

Data Visualization Using R & ggplot2

Naupaka Zimmerman (@naupakaz)
Andrew Tredennick (@ATredennick)

Hat tip to Karthik Ram (@inundata) for original slides

August 10, 2014

Some housekeeping

Install some packages

```
install.packages("ggplot2", dependencies = TRUE)
install.packages("plyr")
install.packages("ggthemes")
install.packages("reshape2")
# for some of the examples towards the end:
install.packages("gridExtra")
install.packages("devtools")
# Then a few packages to acquire data from the web to visualize
install.packages("rfisheries")
install.packages("rgbif")
install.packages("taxize")
# optional
install_github("rWBclimate", "ropensci")
```

Section 1

Why ggplot2?

Why ggplot2?

- ▶ More elegant & compact code than with base graphics
- ▶ More aesthetically pleasing defaults than lattice
- ▶ Very powerful for exploratory data analysis

Why ggplot2?

- ▶ 'gg' is for 'grammar of graphics' (term by Lee Wilkinson)
- ▶ A set of terms that defines the basic components of a plot
- ▶ Used to produce figures using coherent, consistent syntax

Why ggplot2?

- ▶ Supports a continuum of expertise:
- ▶ Easy to get started, plenty of power for complex figures

Section 2

The Grammar

Some terminology

- ▶ **data**
 - ▶ Must be a `data.frame`
 - ▶ Gets pulled into the `ggplot()` object

The iris dataset

```
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
```

plyr and reshape are key for using R

These two packages are the swiss army knives of R.

- ▶ plyr

1. ddply (data frame to data frame ply)

- 1.1 split

- 1.2 apply

- 1.3 combine

2. lply (list to list ply)

3. join

plyr

```
iris[1:2, ]  
  
##   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  
  
# Note the use of the . function to allow Species to be used  
# without quoting  
ddply(iris, .(Species), summarize,  
      mean.Sep.Wid = mean(Sepal.Width, na.rm = TRUE))  
  
##   Species mean.Sep.Wid  
## 1   setosa      3.428  
## 2 versicolor      2.770  
## 3 virginica      2.974
```

plyr and reshape are key for using R

These two packages are the swiss army knives of R.

- ▶ reshape

1. melt
2. dcast (data frame output)
3. acast (vector/matrix/array output)

reshape2

```
iris[1:2, ]
```

```
##   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
```

```
df <- melt(iris, id.vars = "Species")
df[1:2, ]
```

```
##   Species    variable value
## 1  setosa Sepal.Length  5.1
## 2  setosa Sepal.Length  4.9
```

reshape2

```
df[1:2, ]
```

```
##   Species      variable value
## 1  setosa Sepal.Length   5.1
## 2  setosa Sepal.Length   4.9
```

```
dcast(df, Species ~ variable, mean)
```

```
##      Species Sepal.Length Sepal.Width Petal.Length
## 1    setosa      5.006      3.428      1.462
## 2 versicolor      5.936      2.770      4.260
## 3 virginica      6.588      2.974      5.552
##   Petal.Width
## 1      0.246
## 2      1.326
## 3      2.026
```

Section 3

Aesthetics

Some terminology

- ▶ **data**
- ▶ **aesthetics**
- ▶ **How your data are represented visually**
 - ▶ *a.k.a. mapping*
 - ▶ which data on the x
 - ▶ which data on the y
 - ▶ but also: **color**, **size**, shape, transparency

Let's try an example

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

```
## data: Sepal.Length, Sepal.Width, Petal.Length,  
##   Petal.Width, Species [150x5]  
## mapping: x = Sepal.Length, y = Sepal.Width  
## faceting: facet_null()
```

Section 4

Geoms

Some terminology

- ▶ **data**
 - ▶ **aesthetics**
 - ▶ **geometry**
- ▶ **The geometric objects in the plot**
 - ▶ points, lines, polygons, etc
 - ▶ shortcut functions: `geom_point()`,
`geom_bar()`, `geom_line()`

Basic structure

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

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

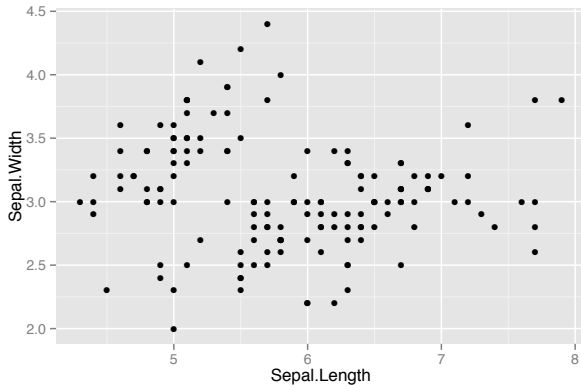
- ▶ Specify the data and variables inside the `ggplot` function.
- ▶ Anything else that goes in here becomes a global setting.
- ▶ Then add layers: geometric objects, statistical models, and facets.

Quick note

- ▶ Never use `qplot` - short for quick plot.
- ▶ You'll end up unlearning and relearning a good bit.

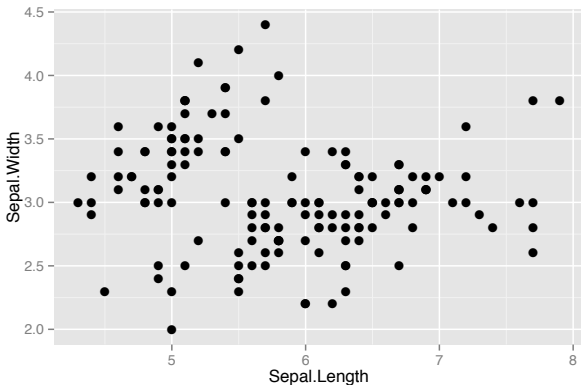
Let's try an example

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



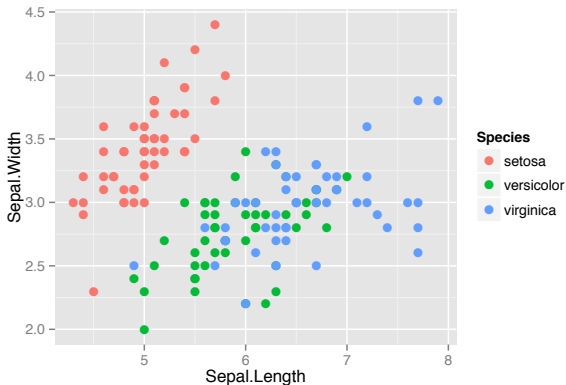
Changing the aesthetics of a geom: Increase the size of points

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



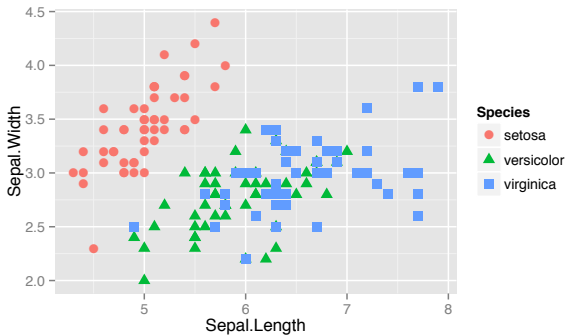
Changing the aesthetics of a geom: Add some color

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



Changing the aesthetics of a geom: Differentiate points by shape

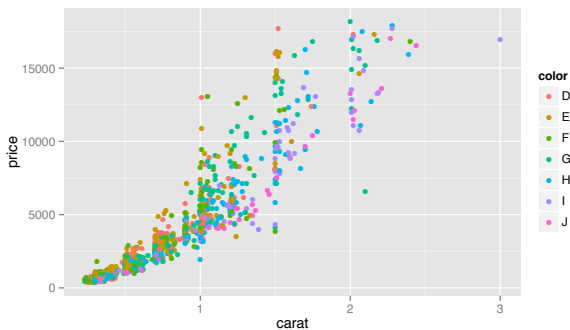
```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(aes(shape = Species), size = 3)  
# Why aes(shape = Species)?
```



Exercise 1

```
# Make a small sample of the diamonds dataset  
d2 <- diamonds[sample(1:dim(diamonds)[1], 1000), ]
```

Then generate this plot below.



Section 5

Stats

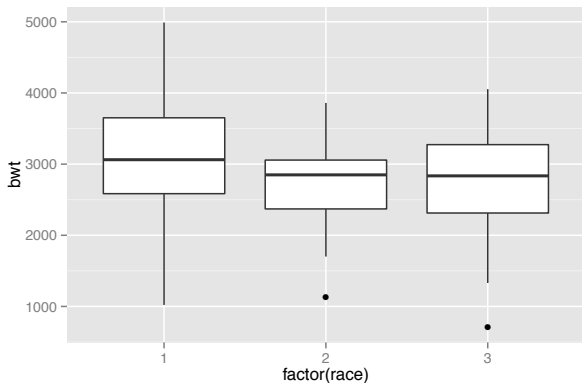
Some terminology

- ▶ data
 - ▶ aesthetics
 - ▶ geometry
 - ▶ stats
- ▶ **Statistical transformations and data summary**
 - ▶ All geoms have associated default stats, and vice versa
 - ▶ e.g. binning for a histogram or fitting a linear model

Built-in stat example: Boxplots

See `?geom_boxplot` for list of options

```
library(MASS)
ggplot(birthwgt, aes(factor(race), bwt)) + geom_boxplot()
```



Built-in stat example: Boxplots

```
myplot <- ggplot(birthwt, aes(factor(race), bwt)) + geom_boxplot()
summary(myplot)

## data: low, age, lwt, race, smoke, ptl, ht, ui, ftv,
##   bwt [189x10]
## mapping:  x = factor(race), y = bwt
## faceting: facet_null()
## -----
## geom_boxplot: outlier.colour = black, outlier.shape = 16, outlier.size =
## stat_boxplot:
## position_dodge: (width = NULL, height = NULL)
```

Section 6

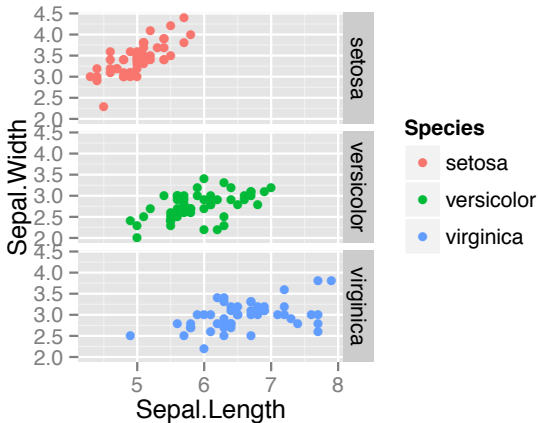
Facets

Some terminology

- ▶ data
 - ▶ aesthetics
 - ▶ geometry
 - ▶ stats
 - ▶ facets
- ▶ **Subsetting data to make lattice plots**
 - ▶ Really powerful

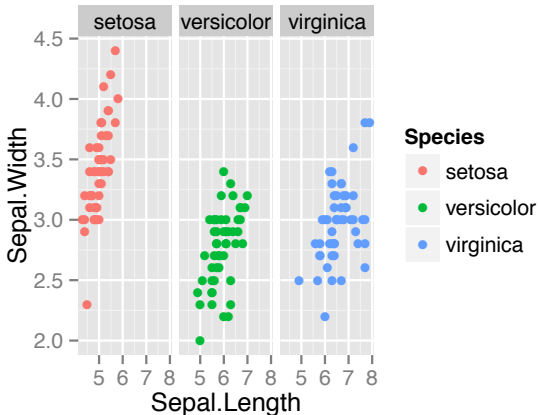
Faceting: single column, multiple rows

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



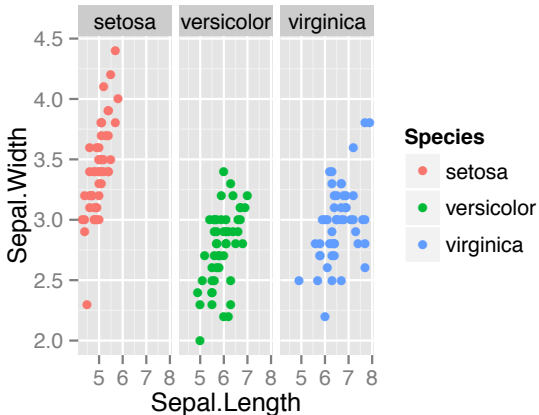
Faceting: single row, multiple columns

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



or just wrap your facets

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point() +  
  facet_wrap(~ Species) # notice lack of .
```



Section 7

Scales

Some terminology

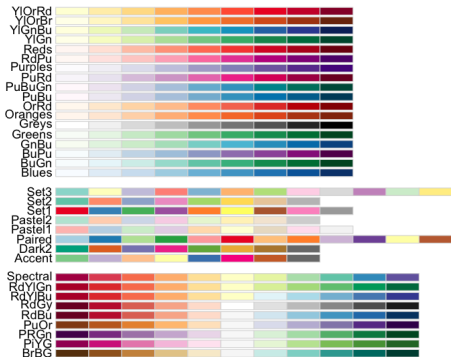
- ▶ data
 - ▶ aesthetics
 - ▶ geometry
 - ▶ stats
 - ▶ facets
 - ▶ scales
- ▶ **Control the mapping from data to aesthetics**
 - ▶ Often used for adjusting color mapping

Colors

```
aes(color = variable) # mapping  
color = "black" # setting  
  
# Or add it as a scale  
scale_fill_manual(values = c("color1", "color2"))
```

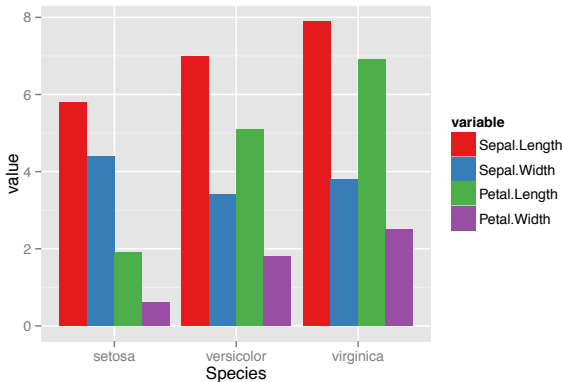
The RColorBrewer package

```
library(RColorBrewer)  
display.brewer.all()
```



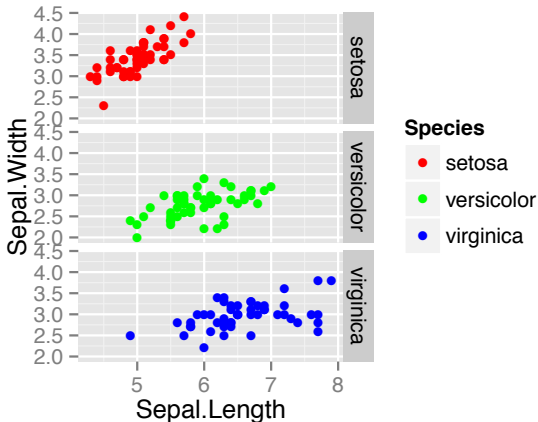
Using a color brewer palette

```
df <- melt(iris, id.vars = "Species")
ggplot(df, aes(Species, value, fill = variable)) +
  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() +  
  facet_grid(Species ~ .) +  
  scale_color_manual(values = c("red", "green", "blue"))
```



Refer to a color chart for beautiful visualizations

<http://tools.medialab.sciences-po.fr/iwanthue/>

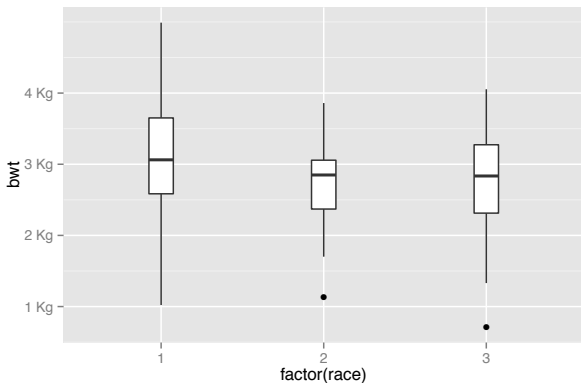
The screenshot shows the 'i want hue' web application interface. At the top, a dark navigation bar contains the text 'i want hue' and several menu items: 'Tutorials', 'Examples', 'Theory', 'Experiment', 'Old version', 'GitHub', and 'Issues'. On the right side of this bar is a '+ Medialab Tools' button. Below the navigation bar is the application's logo, which consists of a colorful starburst icon and the text 'i want hue'. To the right of the logo is the tagline: 'Colors for data scientists. Generate and refine palettes of optimally distinct colors.'

The main content area is divided into three sections:

- Color space:** This section contains three horizontal sliders for adjusting color parameters. The first slider is labeled 'Intense' and has a dropdown arrow. Below it are three sliders: 'H' (Hue) ranging from 0 to 360, 'C' (Chroma) ranging from 0 to 1, and 'L' (Luminance) ranging from 0.2 to 1.1. A checkbox labeled 'Dark background' is located below these sliders.
- Palette:** This section features a dropdown menu set to '5 colors' and another dropdown set to 'soft (k-Means)'. Below these is a button labeled 'Reroll palette'. Underneath, five colored squares are displayed: green, magenta, orange, teal, and purple.
- Visualizations:** In the center of the interface, there are five clusters of colored dots. Each cluster is a different color and shape, representing the output of the color generation process. One cluster is highlighted with a tooltip that says '@ 251700'.

Adding a continuous scale to an axis

```
library(MASS)
ggplot(birthwt, aes(factor(race), bwt)) +
  geom_boxplot(width = .2) +
  scale_y_continuous(labels = (paste0(1:4, " Kg")),
    breaks = seq(1000, 4000, by = 1000))
```



Commonly used scales

```
scale_fill_discrete(); scale_colour_discrete()  
scale_fill_hue(); scale_color_hue()  
scale_fill_manual(); scale_color_manual()  
scale_fill_brewer(); scale_color_brewer()  
scale_linetype(); scale_shape_manual()
```

Section 8

Coordinates

Some terminology

- ▶ **data**
 - ▶ **aesthetics**
 - ▶ **geometry**
 - ▶ **stats**
 - ▶ **facets**
 - ▶ **scales**
 - ▶ **coordinates**
- ▶ Not going to cover this in detail
 - ▶ e.g. polar coordinate plots

Section 9

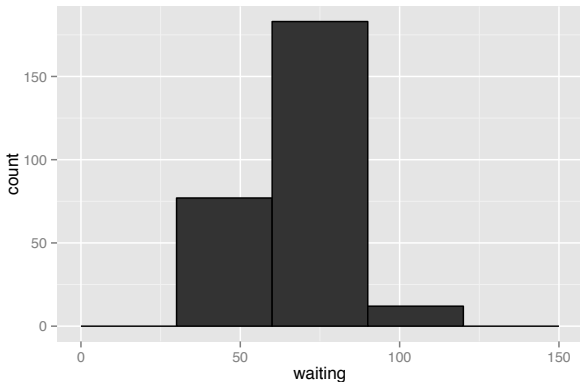
Putting it all together with more examples

Section 10

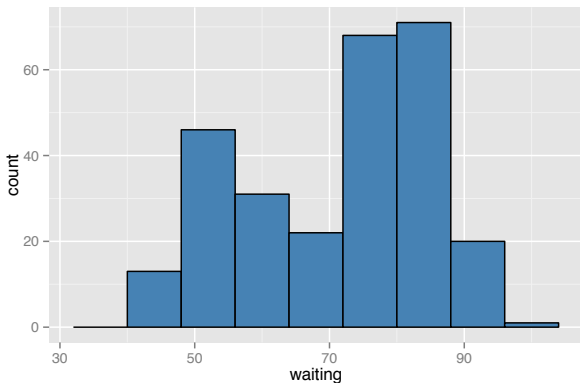
Histograms

See `?geom_histogram` for list of options

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



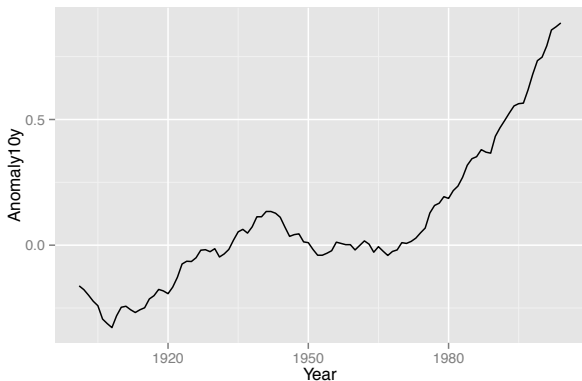
```
h <- ggplot(faithful, aes(x = waiting))  
h + geom_histogram(binwidth = 8, fill = "steelblue",  
colour = "black")
```



Section 11

Line plots

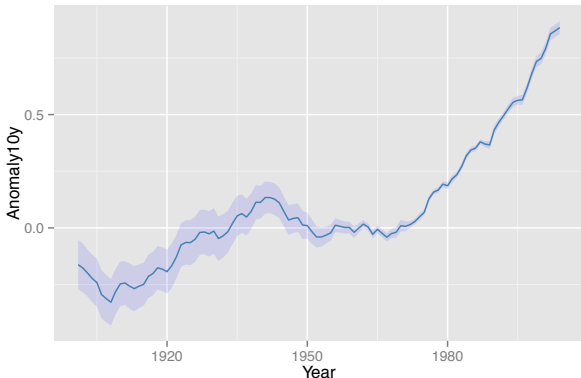
```
climate <- read.csv("../data/climate.csv", header = T)
ggplot(climate, aes(Year, Anomaly10y)) +
  geom_line()
```



```
climate <- read.csv(text =
RCurl::getURL(https://raw.githubusercontent.com/karthikram/ggplot-lecture/master/climate.csv))
```

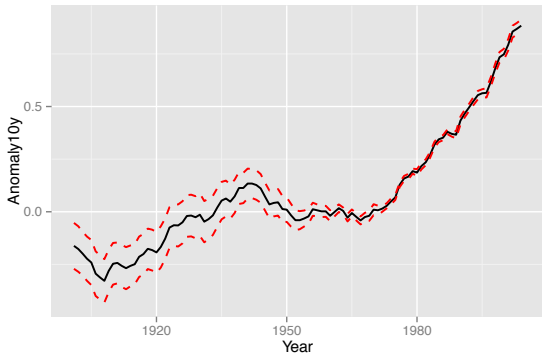
We can also plot confidence regions

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



Exercise 2

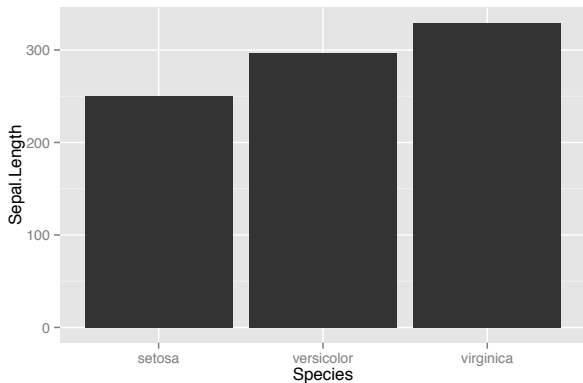
- ▶ Modify the previous plot and change it such that there are three lines instead of one with a confidence band.



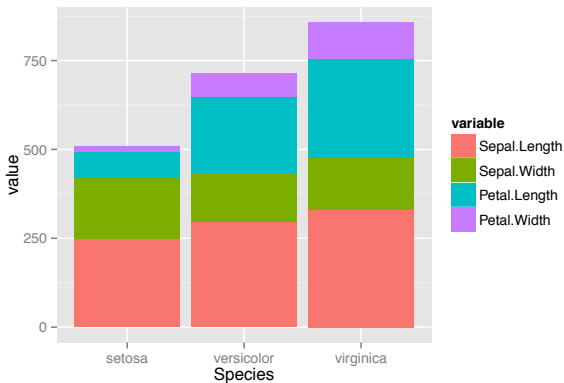
Section 12

Bar plots

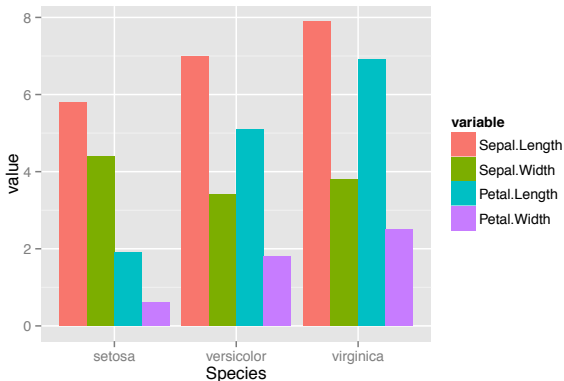
```
ggplot(iris, aes(Species, Sepal.Length)) +  
geom_bar(stat = "identity")
```




```
df <- melt(iris, id.vars = "Species")
ggplot(df, aes(Species, value, fill = variable)) +
  geom_bar(stat = "identity")
```

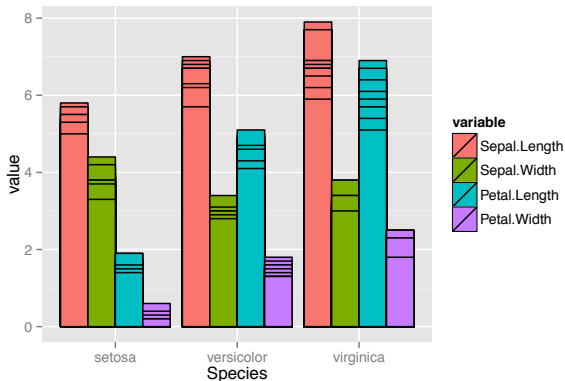


```
ggplot(df, aes(Species, value, fill = variable)) +  
  geom_bar(stat = "identity", position = "dodge")
```



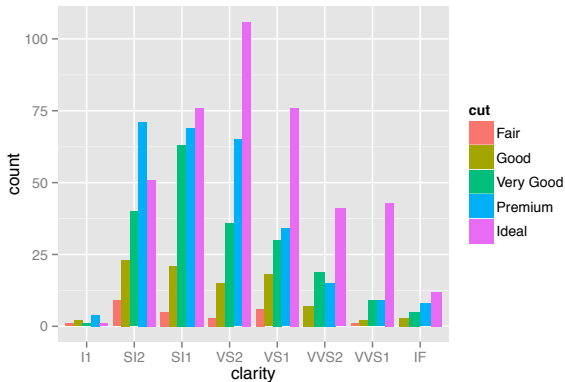
What's going on with the y axis?

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



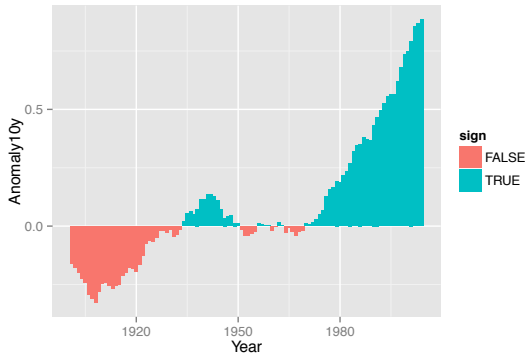
Exercise 3

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 4

- ▶ Using the climate dataset, create a new variable called `sign`. Make it logical (true/false) based on the sign of `Anomaly10y`.
- ▶ Plot a bar plot and use `sign` variable as the fill.

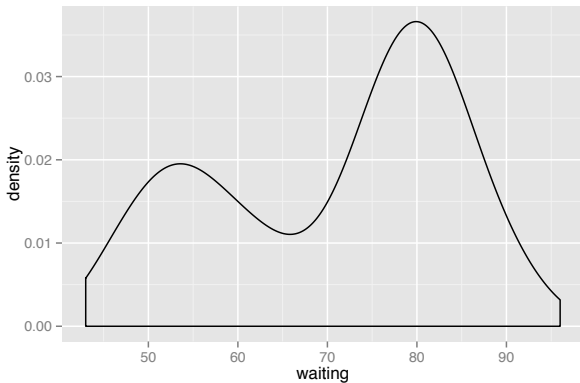


Section 13

Density Plots

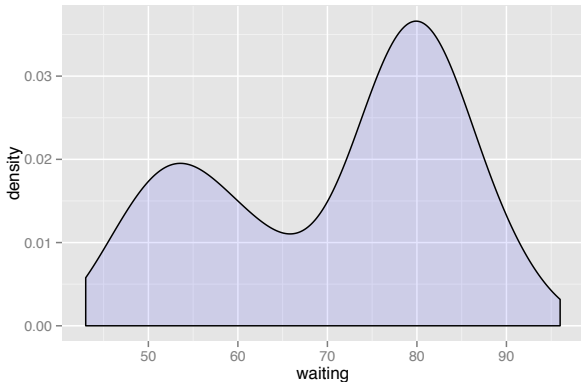
Density plots

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

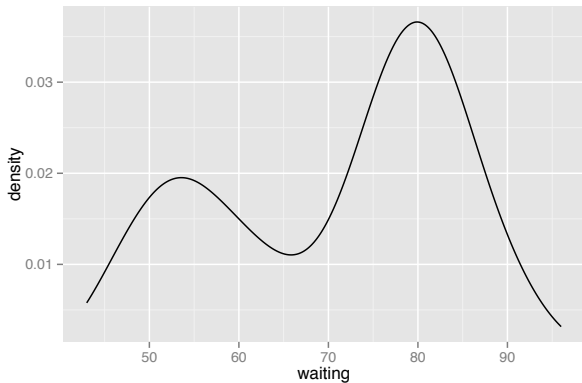


Density plots

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



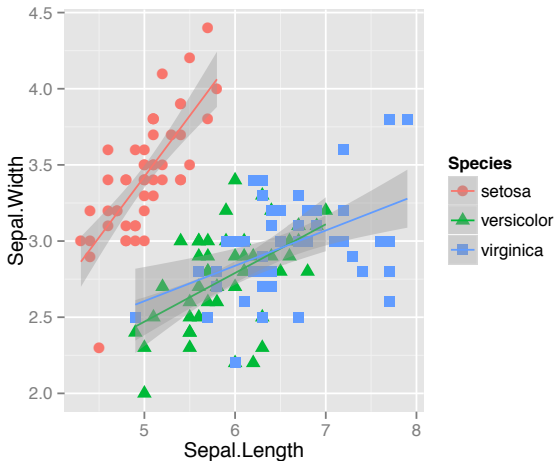

```
ggplot(faithful, aes(waiting)) +  
  geom_line(stat = "density")
```



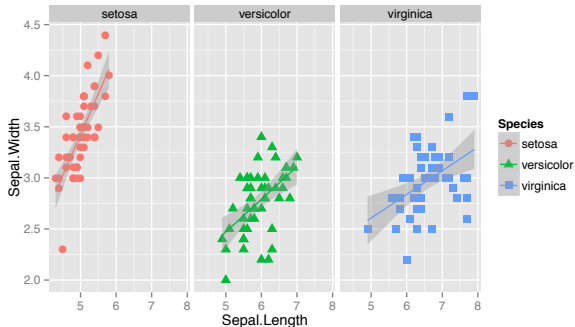
Section 14

Adding smoothers

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



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



Section 15

Themes

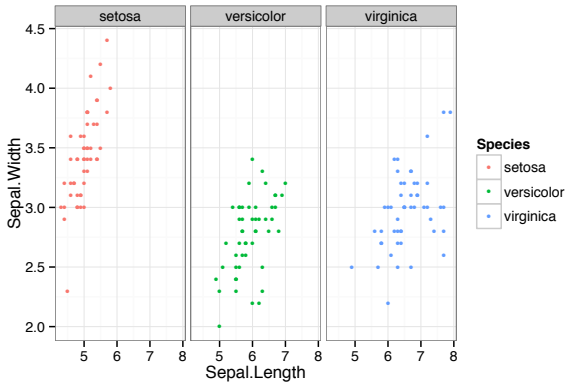
Adding themes

Themes are a great way to define custom plots.

```
+ theme()  
# see ?theme() for more options
```

A more basic theme

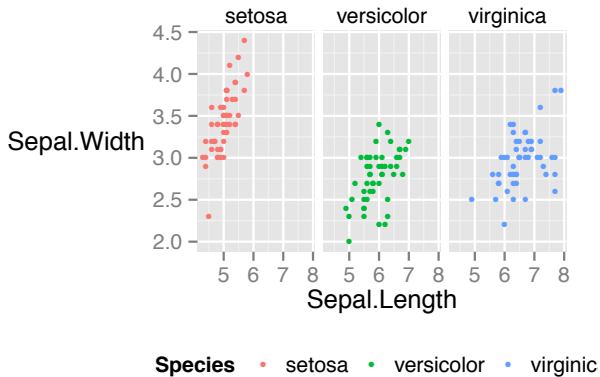
```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(size = 1.2, shape = 16) +  
  facet_wrap(~ Species) +  
  theme_bw()
```



A themed plot

```
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))
```


A themed plot



ggthemes library

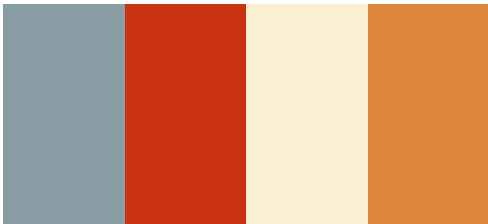
```
install.packages(ggthemes)
library(ggthemes)
# Then add one of these themes to your plot
+ theme_stata()
+ theme_excel()
+ theme_wsj()
+ theme_solarized()
```

Fan of Wes Anderson movies?



Yup, that's a thing

```
# install.packages(wesanderson)
library("wesanderson")
# display a palette
display.wes.palette(4, "Royal1")
```



Royal1

Section 16

Create functions to automate your plotting

Write functions for day to day plots

```
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")
```

Section 17

Publication quality figures

- ▶ If the plot is on your screen

```
ggsave("~/path/to/figure/filename.png")
```

- ▶ If your plot is assigned to an object

```
ggsave(plot1, file = "~/path/to/figure/filename.png")
```

- ▶ Specify a size

```
ggsave(file = "/path/to/figure/filename.png", width = 6,  
height = 4)
```

- ▶ or any format (pdf, png, eps, svg, jpg)

```
ggsave(file = "/path/to/figure/filename.eps")  
ggsave(file = "/path/to/figure/filename.jpg")  
ggsave(file = "/path/to/figure/filename.pdf")
```


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

