````
---
title: "Chapter 7: Data Visualization, ggplot"
author: "M Affouf"
date: "1/8/2018"
output:
  html_document:
    fig_height: 5
    fig_width: 5
    toc: yes
    toc_depth: 5
  pdf_document:
    toc: yes
# output:
#   beamer_presentation: default
#   ioslides_presentation:
#     css: ../../styles.css
---

#part1


```{r knit-setup, echo=FALSE}
# library(knitr)
# opts_chunk$set(echo = TRUE,
#                message = FALSE,
#                warning = FALSE,
#                fig.height = 3.5,
#                fig.width = 3.5,
#                comment = "")
```

## Basic Plots

We covered some basic plots previously, but we are going to expand the ability to customize these basic graphics first.

```{r seed, comment="",echo=FALSE,prompt=TRUE, }
set.seed(3)
```



## Read in Data

```{r plotEx,  fig.align='center',cache=FALSE}
death = read.csv("indicatordeadkids35.csv",
                 as.is=TRUE,header=TRUE, row.names=1)
death[1:2, 1:5]
```

We see that the column names were years, and R doesn't necessarily like to read in a column name that starts with a number and puts an X there. We'll just take off that X and get the years.

```{r}
library(stringr)
year = names(death) %>% str_replace("X","") %>% as.integer
````
````

```
head(year)
```

Basic Plots

```{r plot1, comment="",prompt=TRUE,  fig.align='center',cache=TRUE}
plot(as.numeric(death["Sweden",]) ~ year)
```

Basic Plots

The y-axis label isn't informative, and we can change the label of the y-axis using `ylab` (`xlab` for x), and `main` for the main title/label.

```{r plotEx2, comment="",prompt=TRUE,  fig.align='center',cache=TRUE}
plot(as.numeric(death["Sweden",]) ~ year,
      ylab = "# of deaths per family", main = "Sweden")
```

Basic Plots

Let's drop any of the projections and keep it to year 2012, and change the points to blue.

```{r plotEx3, fig.align='center', cache=TRUE}
plot(as.numeric(death["Sweden",])~year,
      ylab = "# of deaths per family", main = "Sweden",
     xlim = c(1760,2012), pch = 19, cex=1.2,col="blue")
```

Basic Plots

You can also use the `subset` argument in the `plot()` function, only when using formula notation:

```{r plotEx_sub, fig.align='center', cache=TRUE}
plot(as.numeric(death["Sweden",])~year,
      ylab = "# of deaths per family", main = "Sweden",
     subset = year < 2015, pch = 19, cex=1.2,col="blue")
```

Basic Plots

Using `scatter.smooth` plots the points and runs a loess smoother through the data.

```{r plotEx4, comment="",prompt=TRUE, fig.align='center', cache=TRUE}
scatter.smooth(as.numeric(death["Sweden",])~year,span=0.2,
      ylab="# of deaths per family", main = "Sweden",lwd=3,
     subset = year < 2015, pch = 19, cex=0.9,col="grey")
```

Basic Plots {.smaller}

`par(mfrow=c(1,2))` tells R that we want to set a parameter (`par` function) named `mfrow` (number of plots - 1 row, 2 columns) so we can have 2 plots side by side (Sweden and the UK)

```{r plotEx5, comment="",prompt=TRUE,
fig.width=8,fig.height=4,fig.align='center', cache=TRUE}
par(mfrow=c(1,2))
scatter.smooth(as.numeric(death["Sweden",])~year,span=0.2,
      ylab="# of deaths per family", main = "Sweden",lwd=3,
     xlim = c(1760,2012), pch = 19, cex=0.9,col="grey")
scatter.smooth(as.numeric(death["United Kingdom",])~year,span=0.2,
      ylab="# of deaths per family", main = "United Kingdom",lwd=3,
     xlim = c(1760,2012), pch = 19, cex=0.9,col="grey")
```

Basic Plots {.smaller}
We can set the y-axis to be the same.
```{r plotEx6, fig.width=8,fig.height=4,fig.align='center', cache=TRUE}
par(mfrow=c(1,2))
yl = range(death[c("Sweden","United Kingdom"),])
scatter.smooth(as.numeric(death["Sweden",])~year,span=0.2,ylim=yl,
      ylab="# of deaths per family", main = "Sweden",lwd=3,
     xlim = c(1760,2012), pch = 19, cex=0.9,col="grey")
scatter.smooth(as.numeric(death["United Kingdom",])~year,span=0.2,
      ylab="", main = "United Kingdom",lwd=3,ylim=yl,
     xlim = c(1760,2012), pch = 19, cex=0.9,col="grey")
```

Bar Plots
* Stacked Bar Charts are sometimes wanted to show distributions of data

```{r barplot2, fig.align='center', cache=TRUE}
## Stacked Bar Charts
cars = read.csv("kaggleCarAuction.csv",as.is=T)
counts <- table(cars$IsBadBuy, cars$VehicleAge)
barplot(counts, main="Car Distribution by Age and Bad Buy Status",
  xlab="Vehicle Age", col=c("darkblue","red"),
    legend = rownames(counts))
```

Bar Plots
`prop.table` allows you to convert a table to proportions (depends on margin - either row percent or column percent)

```{r barplot2a, fig.align='center', cache=TRUE}
## Use percentages (column percentages)
barplot(prop.table(counts, 2), main="Car Distribution by Age and Bad Buy Status",
  xlab="Vehicle Age", col=c("darkblue","red"),
    legend = rownames(counts))
```

Bar Plots

Using the `beside` argument in `barplot`, you can get side-by-side barplots.
```{r barplot3, fig.align='center', cache=TRUE}
# Stacked Bar Plot with Colors and Legend
barplot(counts, main="Car Distribution by Age and Bad Buy Status",
```

```
  xlab="Vehicle Age", col=c("darkblue","red"),
    legend = rownames(counts), beside=TRUE)
```

Graphics parameters

Set within most plots in the base 'graphics' package:

* pch = point shape, http://voteview.com/symbols_pch.htm
* cex = size/scale
* xlab, ylab = labels for x and y axes
* main = plot title
* lwd = line density
* col = color
* cex.axis, cex.lab, cex.main = scaling/sizing for axes marks, axes labels, and title

Devices

By default, R displays plots in a separate panel. From there, you can export the plot to a variety of image file types, or copy it to the clipboard.

However, sometimes its very nice to save many plots made at one time to one pdf file, say, for flipping through. Or being more precise with the plot size in the saved file.

R has 5 additional graphics devices: bmp(), jpeg(), png(), tiff(), and pdf()

Devices

The syntax is very similar for all of them:

```
pdf("filename.pdf", width=8, height=8) # inches
plot() # plot 1
plot() # plot 2
# etc
dev.off()
```

Basically, you are creating a pdf file, and telling R to write any subsequent plots to that file. Once you are done, you turn the device off. Note that failing to turn the device off will create a pdf file that is corrupt, that you cannot open.

Boxplots, revisited

These are one of my favorite plots. They are way more informative than the barchart + antenna...

```{r boxplots, comment="",prompt=TRUE, fig.align='center', cache=FALSE}
boxplot(weight ~ Diet, data=ChickWeight, outline=FALSE)
points(ChickWeight$weight ~ jitter(as.numeric(ChickWeight$Diet),0.5))
```

Formulas

Formulas have the format of `y ~ x` and functions taking formulas have a `data` argument where you pass the data.frame. You don't need to use `$` or referencing when using formulas:

```{r box_ex, eval=FALSE}
boxplot(weight ~ Diet, data=ChickWeight, outline=FALSE)
```

Colors

R relies on color 'palettes'.

```{r pal, fig.align='center', cache=TRUE}
palette("default")
plot(1:8, 1:8, type="n")
text(1:8, 1:8, lab = palette(), col = 1:8)
```

Colors

The default color palette is pretty bad, so you can try to make your own.

```{r pal2, fig.align='center', cache=TRUE}
palette(c("darkred","orange","blue"))
plot(1:3,1:3,col=1:3,pch =19,cex=2)
```

Colors

It's actually pretty hard to make a good color palette. Luckily, smart and artistic people have spent a lot more time thinking about this. The result is the 'RColorBrewer' package

RColorBrewer::display.brewer.all() will show you all of the palettes available. You can even print it out and keep it next to your monitor for reference.

The help file for brewer.pal() gives you an idea how to use the package.

You can also get a "sneak peek" of these palettes at: www.colorbrewer2.com . You would provide the number of levels or classes of your data, and then the type of data: sequential, diverging, or qualitative. The names of the RColorBrewer palettes are the string after 'pick a color scheme:'

Colors

```{r pal3, fig.align='center', cache=FALSE}
palette("default")
plot(weight ~ Time, data= ChickWeight, pch = 19, col = Diet)
```

Colors

```{r pal4, fig.align='center', cache=FALSE}
library(RColorBrewer)
palette(brewer.pal(5,"Dark2"))
plot(weight ~ Time, data=ChickWeight, pch = 19,  col = Diet)
```

Colors

```{r pal5, fig.align='center', cache=FALSE}
library(RColorBrewer)
palette(brewer.pal(5,"Dark2"))
plot(weight ~ jitter(Time,amount=0.2),data=ChickWeight,
     pch = 19,  col = Diet,xlab="Time")
```

Adding legends

The legend() command adds a legend to your plot. There are tons of arguments to pass it.

x, y=NULL: this just means you can give (x,y) coordinates, or more commonly just give x, as a character string:
"top","bottom","topleft","bottomleft","topright","bottomright".

legend: unique character vector, the levels of a factor

pch, lwd: if you want points in the legend, give a pch value. if you want lines, give a lwd value.

col: give the color for each legend level

Adding legends

```{r leg1, fig.align='center', cache=FALSE}
palette(brewer.pal(5,"Dark2"))
plot(weight ~ jitter(Time,amount=0.2),data=ChickWeight,
                pch = 19,  col = Diet,xlab="Time")
legend("topleft", paste("Diet",levels(ChickWeight$Diet)),
       col = 1:length(levels(ChickWeight$Diet)),
       lwd = 3, ncol = 2)
```

Coloring by variable

```{r circ, comment="",prompt=TRUE, fig.align='center', cache=FALSE}
circ = read.csv("Charm_City_Circulator_Ridership.csv",
           header=TRUE,as.is=TRUE)
palette(brewer.pal(7,"Dark2"))
```

```
dd = factor(circ$day)
plot(orangeAverage ~ greenAverage, data=circ,
     pch=19, col = as.numeric(dd))
legend("bottomright", levels(dd), col=1:length(dd), pch = 19)
```

Coloring by variable

```{r circ2, comment="",prompt=TRUE, fig.align='center', cache=FALSE}
dd = factor(circ$day, levels=c("Monday","Tuesday","Wednesday",
            "Thursday","Friday","Saturday","Sunday"))
plot(orangeAverage ~ greenAverage, data=circ,
     pch=19, col = as.numeric(dd))
legend("bottomright", levels(dd), col=1:length(dd), pch = 19)
```

ggplot2

`ggplot2` is a package of plotting that is very popular and powerful. `qplot` is a short hand for "quick plot". We can simply do a boxplot:

```{r geoboxplot, comment="",prompt=TRUE, fig.align='center', cache=FALSE}
library(ggplot2)
qplot(factor(Diet), y = weight,
      data = ChickWeight, geom = "boxplot")
```

ggplot2

The generic plotting function is `ggplot`:

```{r geoboxplot_g, comment="",prompt=TRUE, fig.align='center',
cache=FALSE}
g = ggplot(aes(x = Diet, y = weight), data = ChickWeight)
g + geom_boxplot()
```

Boxplots revisited again

We can do the same plot, by just saying we want a boxplot and points (and jitter the points)

```{r geoboxpoint, comment="",prompt=TRUE, fig.align='center', cache=TRUE}
qplot( factor(Diet), y = weight, data = ChickWeight,
       geom = c("boxplot", "jitter"))
```

ggplot2: Adding 2 geoms together

To have multiple geometrics, just "add" them

```{r geoboxplot_add, comment="",prompt=TRUE, fig.align='center',
cache=FALSE}
g + geom_boxplot() + geom_point(position = "jitter")
```

```

## ggplot2: Adding 2 geoms together

To have multiple geometrics, just "add" them

```{r geoboxplot_addjitter, fig.align='center', cache=FALSE}
g + geom_boxplot() + geom_jitter()
```

Histograms again

We can do histograms again using `hist`. Let's do histograms of weight at all time points for the chick's weights. We reiterate how useful these are to show your data.

```{r hist, comment="",prompt=TRUE, fig.align='center', cache=FALSE}
hist(ChickWeight$weight, breaks = 20)
```

Multiple Histograms

```{r ghist, comment="",prompt=TRUE, fig.align='center', cache=TRUE}
qplot(x = weight,
      fill = factor(Diet),
      data = ChickWeight,
      geom = c("histogram"))
```

Multiple Histograms

Alpha refers tot he opacity of the color, less is
```{r ghist_alpha, comment="",prompt=TRUE, fig.align='center', cache=TRUE}
qplot(x = weight, fill = Diet, data = ChickWeight,
      geom = c("histogram"), alpha=I(.7))
```

Multiple Densities

We cold also do densities
```{r gdens, comment="",prompt=TRUE, fig.align='center', cache=TRUE}
qplot(x= weight, fill = Diet, data = ChickWeight,
      geom = c("density"), alpha=I(.7))
```

Multiple Densities

```{r gdens_alpha, comment="",prompt=TRUE, fig.align='center', cache=TRUE}
qplot(x= weight, colour = Diet, data = ChickWeight,
      geom = c("density"), alpha=I(.7))
```

Multiple Densities
```

```{r gdens_alpha_gg, comment="",prompt=TRUE, fig.align='center',
cache=TRUE}
ggplot(aes(x= weight, colour = Diet),
  data = ChickWeight) + geom_density(alpha=I(.7))
```

Multiple Densities

You can take off the lines of the bottom like this
```{r gdens_line_alpha, comment="",prompt=TRUE, fig.align='center',
cache=TRUE}
ggplot(aes(x = weight, colour = Diet), data = ChickWeight) +
  geom_line(stat = "density")
```

Spaghetti plot

We can make a spaghetti plot by telling ggplot we want a "line", and each line is colored by Chick.

```{r spaghetti, comment="",prompt=TRUE, fig.align='center', cache=FALSE}
qplot(x=Time, y=weight, colour = Chick,
      data = ChickWeight, geom = "line")
```

Spaghetti plot: Facets

In ggplot2, if you want separate plots for something, these are referred to as facets.

```{r fac_spag, comment="",prompt=TRUE, fig.align='center', cache=FALSE}
qplot(x = Time, y = weight, colour = Chick,
      facets = ~Diet, data = ChickWeight, geom = "line")
```

Spaghetti plot: Facets

We can turn off the legend (referred to a "guide" in ggplot2). (Note - there is different syntax with the `+`)
```{r fac_spag_noleg, comment="",prompt=TRUE, fig.align='center',
cache=FALSE}
qplot(x=Time, y=weight, colour = Chick,
      facets = ~ Diet,  data = ChickWeight,
        geom = "line") + guides(colour=FALSE)
```

Spaghetti plot: Facets

```{r fac_spag2, comment="",prompt=TRUE, fig.align='center', cache=FALSE}
ggplot(aes(x = Time, y = weight, colour = Chick),
    data = ChickWeight) + geom_line() +
    facet_wrap(facets = ~Diet) + guides(colour = FALSE)
```

ggplot2

Let's try this out on the childhood mortality data used above. However, let's do some manipulation first, by using `gather` on the data to convert to long.

```{r}
library(tidyr)
long = death
long$state = rownames(long)
long = long %>% gather(year, deaths, -state)
head(long, 2)
```

ggplot2

Let's also make the year numeric, as we did above in the stand-alone `year` variable.

```{r}
library(stringr)
library(dplyr)
long$year = long$year %>% str_replace("^X", "") %>% as.numeric
long = long %>% filter(!is.na(deaths))
```

ggplot2

```{r geom_line, comment="",prompt=TRUE, fig.align='center', cache=FALSE}
qplot(x = year, y = deaths, colour = state,
    data = long, geom = "line") + guides(colour = FALSE)
```

ggplot2

Let's try to make it different like base R, a bit. We use `tile` for the geometric unit:

```{r geom_tile}
qplot(x = year, y = state, colour = deaths,
    data = long, geom = "tile") + guides(colour = FALSE)
```

ggplot2

Useful links:

* http://docs.ggplot2.org/0.9.3/index.html
* http://www.cookbook-r.com/Graphs/

#Part 2
##ggplot2

##Agenda

This lecture continues our introduction to the `ggplot2` package developed by Hadley Wickham.

Let's begin by loading the packages we'll use this class

```{r}
library(ggplot2)
library(MASS)  # Useful for data sets
library(plyr)  # We'll need mapvalues
```

Review of ggplot2 basics

ggplot2 has a slightly steeper learning curve than the base graphics functions, but it also generally produces far better and more easily customizable graphics.

There are two basic calls in ggplot:

- `qplot(x, y, ..., data)`: a "quick-plot" routine, which essentially replaces the base `plot()`
- `ggplot(data, aes(x, y, ...), ...)`: defines a graphics object from which plots can be generated, along with *aesthetic mappings* that specify how variables are mapped to visual properties.

ggplot function

The `ggplot2` library comes with a dataset called `diamonds`. Let's look at it

```{r}
dim(diamonds)
head(diamonds)
```

It is a data frame of 53,940 diamonds, recording their attributes such as carat, cut, color, clarity, and price.

We will make a scatterplot showing the price as a function of the carat (size). (The data set is large so the plot may take a few moments to generate.)

```{r fig.width=10, fig.height=4, dpi=70, cache=TRUE}
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price))
diamond.plot + geom_point()
```

The data set looks a little weird because a lot of diamonds are concentrated on the 1, 1.5 and 2 carat mark.

Let's take a step back and try to understand the ggplot syntax.

1) The first thing we did was to define a graphics object, `diamond.plot`. This definition told R that we're using the `diamonds` data, and that we want to display `carat` on the x-axis, and `price` on the y-axis.

2) We then called `diamond.plot + geom_point()` to get a scatterplot.

The arguments passed to `aes()` are called **mappings**. Mappings specify what variables are used for what purpose. When you use `geom_point()` in the second line, it pulls `x`, `y`, `colour`, `size`, etc., from the **mappings** specified in the `ggplot()` command.

You can also specify some arguments to `geom_point` directly if you want to specify them for each plot separately instead of pre-specifying a default.

Here we shrink the points to a smaller size, and use the `alpha` argument to make the points transparent.

```{r fig.width=10, fig.height=4, dpi=70, cache=TRUE}
diamond.plot + geom_point(size = 0.7, alpha = 0.3)
```

If we wanted to let point color depend on the color indicator of the diamond, we could do so in the following way.

```{r fig.width=10, fig.height=6, dpi=70, cache=TRUE}
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))
diamond.plot + geom_point()
```

If we didn't know anything about diamonds going in, this plot would indicate to us that **D** is likely the highest diamond grade, while **J** is the lowest grade.

We can change colors by specifying a different color palette. Here's how we can switch to the `cbPalette` we saw last class.

```{r fig.width=10, fig.height=6, dpi=70, cache=TRUE}
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))
diamond.plot + geom_point() + scale_colour_manual(values=cbPalette)
```

To make the scatterplot look more typical, we can switch to logarithmic coordinate axis spacing.

```{r}
diamond.plot + geom_point() +
  coord_trans(x = "log10", y = "log10")
```

Just like in lattice, we can create plots showing the relationship between variables across different values of a factor. For instance, here's a scatterplot showing how diamond price varies with carat size, conditioned on color. It's created using the `facet_wrap(~ factor1 + factor2 + ... + factorn)` command.

```{r, fig.width=12, fig.height=6, dpi=70, cache=TRUE}
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour =
```

color))

```
diamond.plot + geom_point() + facet_wrap(~ cut) +
  coord_trans(x = "log10", y = "log10")
```

You can also use `facet_grid()` to produce this type of output.

```{r, fig.width=11, fig.height=4.5, dpi=70, cache=TRUE}
diamond.plot + geom_point() + facet_grid(. ~ cut) +
  coord_trans(x = "log10", y = "log10")
```

```{r, fig.width = 8, fig.height = 10, dpi = 70, cache = TRUE}
diamond.plot + geom_point() + facet_grid(cut ~ .) +
  coord_trans(x = "log10", y = "log10")
```

`ggplot` can create a lot of different kinds of plots, just like lattice. Here are some examples.

Function	Description
`geom_point(...)`	Points, i.e., scatterplot
`geom_bar(...)`	Bar chart
`geom_line(...)`	Line chart
`geom_boxplot(...)`	Boxplot
`geom_violin(...)`	Violin plot
`geom_density(...)`	Density plot with one variable
`geom_density2d(...)`	Density plot with two variables
`geom_histogram(...)`	Histogram

A barchart example

We'll use the `birthwt` data for this example, so let's start by loading an cleaning it. All of this code is borrowed from Lecture 5.

```{r}
# Assign better variable names
colnames(birthwt) <- c("birthwt.below.2500", "mother.age",
"mother.weight",
    "race", "mother.smokes", "previous.prem.labor", "hypertension",
"uterine.irr",
    "physician.visits", "birthwt.grams")

# Assign better labels to categorical variables
birthwt <- transform(birthwt,
            race = as.factor(mapvalues(race, c(1, 2, 3),
                              c("white","black", "other"))),
            mother.smokes = as.factor(mapvalues(mother.smokes,
                              c(0,1), c("no", "yes"))),
            hypertension = as.factor(mapvalues(hypertension,
                              c(0,1), c("no", "yes"))),
            uterine.irr = as.factor(mapvalues(uterine.irr,
                              c(0,1), c("no", "yes"))),
            birthwt.below.2500 = as.factor(mapvalues(birthwt.below.2500,
```

```
                                c(0,1), c("no", "yes")))
            )

```


A bar chart

```{r}
qplot(x = race, data = birthwt, geom = "bar")
```


Histograms and density plots

```{r}
base.plot <- ggplot(birthwt, aes(x = mother.age)) +
  xlab("Mother's age")
base.plot + geom_histogram()
base.plot + geom_histogram(aes(fill = race))
base.plot + geom_density()
base.plot + geom_density(aes(fill = race), alpha = 0.5)
```

Box plots and violin plots

```{r}
base.plot <- ggplot(birthwt, aes(x = as.factor(physician.visits), y =
birthwt.grams)) +
  xlab("Number of first trimester physician visits") +
  ylab("Baby's birthweight (grams)")

# Box plot
base.plot + geom_boxplot()

# Violin plot
base.plot + geom_violin()
```

Visualizing means

Previously we calculated the following table:
```{r}
bwt.summary <- aggregate(birthwt.grams ~ race + mother.smokes, data =
birthwt, FUN = mean) # aggregate
bwt.summary
```

We can plot this table in a nice bar chart as follows:

```{r, fig.width = 6}
# Define basic aesthetic parameters
p.bwt <- ggplot(data = bwt.summary, aes(y = birthwt.grams, x = race, fill
= mother.smokes))

# Pick colors for the bars
```

```
bwt.colors <- c("#009E73", "#999999")

# Display barchart
p.bwt + geom_bar(stat = "identity", position = "dodge") +
  ylab("Average birthweight") +
  xlab("Mother's race") +
  guides(fill = guide_legend(title = "Mother's smoking status")) +
  scale_fill_manual(values=bwt.colors)
```


Statistical overlays in ggplot

One of the main advantages of `ggplot` is that it makes it really easy to overlay model fits, confidence bars, etc. We'll get more practice with this next week, when we talk more about how to do statistical inference in R.

For the time being, let's just display a scatterplot smoother.

```{r fig.width=12, fig.height=6, dpi=70, cache=TRUE}
diamond.plot + stat_smooth()
```

This shows curves modelling the relationship between diamond size in carats and price for the different diamond colours.

We could use the `geom_point()` command to also display the underlying points, but that would make the curves difficult to see. Let's try it anyway.

```{r fig.width=12, fig.height=6, dpi=70, cache=TRUE}
diamond.plot + geom_point() + stat_smooth()
```

We can make this look better by decreasing the point opacity so that the trend curves aren't so obscured.

```{r fig.width=12, fig.height=6, dpi=70, cache=TRUE}
diamond.plot + geom_point(size = 1.5, alpha = 0.25) + stat_smooth()
```


Birthweight regression overlay

In addition to the more flexible smoother in the example above, we can also overlay more structured summaries such as regression curves.

Here's the plot that we saw at the end of Lecture 5, which makes good use of regression line overlays.

```{r, fig.height=5, fig.width=6, fig.align='center'}
ggplot(birthwt, aes(x=mother.age, y=birthwt.grams, shape=mother.smokes,
```

```
color=mother.smokes)) +
  geom_point() + # Adds points (scatterplot)
  geom_smooth(method = "lm") + # Adds regression lines
  ylab("Birth Weight (grams)") + # Changes y-axis label
  xlab("Mother's Age (years)") + # Changes x-axis label
  ggtitle("Birth Weight by Mother's Age") # Changes plot title
```

Recall that when we calculated correlations between birth weight and mother's age, we got the following results.

```{r}
by(data = birthwt[c("birthwt.grams", "mother.age")],
   INDICES = birthwt["mother.smokes"],
   FUN = function(x) {cor(x[,1], x[,2])})
```

This tells us that the association between mother's age and the baby's birthweight seems to depend on the mother's smoking status. Among mothers that smoke, this association is negative (older mothers tend to give birth to lower weight babies), while among mothers that don't smoke, the association is positive.

We can read off the same conclusions more simply from the plot. The regression lines capture the association between birthweight and mother's age. The fact that the observed points are very spread out around the regression lines visually conveys the fact that the correlation between birthweight and mother's age is not very large.

Is that an outlier?

The plot shows us something that the correlation calculation cannot: there's a non-smoking mother who was 45 years old when she gave birth, and the combination of her age and the baby's weight (around 5000g) are well outside the range of all other data points. Let's see what happens when we remove this observation and re-plot.

```{r, fig.height=5, fig.width=6, fig.align='center'}
# Get new data set that no longer contains the outlier
birthwt.sub <- subset(birthwt, subset = mother.age < 40)

ggplot(birthwt.sub, aes(x=mother.age, y=birthwt.grams,
shape=mother.smokes, color=mother.smokes)) +
  geom_point() + # Adds points (scatterplot)
  geom_smooth(method = "lm") + # Adds regression lines
  ylab("Birth Weight (grams)") + # Changes y-axis label
  xlab("Mother's Age (years)") + # Changes x-axis label
  ggtitle("Birth Weight by Mother's Age") # Changes plot title
```

The general trends appear to be qualitatively the same, but the association in the non-smoking group appears to be far less strong than we were led to believe when we still had the outlying point in our data.

ggplot2 also does maps

We'll need the `maps` library for this example.

```{r}
library(maps)
```

One of the data sets that comes with R is the `USArrests` data

```{r}
head(USArrests)
```

Here's how we can get a headmap of Murder rates (per 100,000 population) on a map of the US.

##map#1
```{r, fig.width = 6.5, fig.height = 3.5, fig.align='center'}

# Create data frame for map data (US states)
states <- map_data("state")

# Here's what the states data frame looks like
str(states)

# Make a copy of the data frame to manipulate
arrests <- USArrests

# Convert everything to lower case
names(arrests) <- tolower(names(arrests))
arrests$region <- tolower(rownames(USArrests))

# Merge the map data with the arrests data based on region
choro <- merge(states, arrests, sort = FALSE, by = "region")
choro <- choro[order(choro$order), ]

# Plot a map, filling in the states based on murder rate
qplot(long, lat, data = choro, group = group, fill = murder,
  geom = "polygon")
```

Essentially what's happening here is that the map data (here called `states`) includes the latitute and longitude coordinates for the boundaries of each state. Specifying `geom = "polygon"` in the `qplot` function results in the states being traced out based on the lat/long coordinates. The `arrests` data provides crime rates for all the states, which allows us to color each polygon (state) based on the murder rate.

It may be counter-intuitive that light blue indicates areas with high murder rates while dark blue indicates areas with low murder rates. We can fix this by specifying a different gradient using the `scale_colour_gradient()` command. Here's an example.
##map#2

```{r, fig.width = 6.5, fig.height = 3.5, fig.align='center'}
qplot(long, lat, data = choro, group = group, fill = murder,
  geom = "polygon") +
  scale_fill_gradient(low = "#56B1F7", high = "#132B43")
```

All we've done here is swapped the defaults for `low` and `high`.