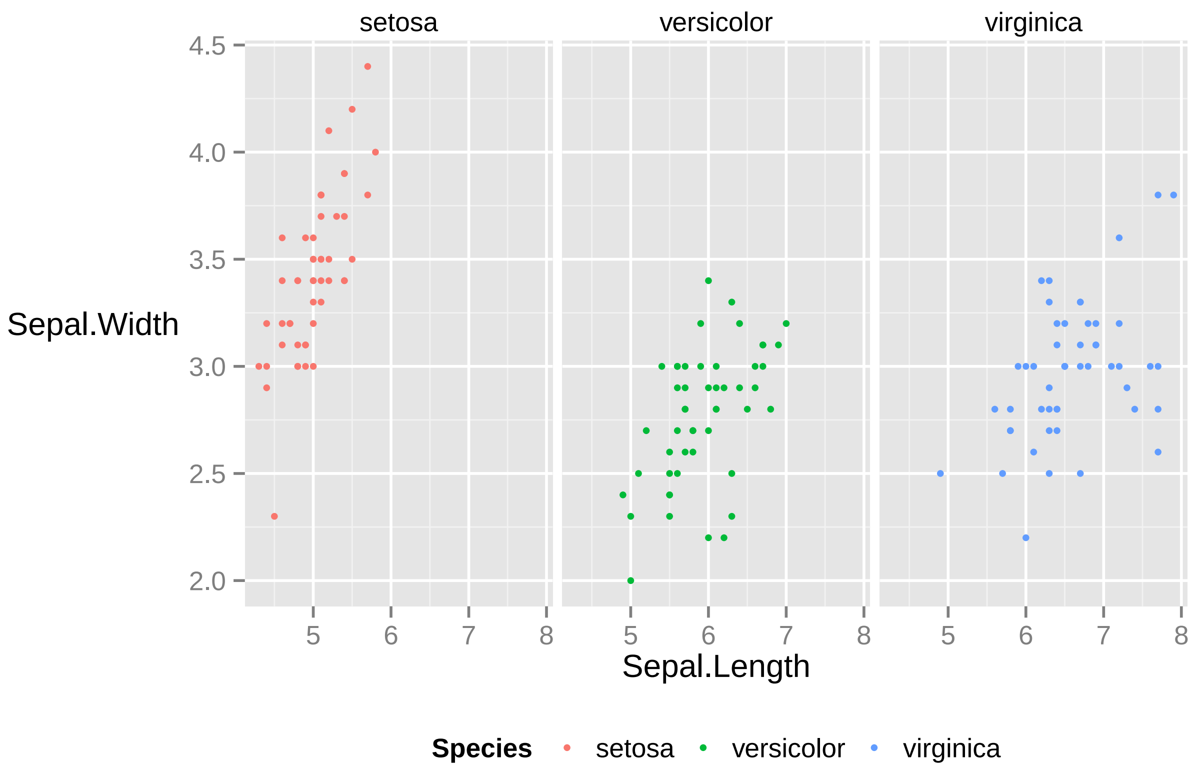


Themes

Themes are a great way to define custom plots

A themed plot

```r
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 1.2, shape = 16) +
  facet_wrap( ~ Species) +
  theme(legend.key = element_rect(fill = NA),
        legend.position = "bottom",
        strip.background = element_rect(fill = NA),
        axis.title.y = element_text(angle = 0))
```

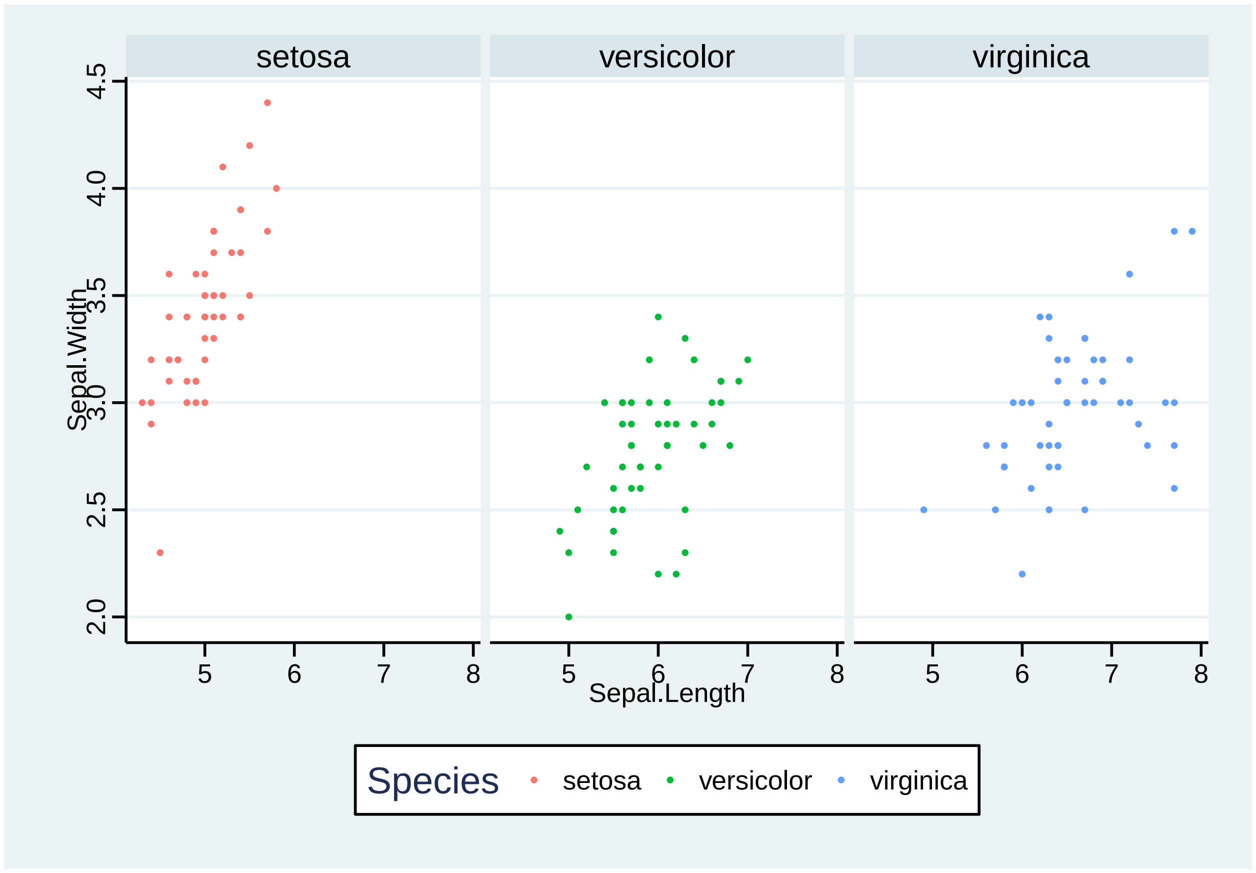


ggthemes library

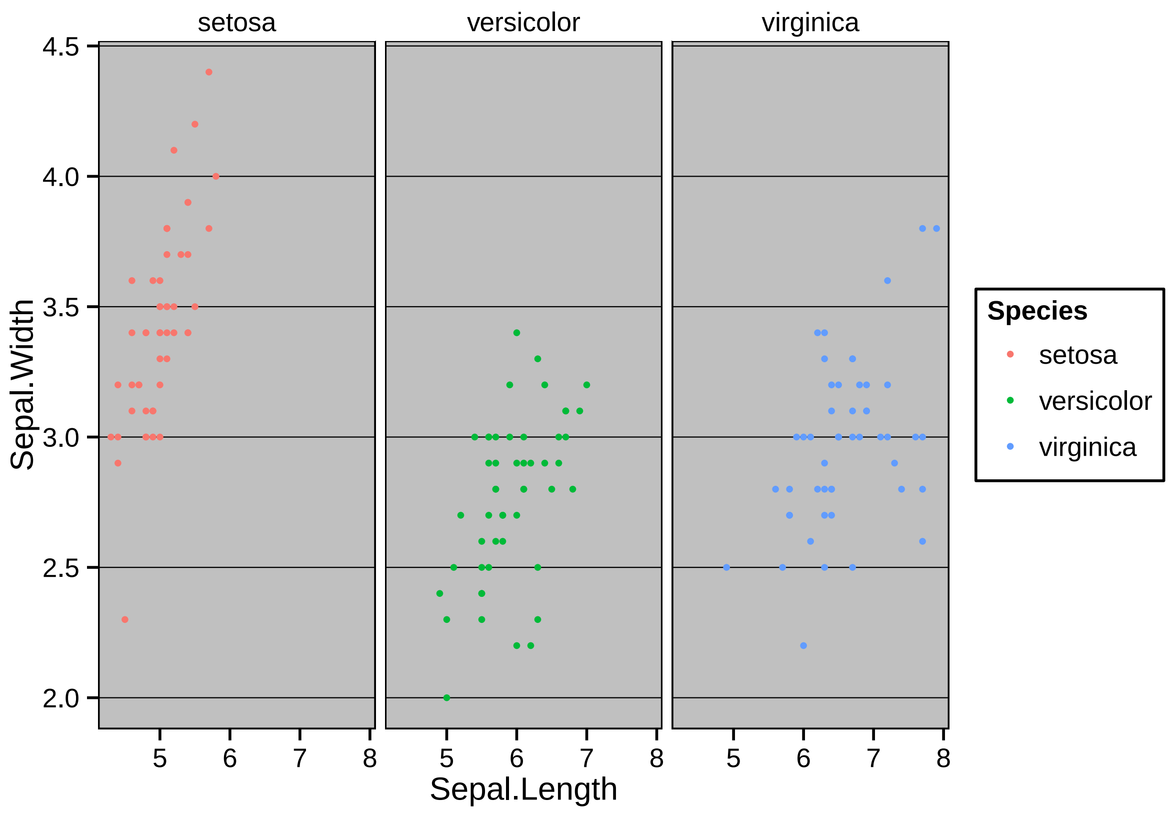
```
install.packages('ggthemes', repos="http://cran.rstudio.com/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/g5/s26yfnqs4dj12pn6w88g__cc0000gn/T//RtmpAsixXD/downloaded_packages
```

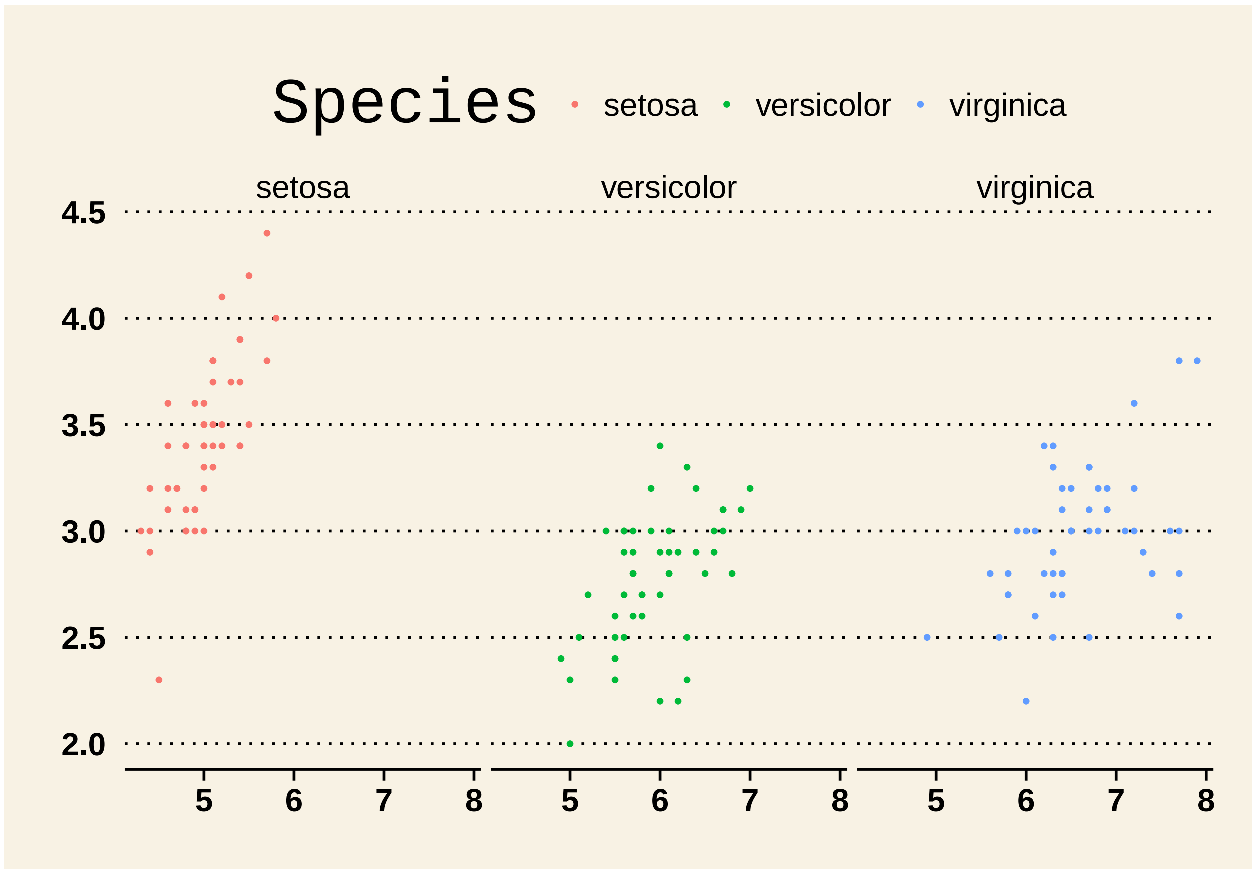
```
library(ggthemes)

plot <- ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 1.2, shape = 16) +
  facet_wrap( ~ Species)

# Then add one of these themes to your plot

plot + theme_stata()
```



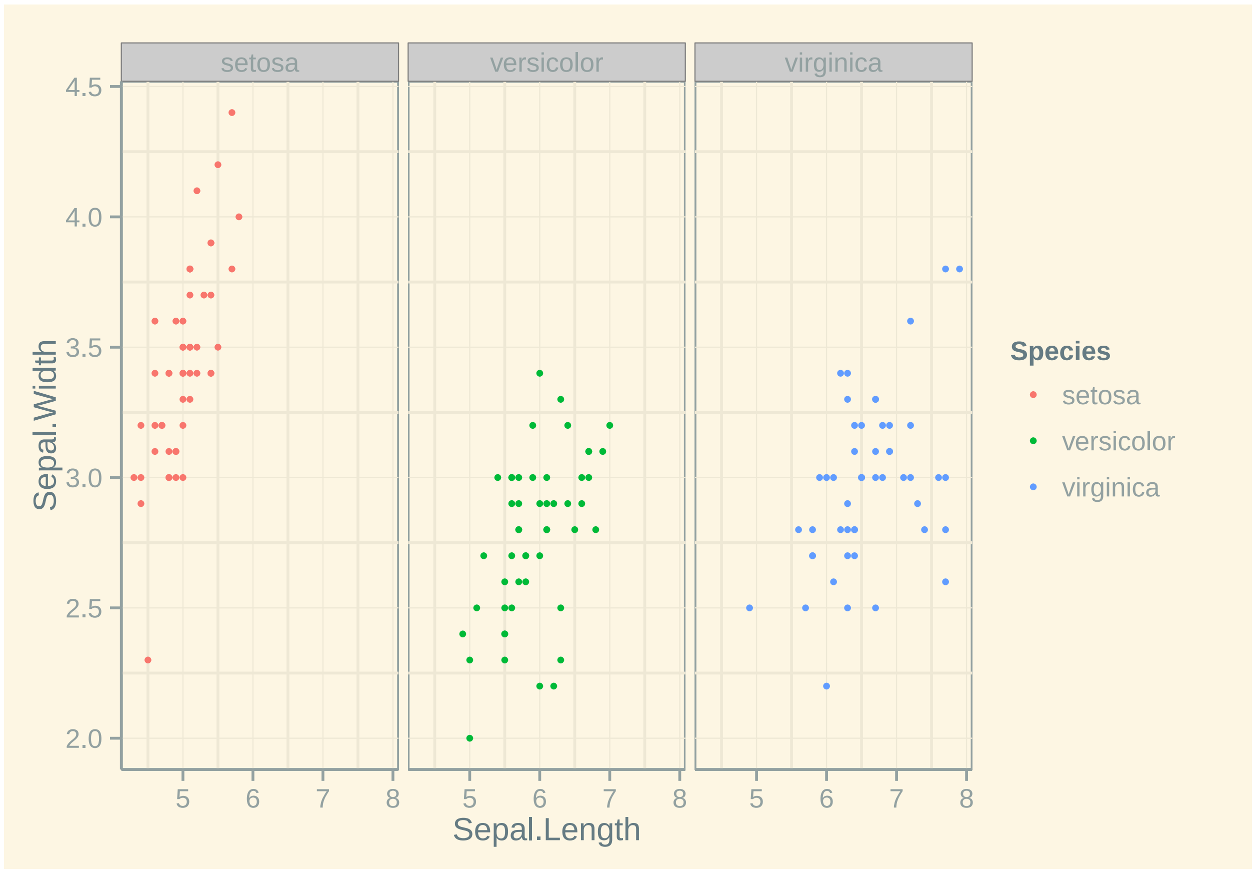
```
plot + theme_excel()
```



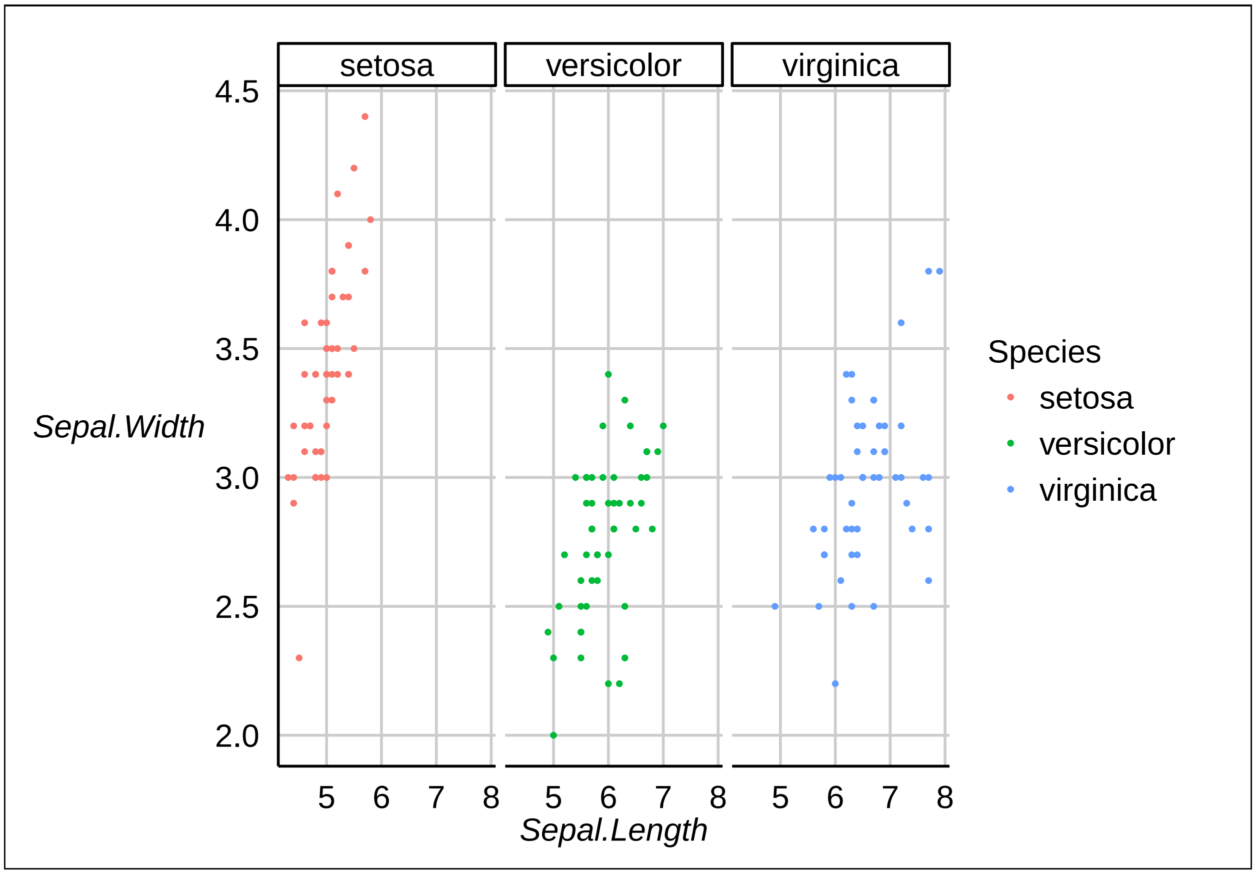
```r
plot + theme_wsj()
```



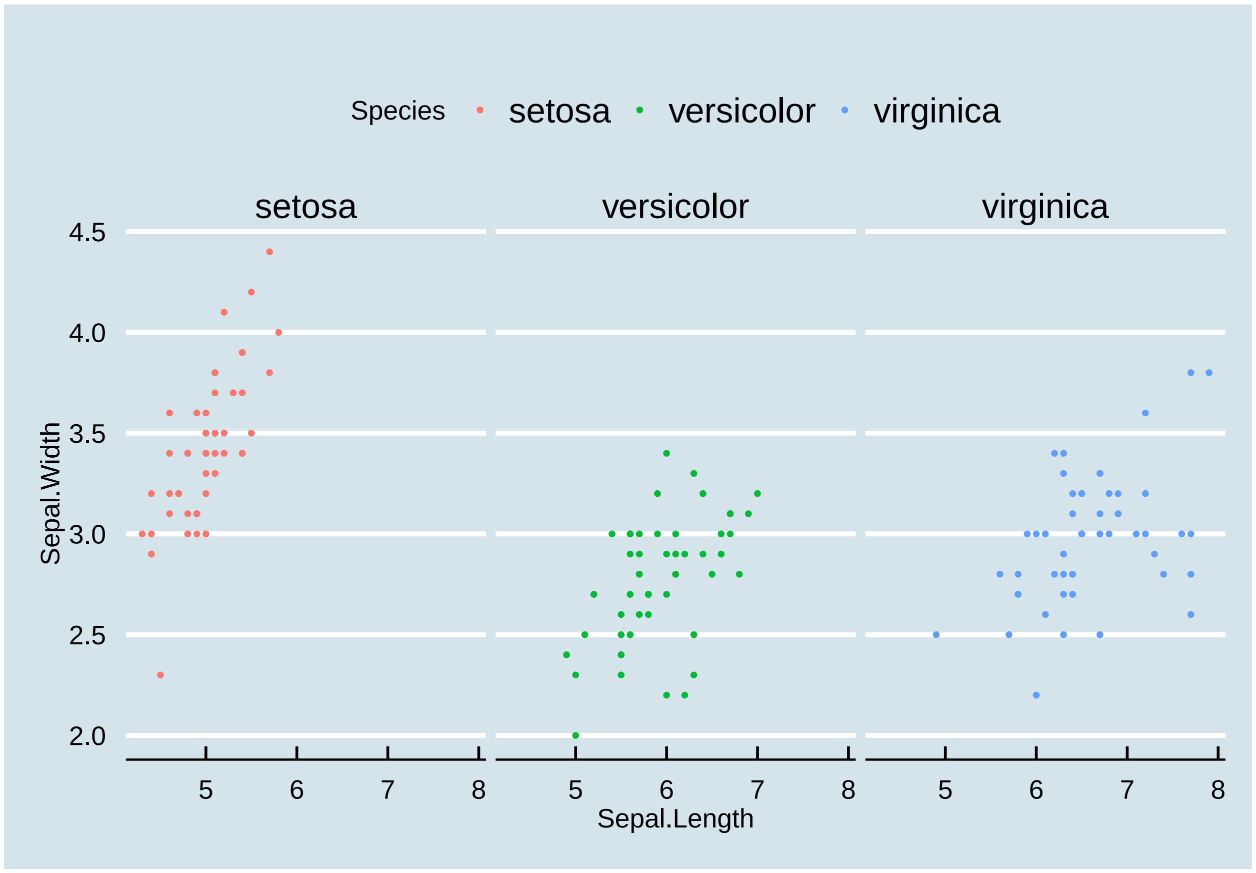
```
plot + theme_solarized()
```



```
plot + theme_gdocs()
```



```r
plot + theme_economist()
```



Create functions to automate your plotting

my_custom_plot <- function(df, title = “”, . . .) { ggplot(df, . . .) + ggtitle(title) + whatever_geoms() + theme(. . .)

Then just call your function to generate a plot. It’s a lot easier to fix one function that do it over and over for many plots

plot1 <- my_custom_plot(dataset1, title = “Figure 1”)

Scales

```
scale_fill_discrete(); scale_colour_discrete()
```

```
## discrete_scale(aesthetics = "fill", scale_name = "hue", palette = hue_pal(h,
##     c, l, h.start, direction), na.value = na.value)

## discrete_scale(aesthetics = "colour", scale_name = "hue", palette = hue_pal(h,
##     c, l, h.start, direction), na.value = na.value)
```

```
scale_fill_hue(); scale_color_hue()
```

```
## discrete_scale(aesthetics = "fill", scale_name = "hue", palette = hue_pal(h,
##     c, l, h.start, direction), na.value = na.value)
## discrete_scale(aesthetics = "colour", scale_name = "hue", palette = hue_pal(h,
##     c, l, h.start, direction), na.value = na.value)
```

```r
scale_fill_manual();  scale_color_manual()
```

```
## discrete_scale(aesthetics = aesthetic, scale_name = "manual",
##     palette = pal)
```

```
## discrete_scale(aesthetics = aesthetic, scale_name = "manual",
##     palette = pal)
```

```r
scale_fill_brewer(); scale_color_brewer()
```

```
## discrete_scale(aesthetics = "fill", scale_name = "brewer", palette = brewer_pal(type,
##     palette))
```

```
## discrete_scale(aesthetics = "colour", scale_name = "brewer",
##     palette = brewer_pal(type, palette))
```

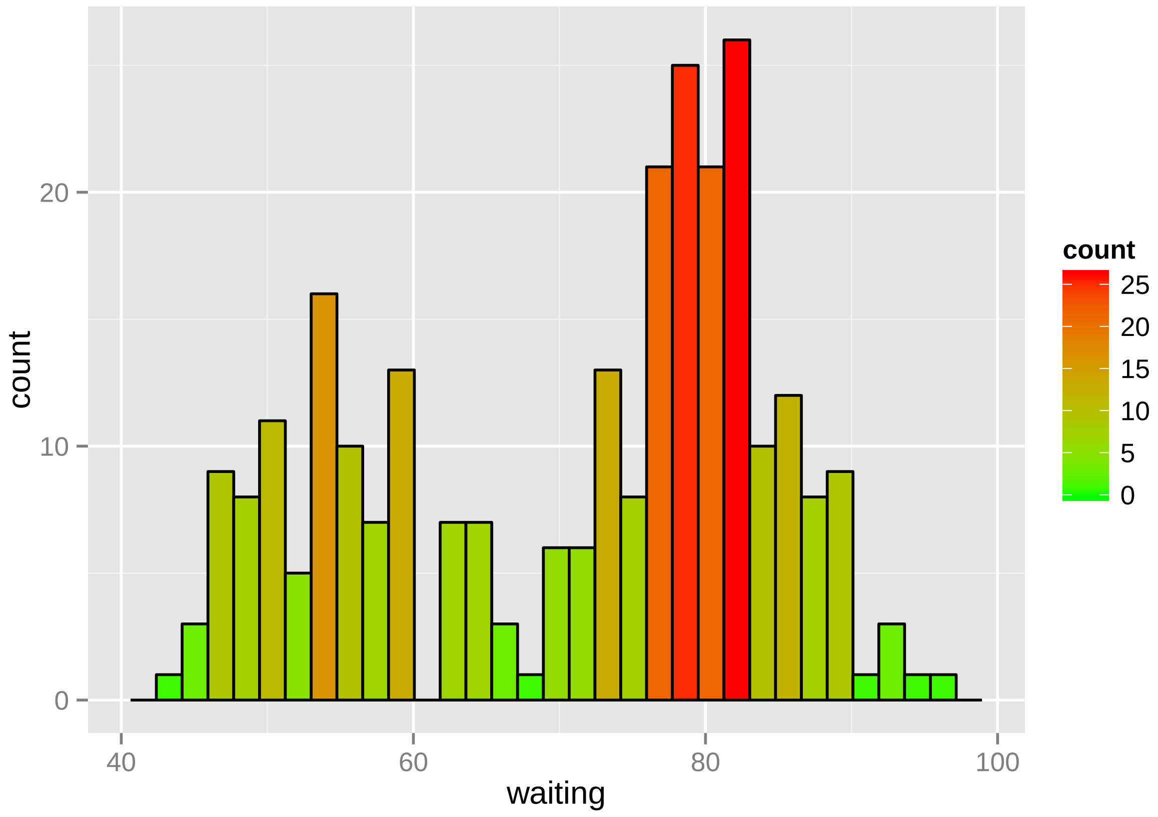
```r
scale_linetype(); scale_shape_manual()
```

```
## discrete_scale(aesthetics = "linetype", scale_name = "linetype_d",
##     palette = linetype_pal(), na.value = na.value)
```

```
## discrete_scale(aesthetics = aesthetic, scale_name = "manual",
##     palette = pal)
```

Gradients

```r
h + geom_histogram( aes(fill = ..count..), color="black") +
scale_fill_gradient(low="green", high="red")
```



Publication quality figures

If the plot is in your screen:

```
ggsave('filename.png')
```

```
## Saving 6.5 x 4.5 in image
```

If the plot is assigned to an object

```
plot1 <-ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point()
ggsave(plot1, file = "filename.png")
```

```
## Saving 6.5 x 4.5 in image
```

Specify a size

```
ggsave(file = "filename.png", width = 6, height =4) # default units in inches
```

Or any format (pdf, png, eps, svg, jpg)

```r
ggsave(file = "filename.eps")
```

```
## Saving 6.5 x 4.5 in image
```

```r
ggsave(file = "filename.jpg")
```

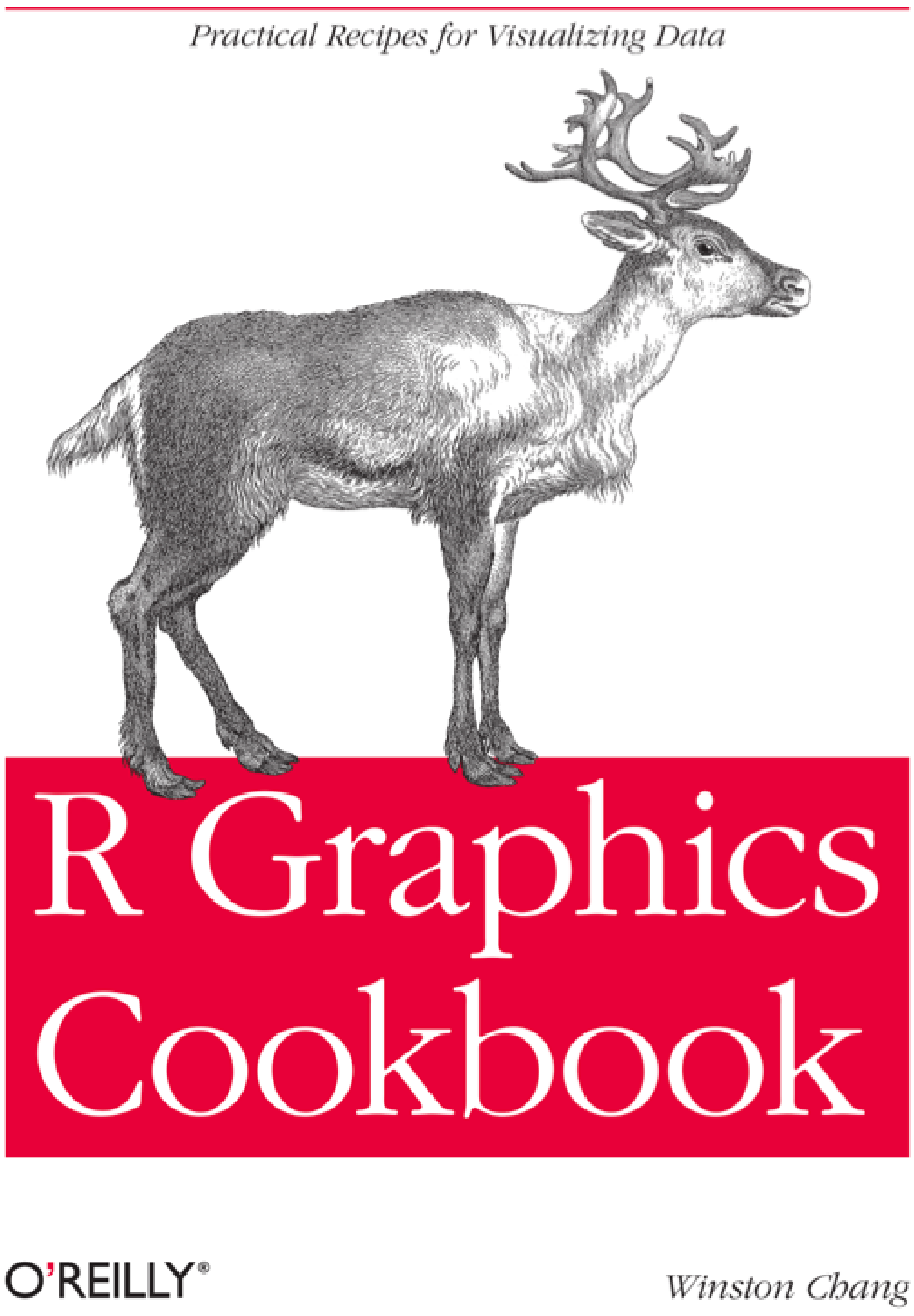
```
## Saving 6.5 x 4.5 in image
```

```r
ggsave(file = "filename.pdf") # preferred format for final publication images
```

```
## Saving 6.5 x 4.5 in image
```

Further help

- You've just scratched the surface with ggplot2.
- Practice
- Read the docs (either locally in `R` or at
- http://docs.ggplot2.org/current/
- Work together



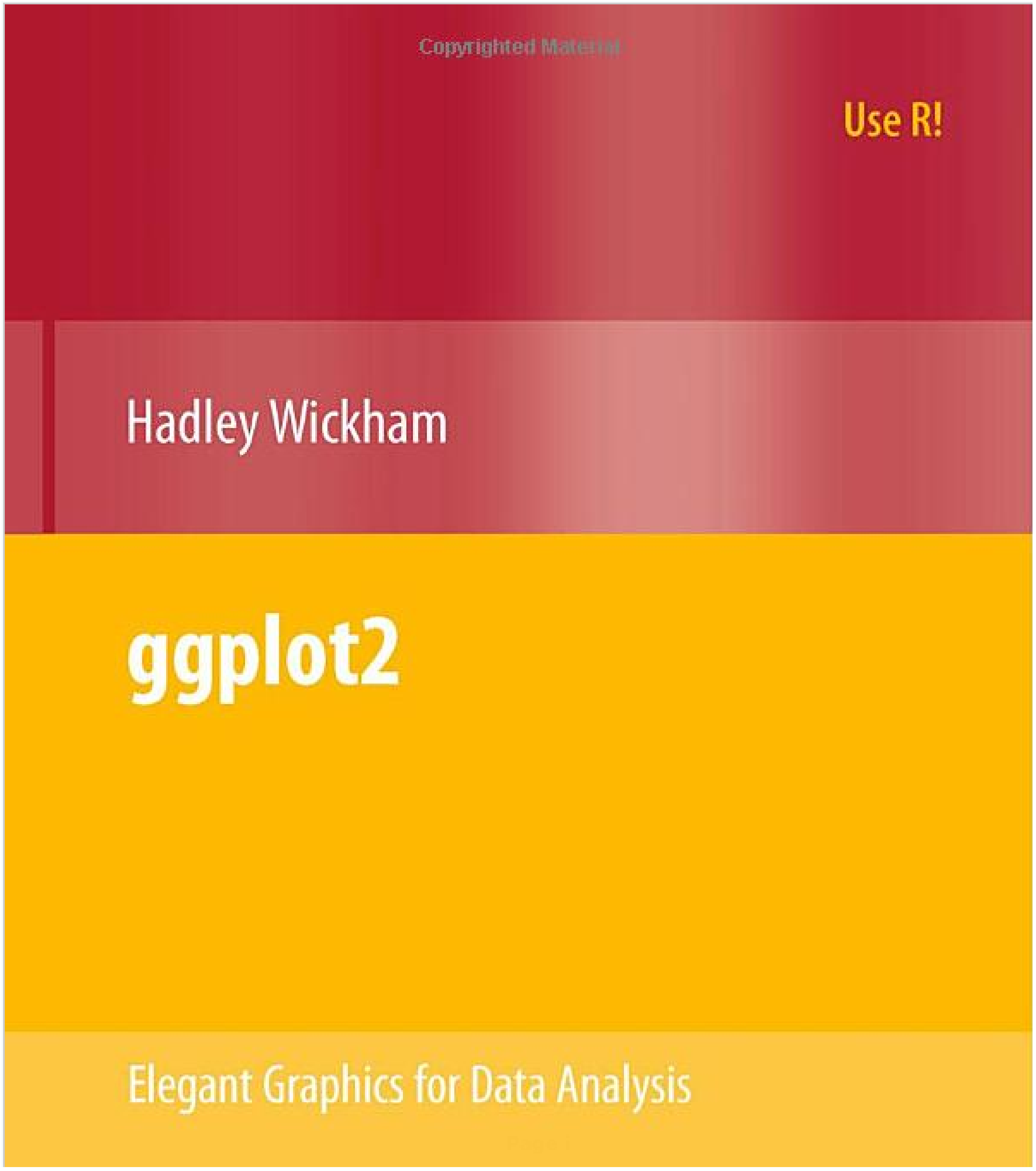
Practical Recipes for Visualizing Data

R Graphics Cookbook

O'REILLY®

Winston Chang

- stackedit


Copyrighted Material
Use R!
Hadley Wickham
ggplot2
Elegant Graphics for Data Analysis