Chapter 7: Data Visualization, ggplot

M Affouf

1/8/2018

part1

Basic Plots

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

Read in Data

```
death = read.csv("indicatordeadkids35.csv",
                 as.is=TRUE,header=TRUE, row.names=1)
death[1:2, 1:5]
```

```
##             X1760 X1761 X1762 X1763 X1764
## Afghanistan    NA    NA    NA    NA    NA
## Albania        NA    NA    NA    NA    NA
```

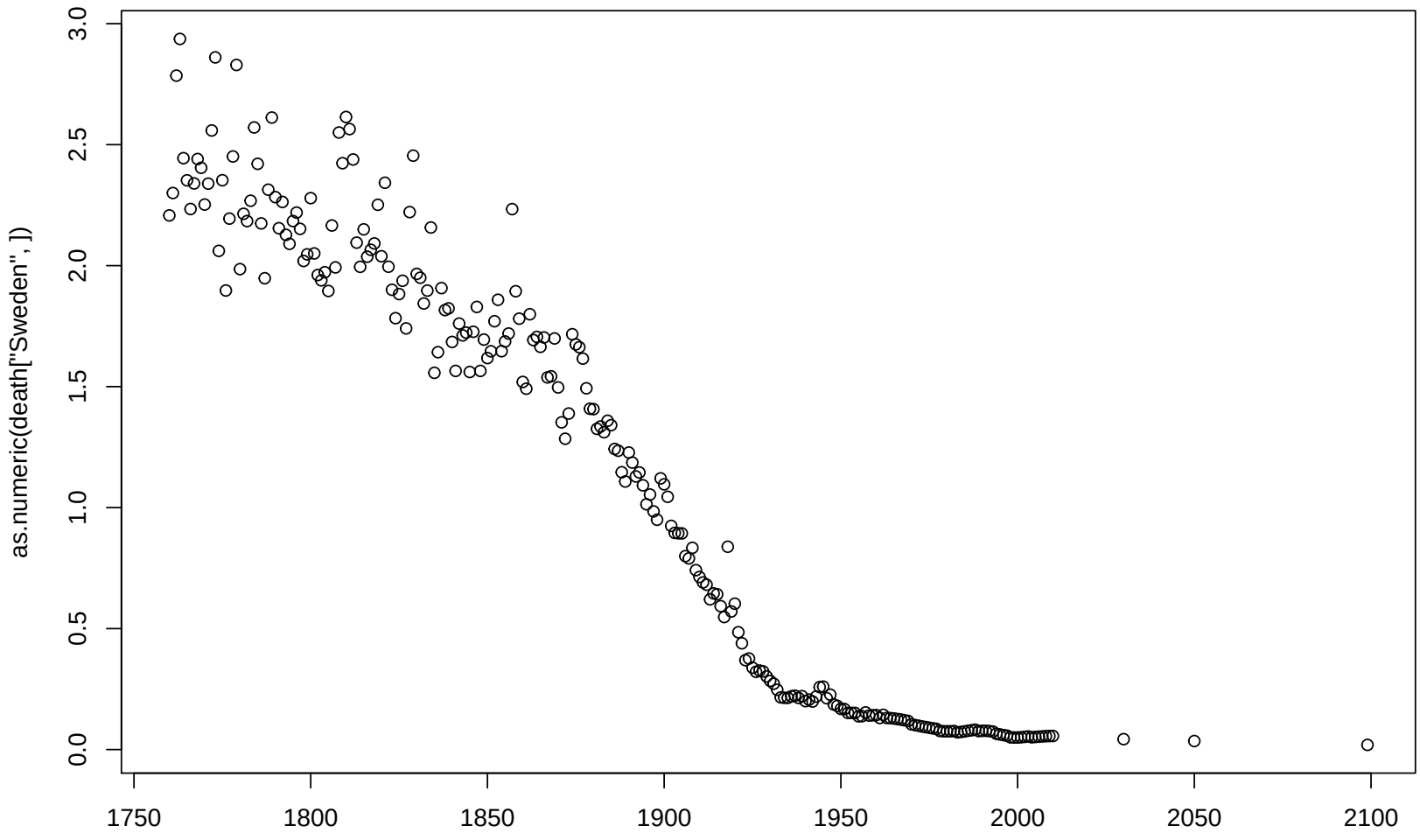
We see that the column names were years, and R doesn't necessarily like to read in a column name that starts with a number and puts an X there.
We'll just take off that X and get the years.

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

```
## [1] 1760 1761 1762 1763 1764 1765
```

Basic Plots

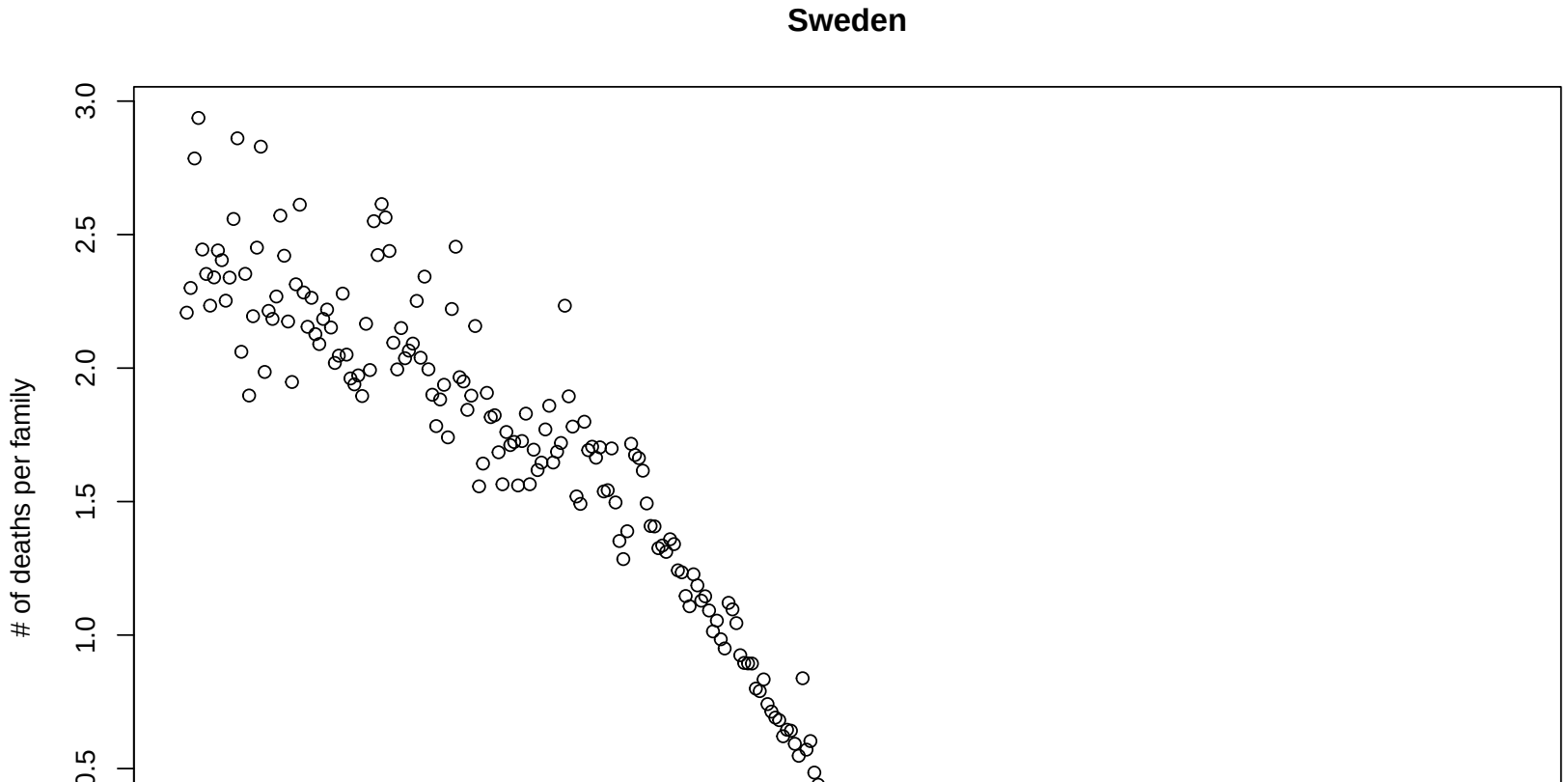
```
> plot(as.numeric(death["Sweden",]) ~ year)
```



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.

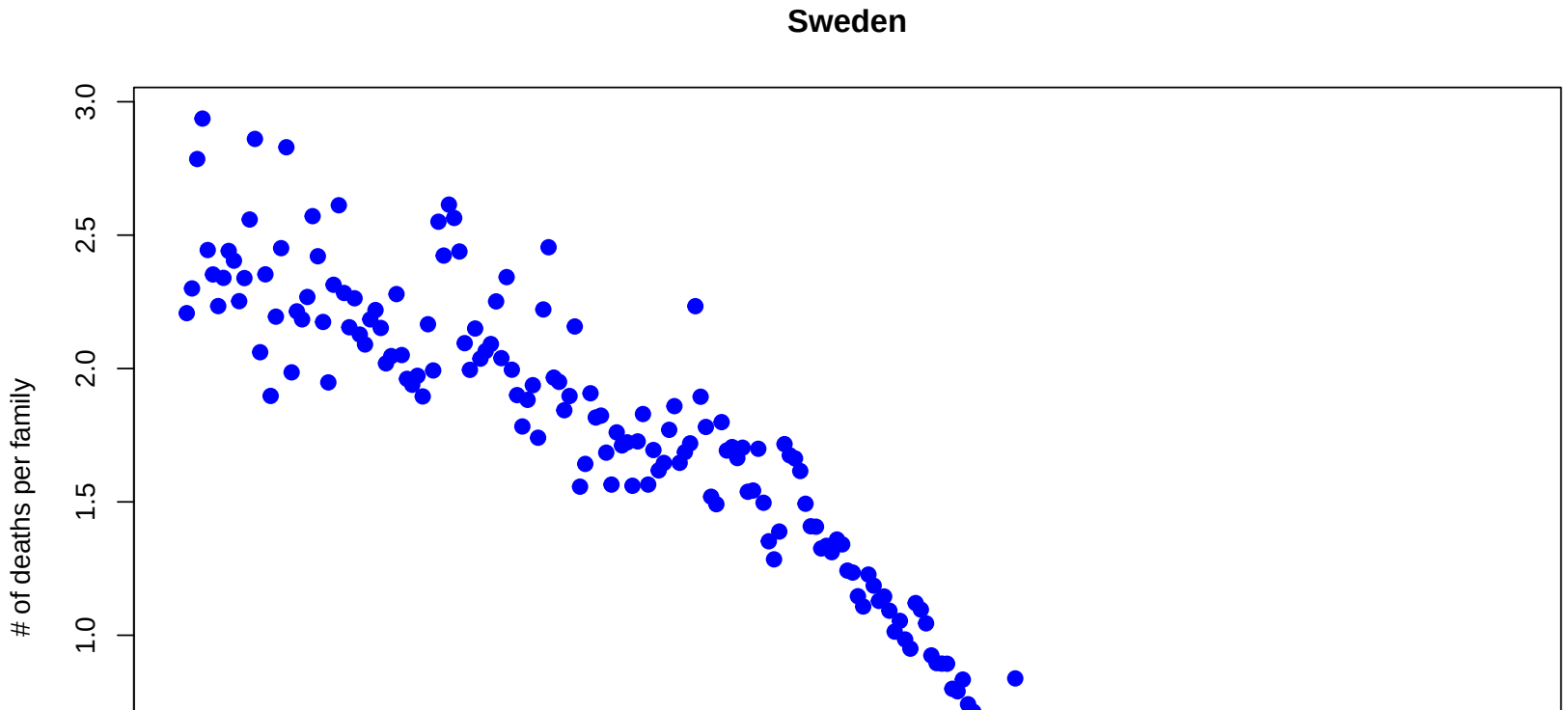
```
> plot(as.numeric(death["Sweden",]) ~ year,
+       ylab = "# of deaths per family", main = "Sweden")
```



Basic Plots

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

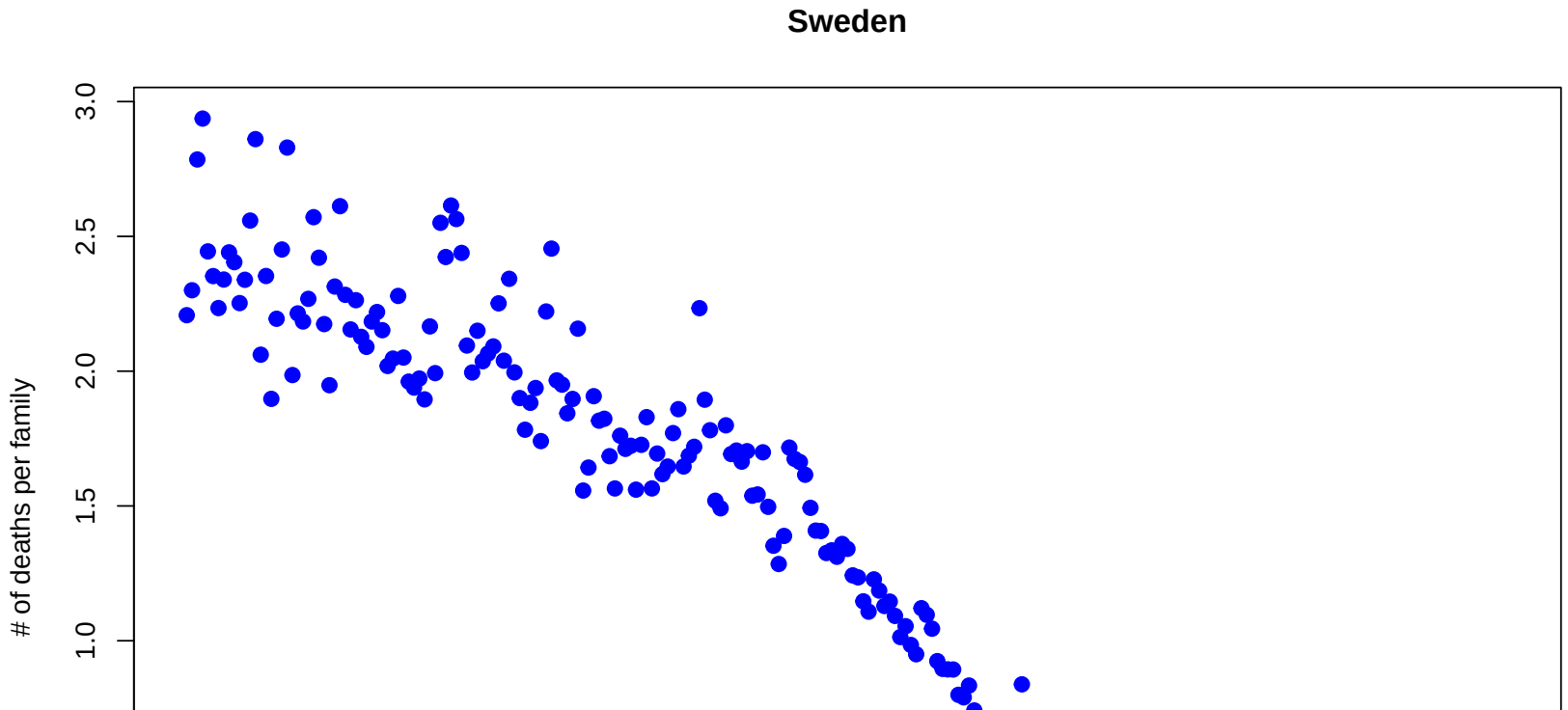
```
plot(as.numeric(death["Sweden",])~year,
      ylab = "# of deaths per family", main = "Sweden",
     xlim = c(1760,2012), pch = 19, cex=1.2,col="blue")
```



Basic Plots

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

```
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.

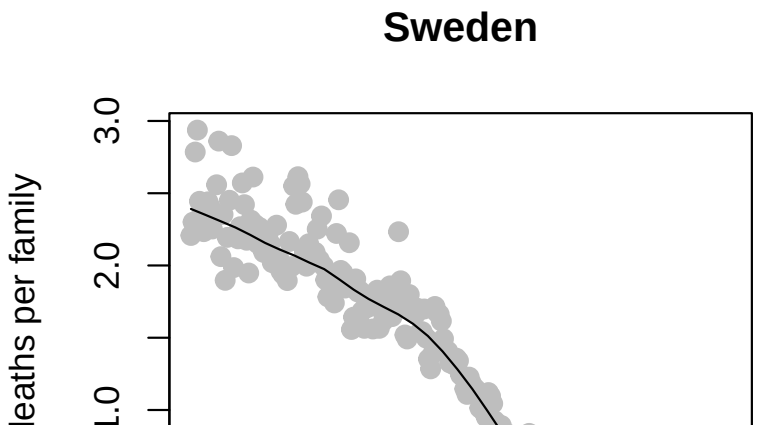
```
> scatter.smooth(as.numeric(death["Sweden",])~year,span=0.2
+       ylab="# of deaths per family", main = "Sweden",lwd=
+      subset = year < 2015, pch = 19, cex=0.9,col="grey")
```

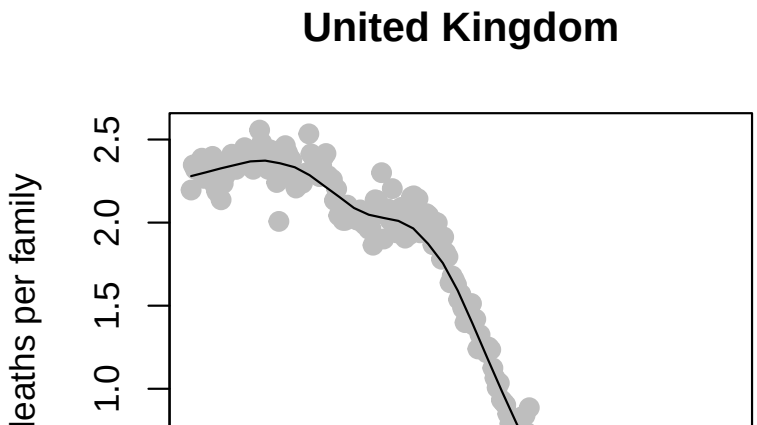
```
Warning in plot.window(...): "subset" is not a graphical pa

Warning in plot.xy(xy, type, ...): "subset" is not a graphi

Warning in axis(side = side, at = at, labels = labels, ...)
not a graphical parameter

Warning in axis(side = side, at = at, labels = labels, ...)
not a graphical parameter

Warning in box(...): "subset" is not a graphical parameter
```

Basic Plots

`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)

```
> par(mfrow=c(1,2))
> scatter.smooth(as.numeric(death["Sweden",])~year,span=0.2
+       ylab="# of deaths per family", main = "Sweden",lwd=
+      xlim = c(1760,2012), pch = 19, cex=0.9,col="grey")
> scatter.smooth(as.numeric(death["United Kingdom",])~year,
+       ylab="# of deaths per family", main = "United Kingdo
+      xlim = c(1760,2012), pch = 19, cex=0.9,col="grey")
```

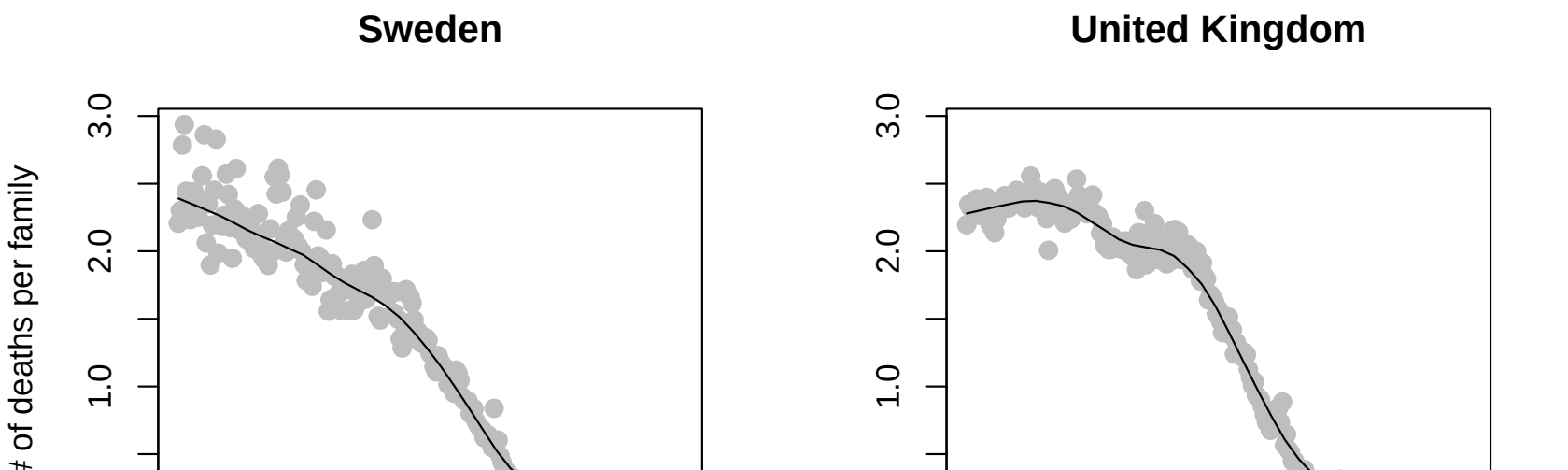




Basic Plots

We can set the y-axis to be the same.

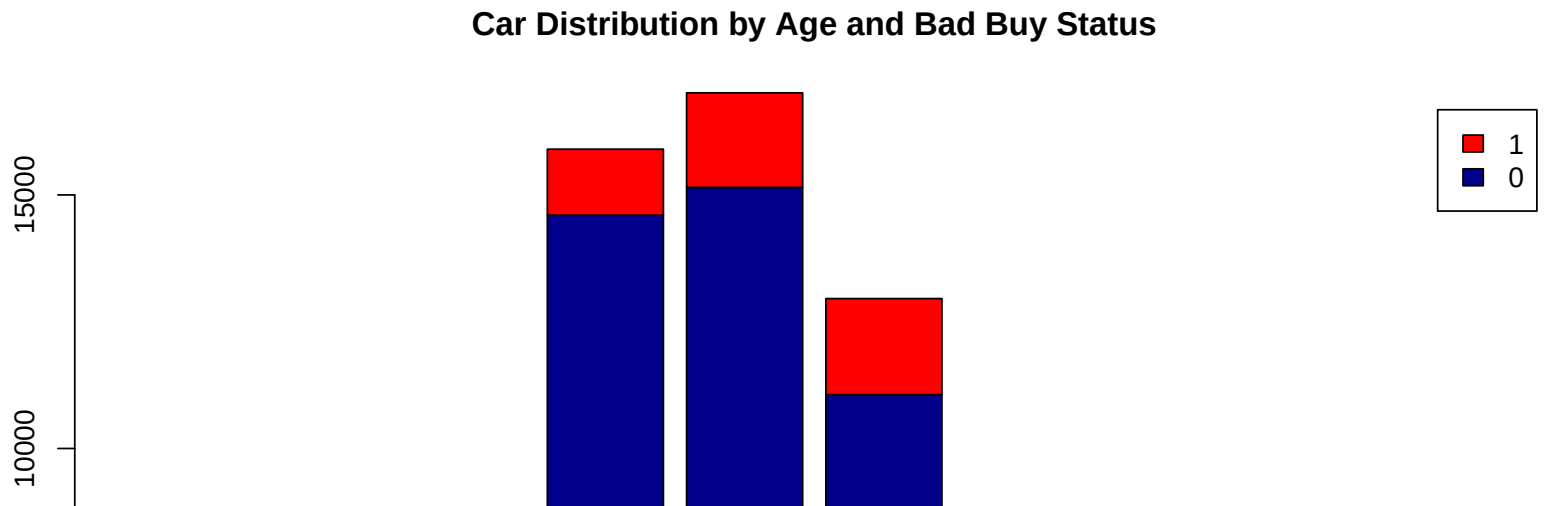
```
par(mfrow=c(1,2))
yl = range(death[c("Sweden","United Kingdom"),])
scatter.smooth(as.numeric(death["Sweden",])~year,span=0.2,y
      ylab="# of deaths per family", main = "Sweden",lwd=3,
     xlim = c(1760,2012), pch = 19, cex=0.9,col="grey")
scatter.smooth(as.numeric(death["United Kingdom",])~year,sp
      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

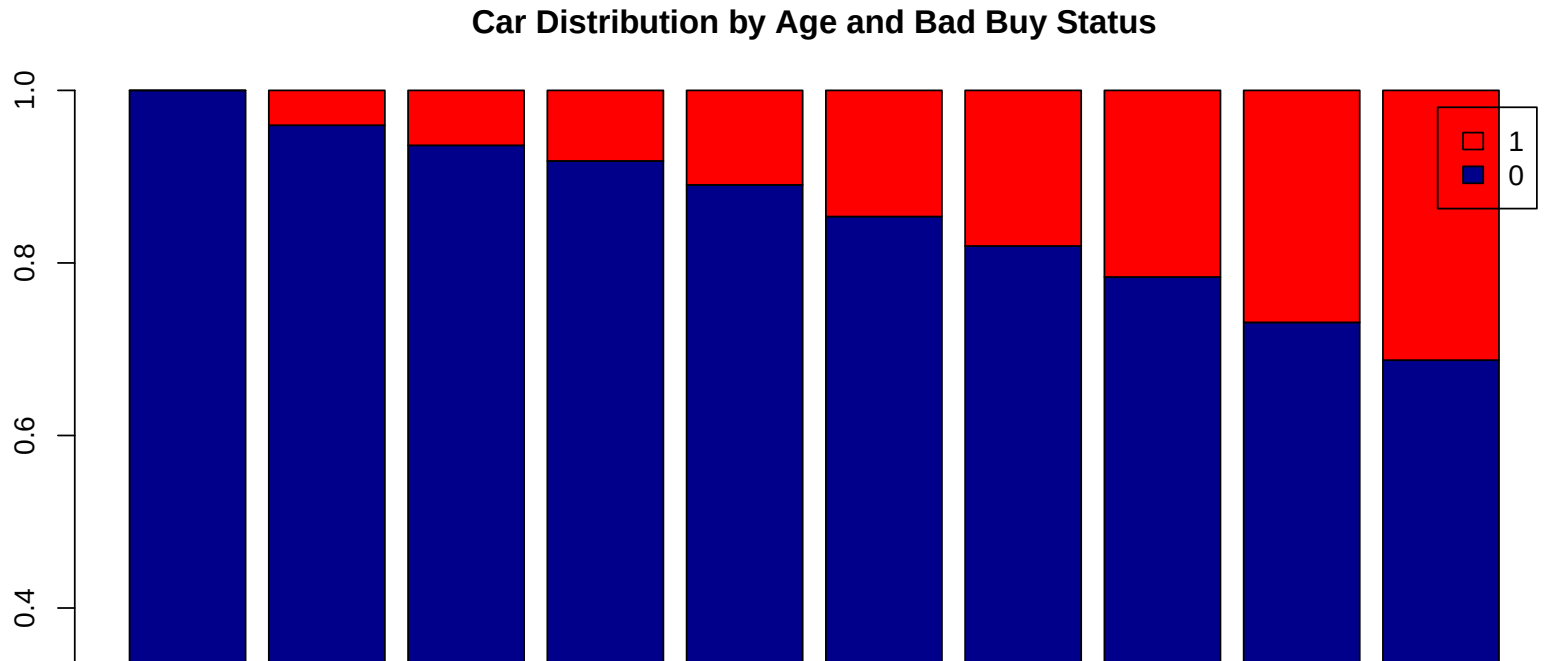
```
## Stacked Bar Charts
cars = read.csv("kaggleCarAuction.csv",as.is=T)
counts <- table(cars$IsBadBuy, cars$VehicleAge)
barplot(counts, main="Car Distribution by Age and Bad Buy S
  xlab="Vehicle Age", col=c("darkblue","red"),
    legend = rownames(counts))
```



Bar Plots

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

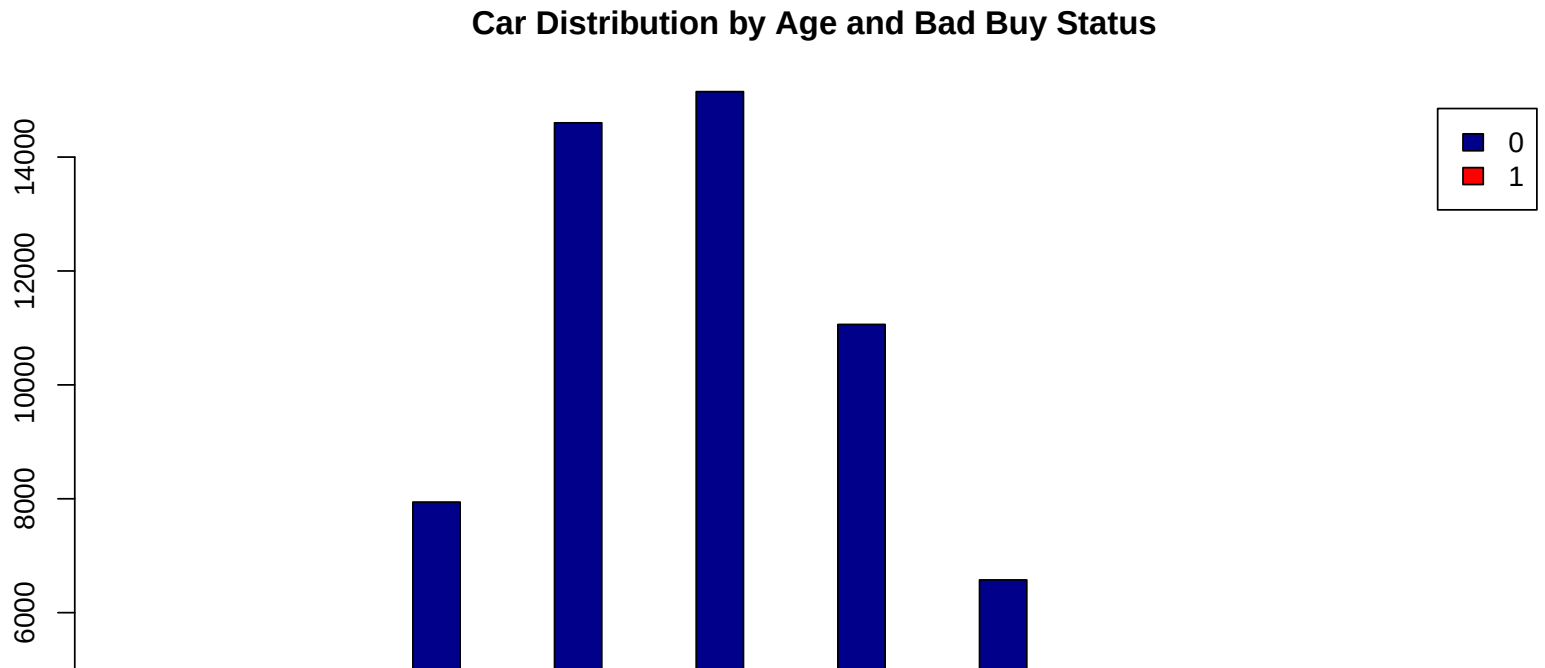
```
## Use percentages (column percentages)
barplot(prop.table(counts, 2), main="Car Distribution by Ag
  xlab="Vehicle Age", col=c("darkblue","red"),
    legend = rownames(counts))
```



Bar Plots

Using the `beside` argument in `barplot`, you can get side-by-side barplots.

```
# Stacked Bar Plot with Colors and Legend
barplot(counts, main="Car Distribution by Age and Bad Buy S
  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:

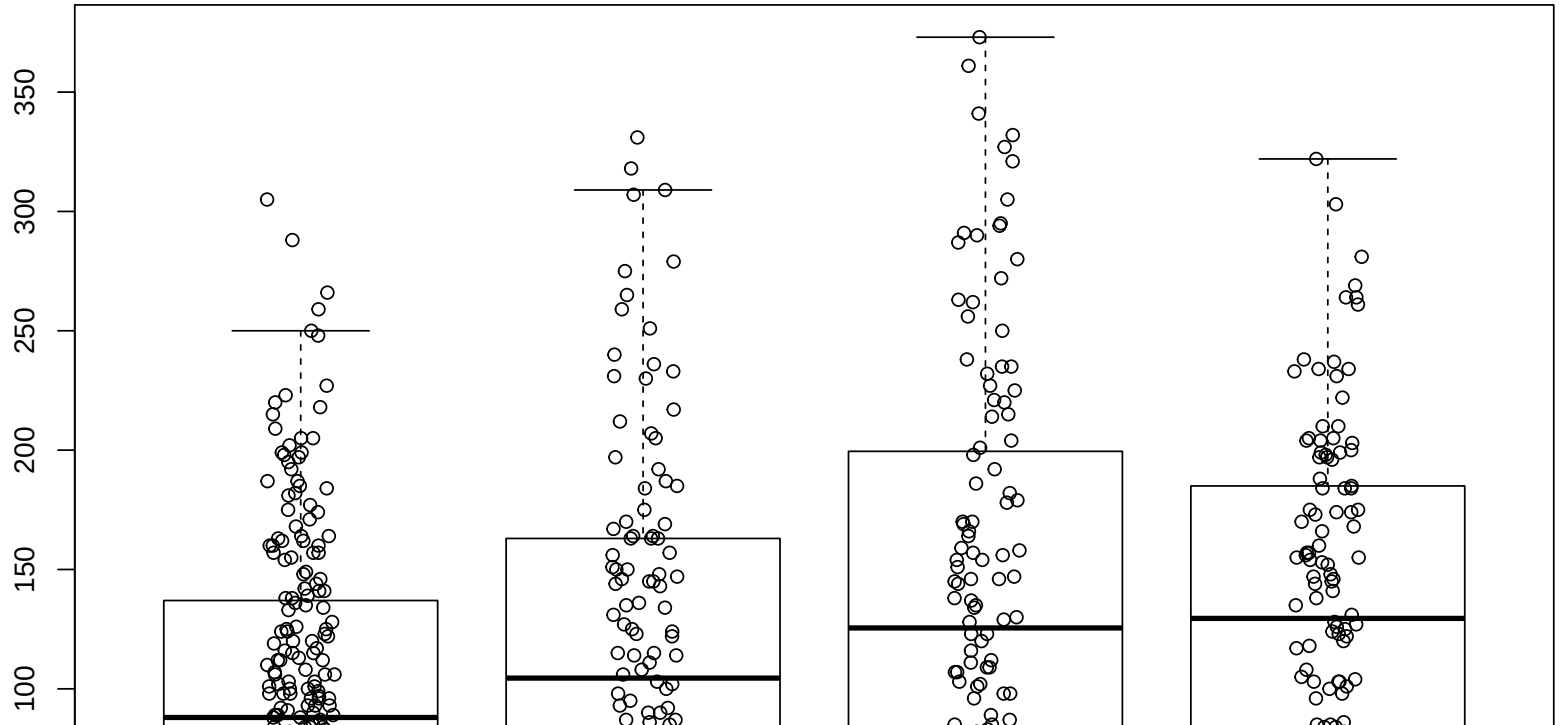
```
pdf("filename.pdf", width=8, height=8) # inches
plot() # plot 1
plot() # plot 2
# etc
dev.off()
```

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...

```
> boxplot(weight ~ Diet, data=ChickWeight, outline=FALSE)
> points(ChickWeight$weight ~ jitter(as.numeric(ChickWeight
```



Formulas

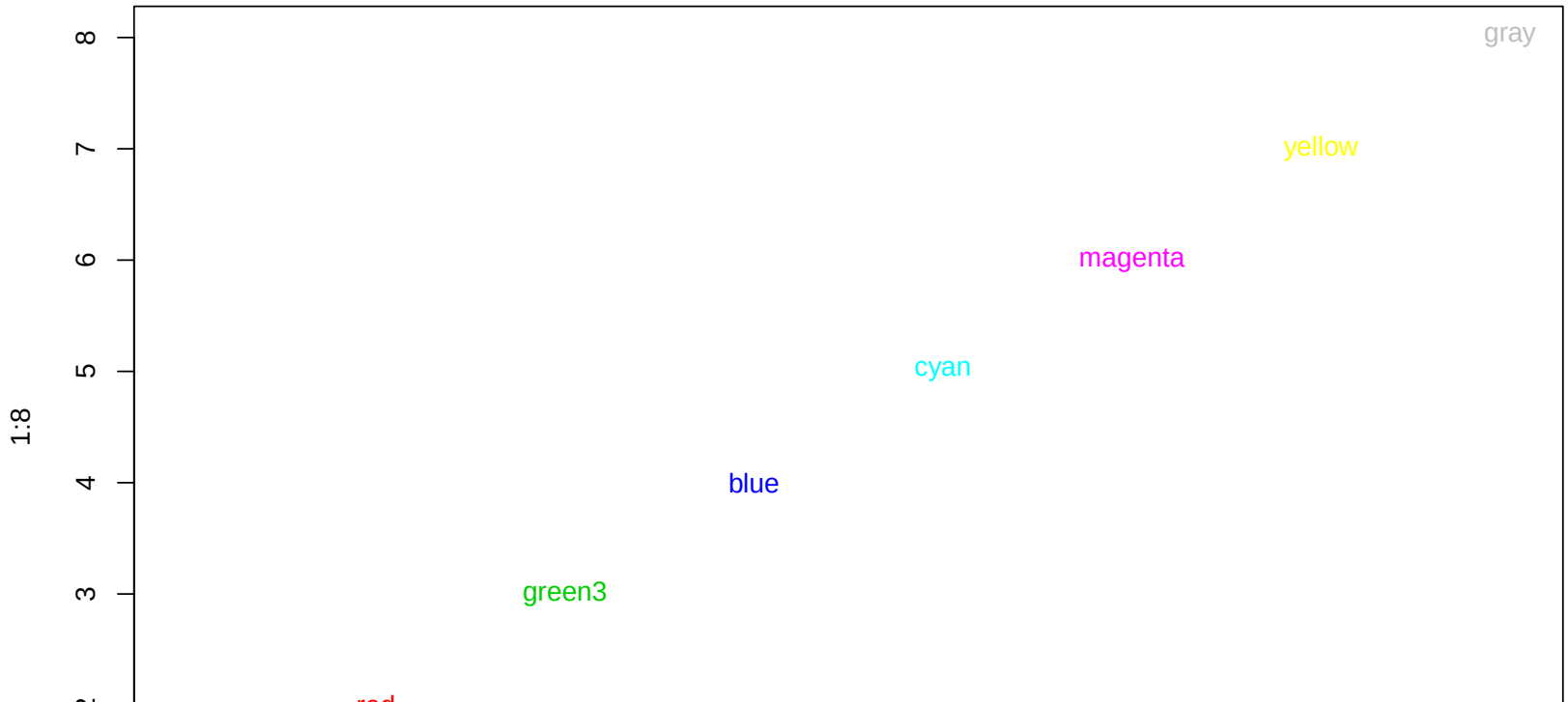
Formulas have the format of `y ~ x` and functions taking formulas have a `data` argument where you pass the data.frame. You don't need to use `$` or referencing when using formulas:

```
boxplot(weight ~ Diet, data=ChickWeight, outline=FALSE)
```

Colors

R relies on color 'palettes'.

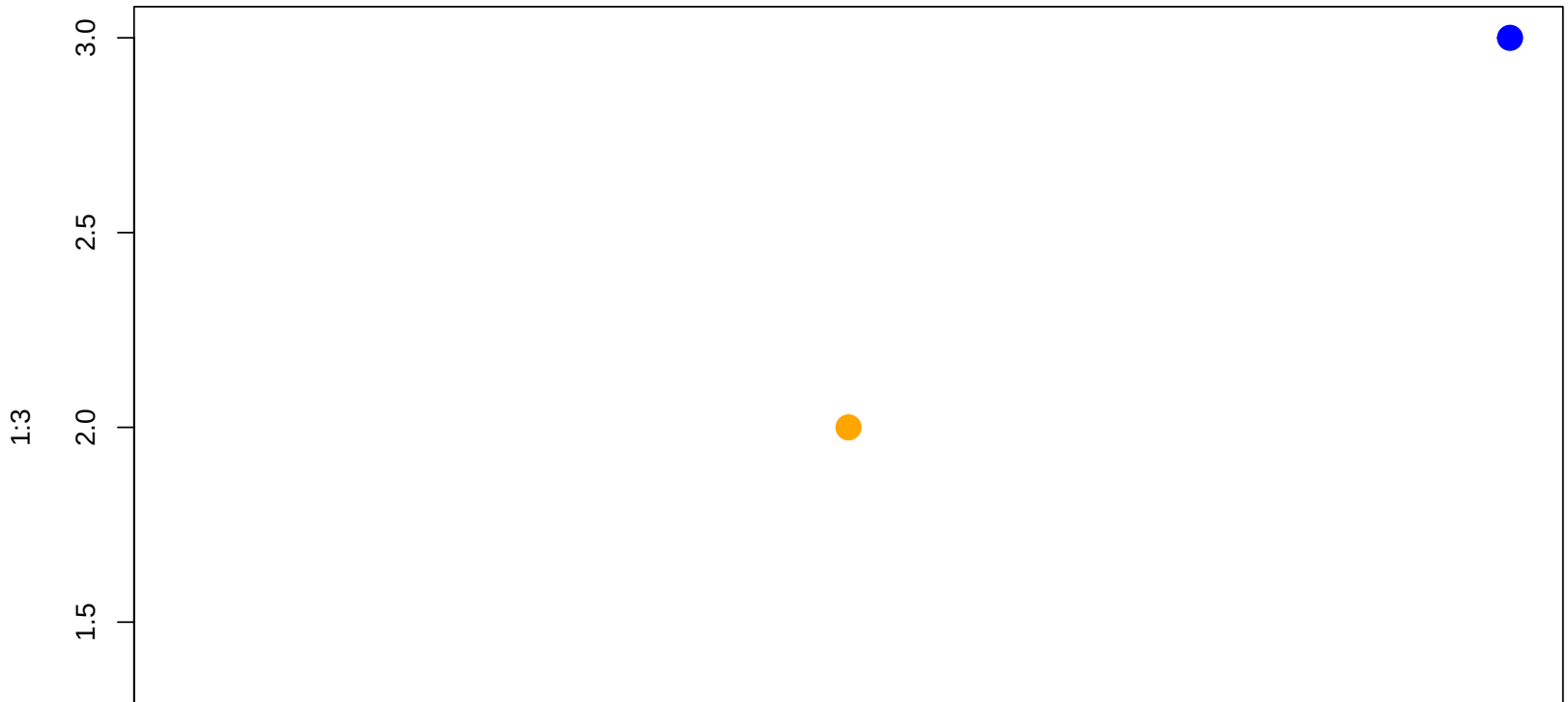
```
palette("default")
plot(1:8, 1:8, type="n")
text(1:8, 1:8, lab = palette(), col = 1:8)
```



Colors

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

```
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

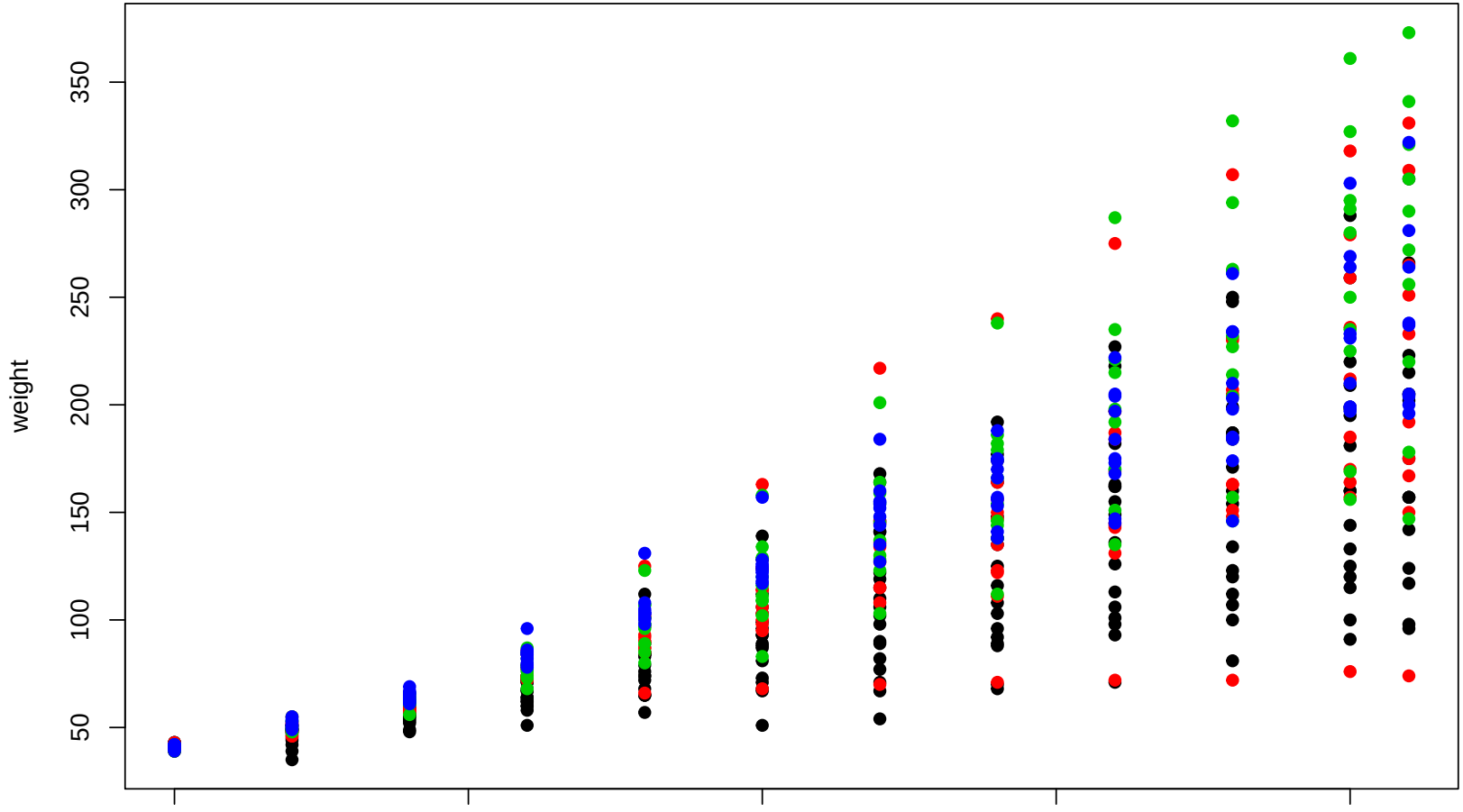
RColorBrewer::display.brewer.all() will show you all of the palettes available. You can even print it out and keep it next to your monitor for reference.

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:'

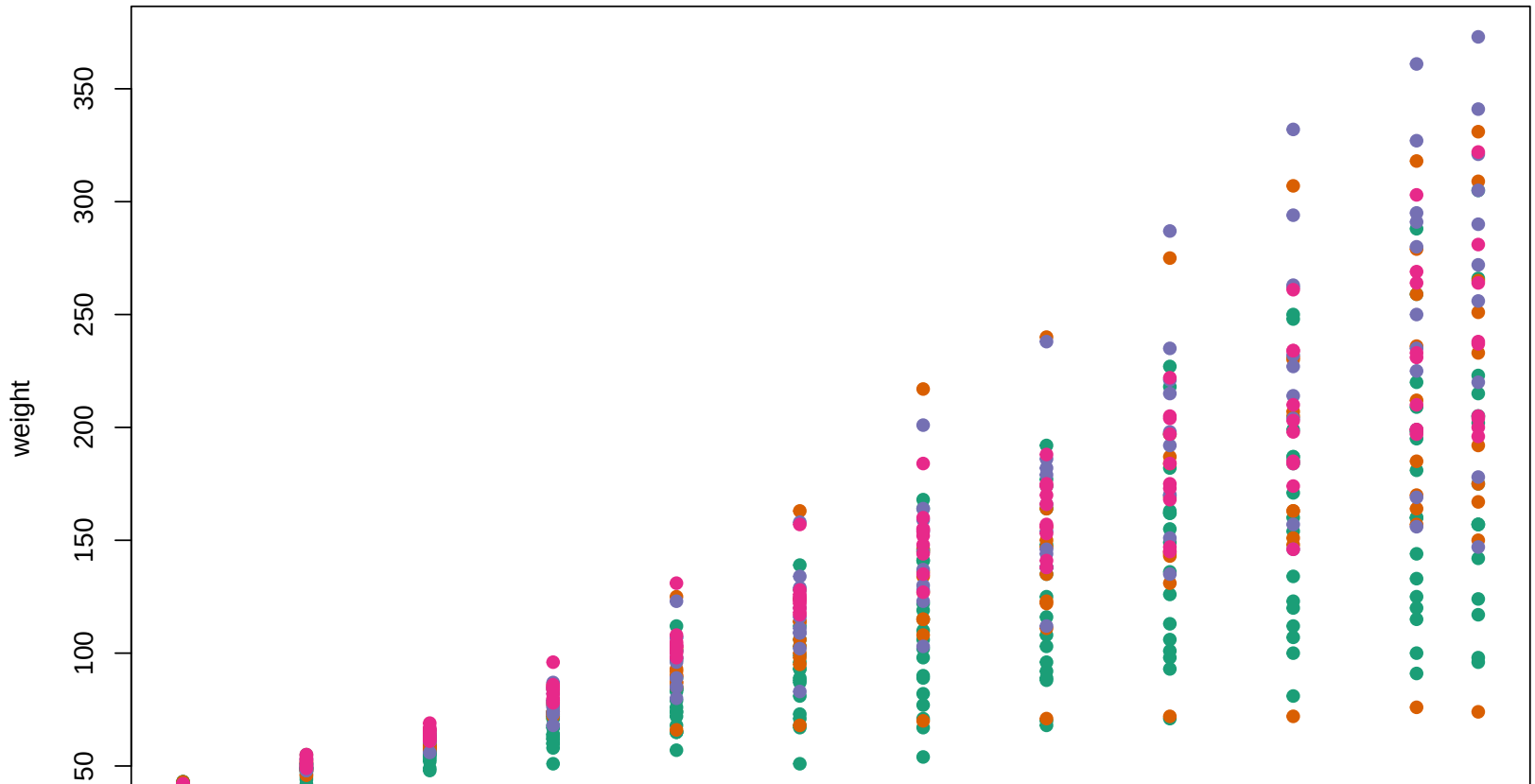
Colors

```
palette("default")
plot(weight ~ Time, data= ChickWeight, pch = 19, col = Diet
```



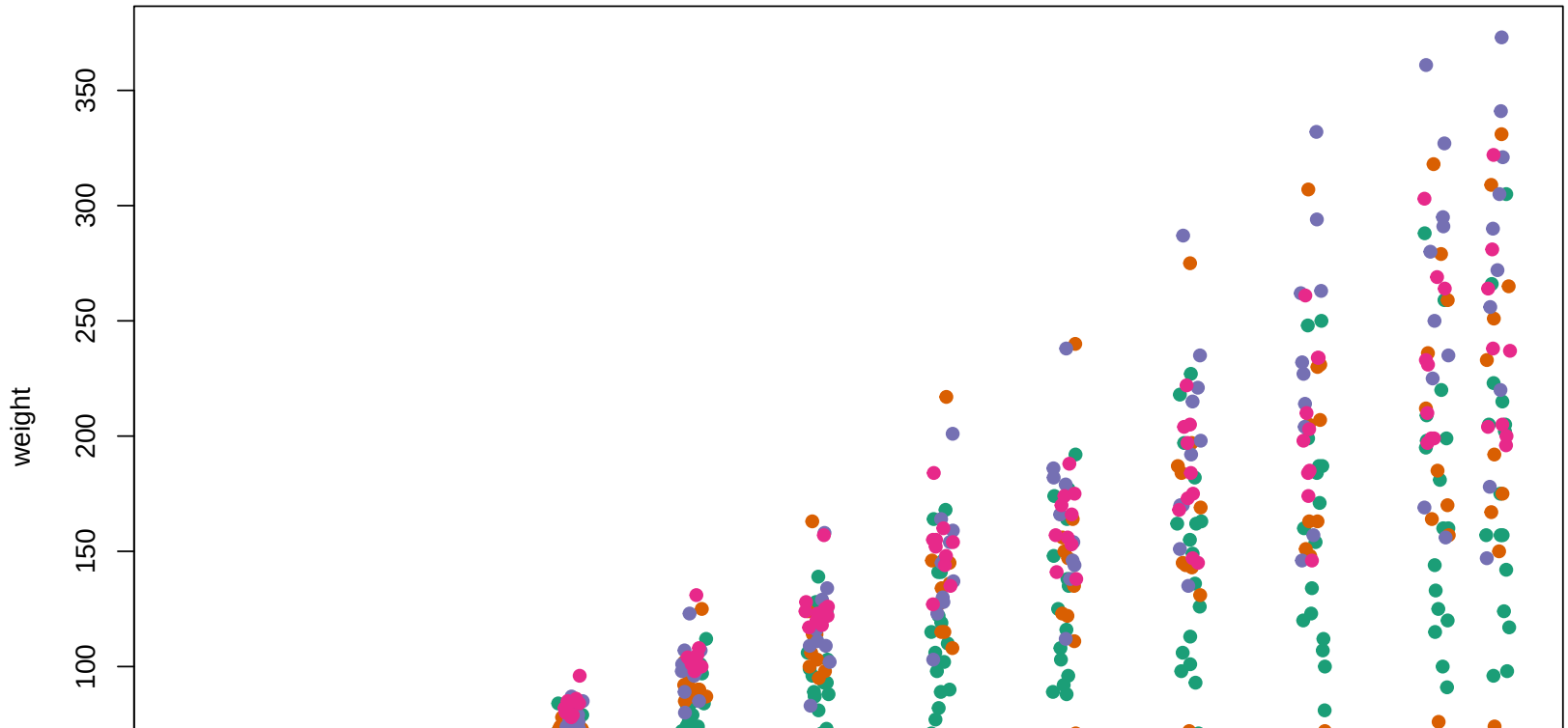
Colors

```
library(RColorBrewer)
palette(brewer.pal(5,"Dark2"))
plot(weight ~ Time, data=ChickWeight, pch = 19,  col = Diet
```



Colors

```
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”.

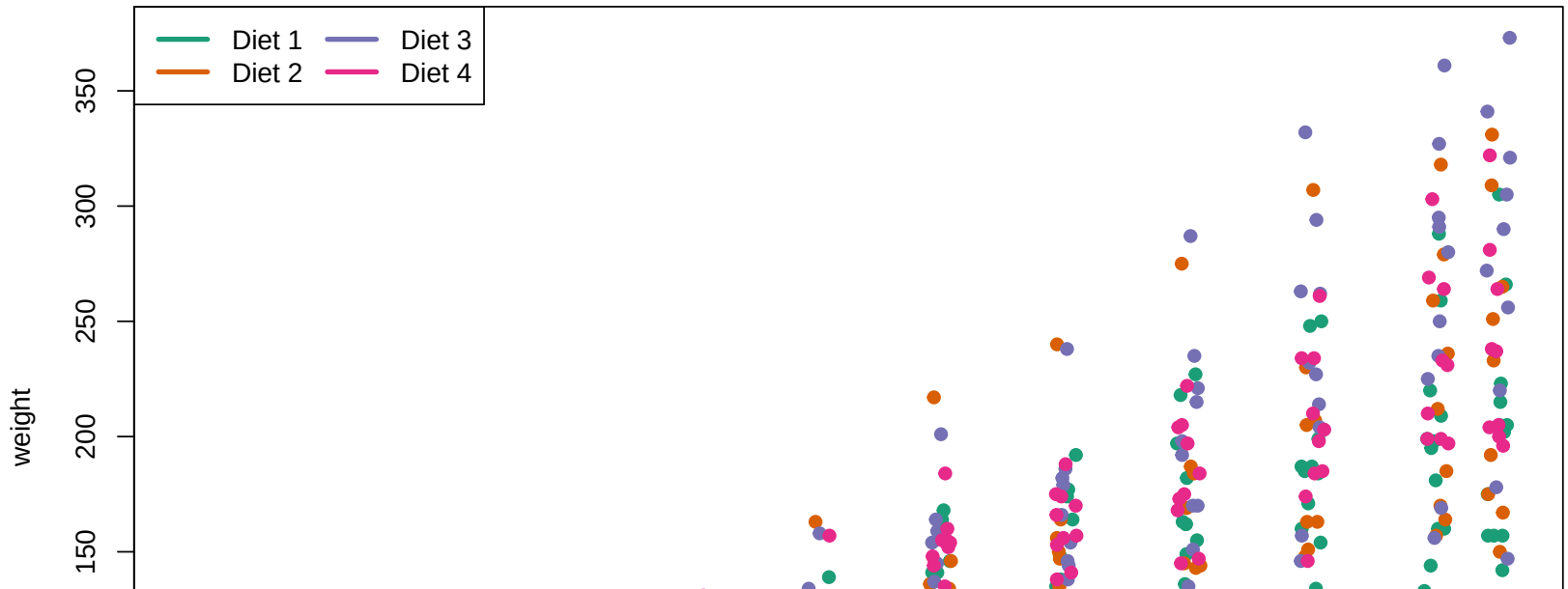
legend: unique character vector, the levels of a factor

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

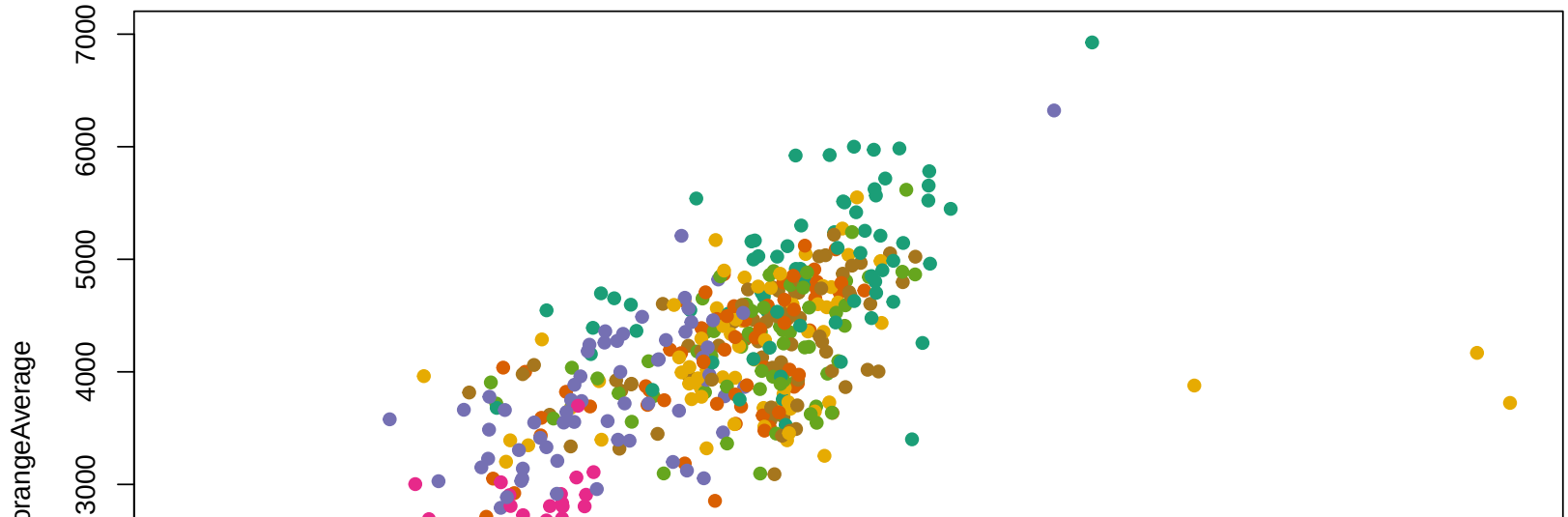
Adding legends

```
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)
```



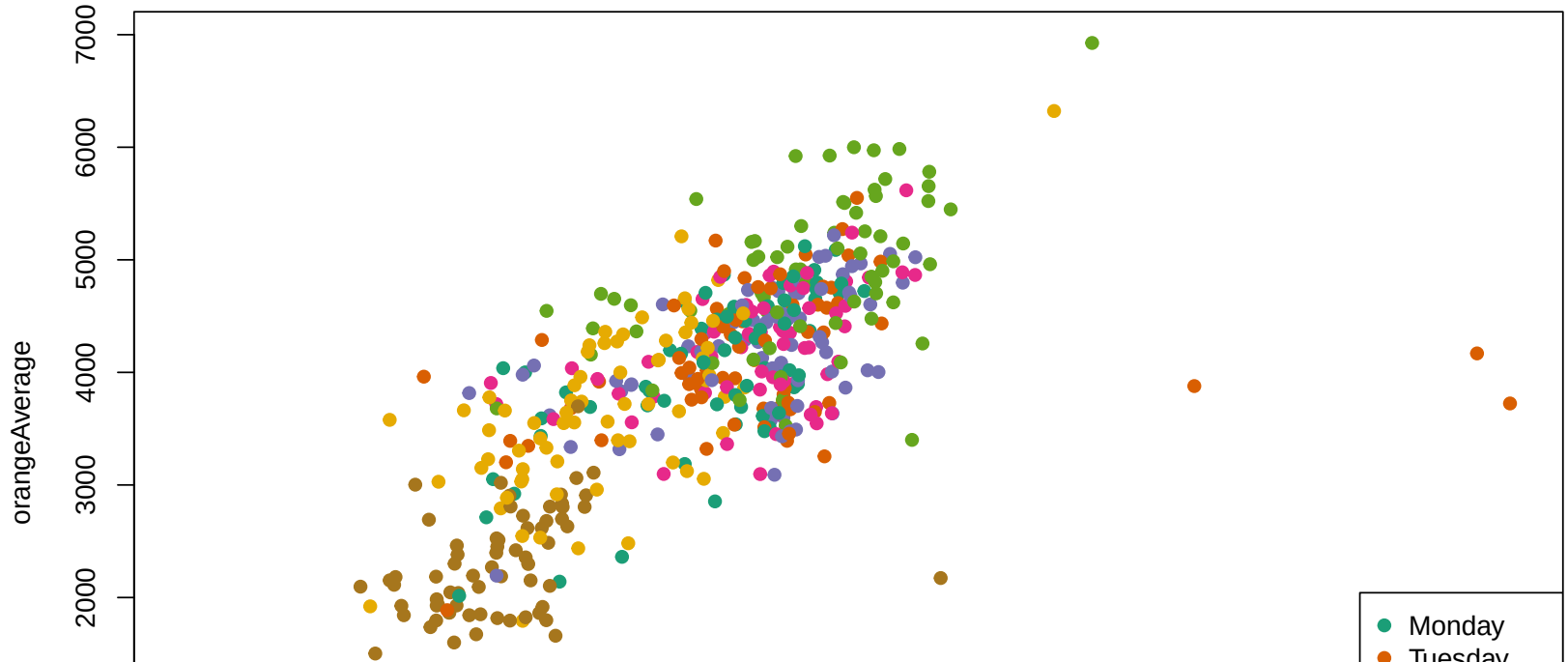
Coloring by variable

```
> 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 =
```



Coloring by variable

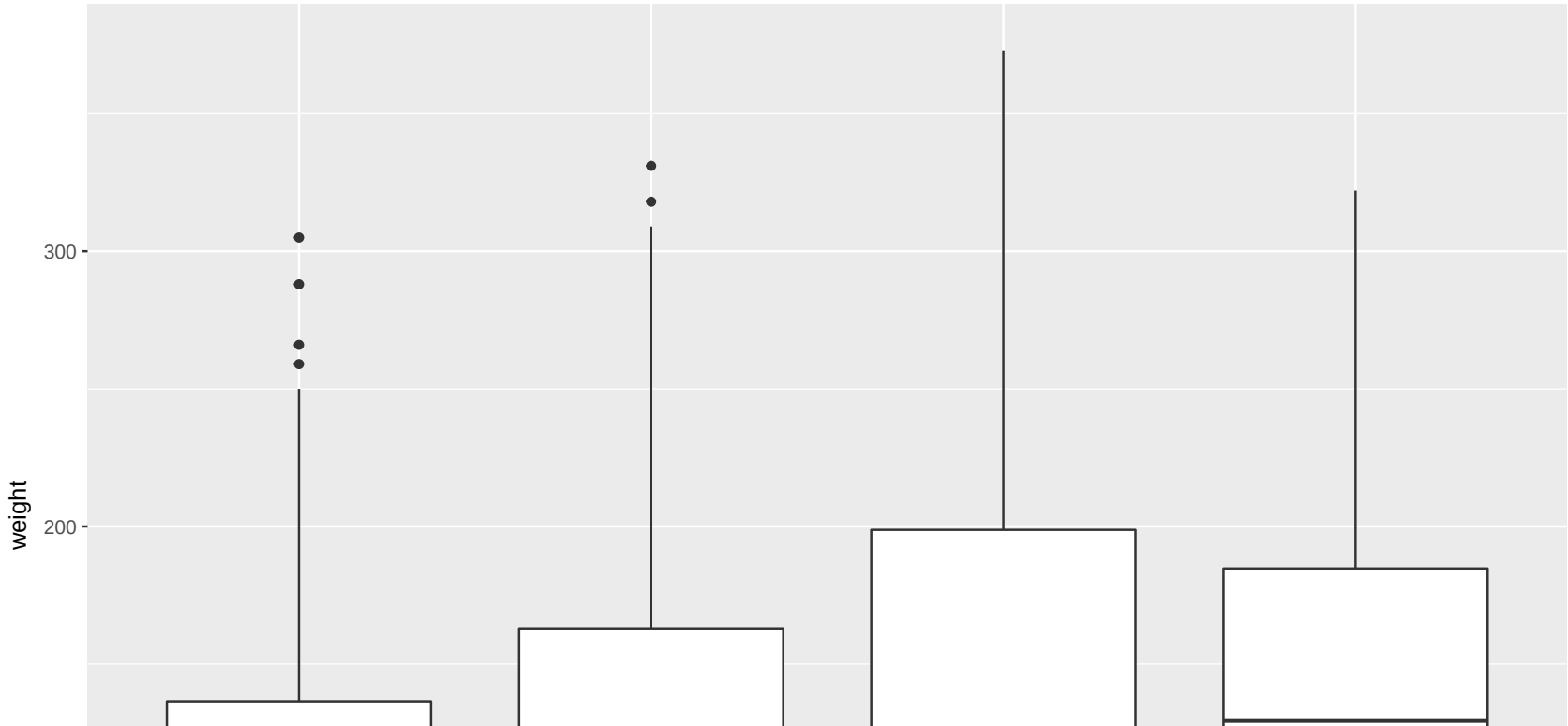
```
> dd = factor(circ$day, levels=c("Monday","Tuesday","Wednes
+              "Thursday","Friday","Saturday","Sunday"))
> plot(orangeAverage ~ greenAverage, data=circ,
+      pch=19, col = as.numeric(dd))
> legend("bottomright", levels(dd), col=1:length(dd), pch =
```



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:

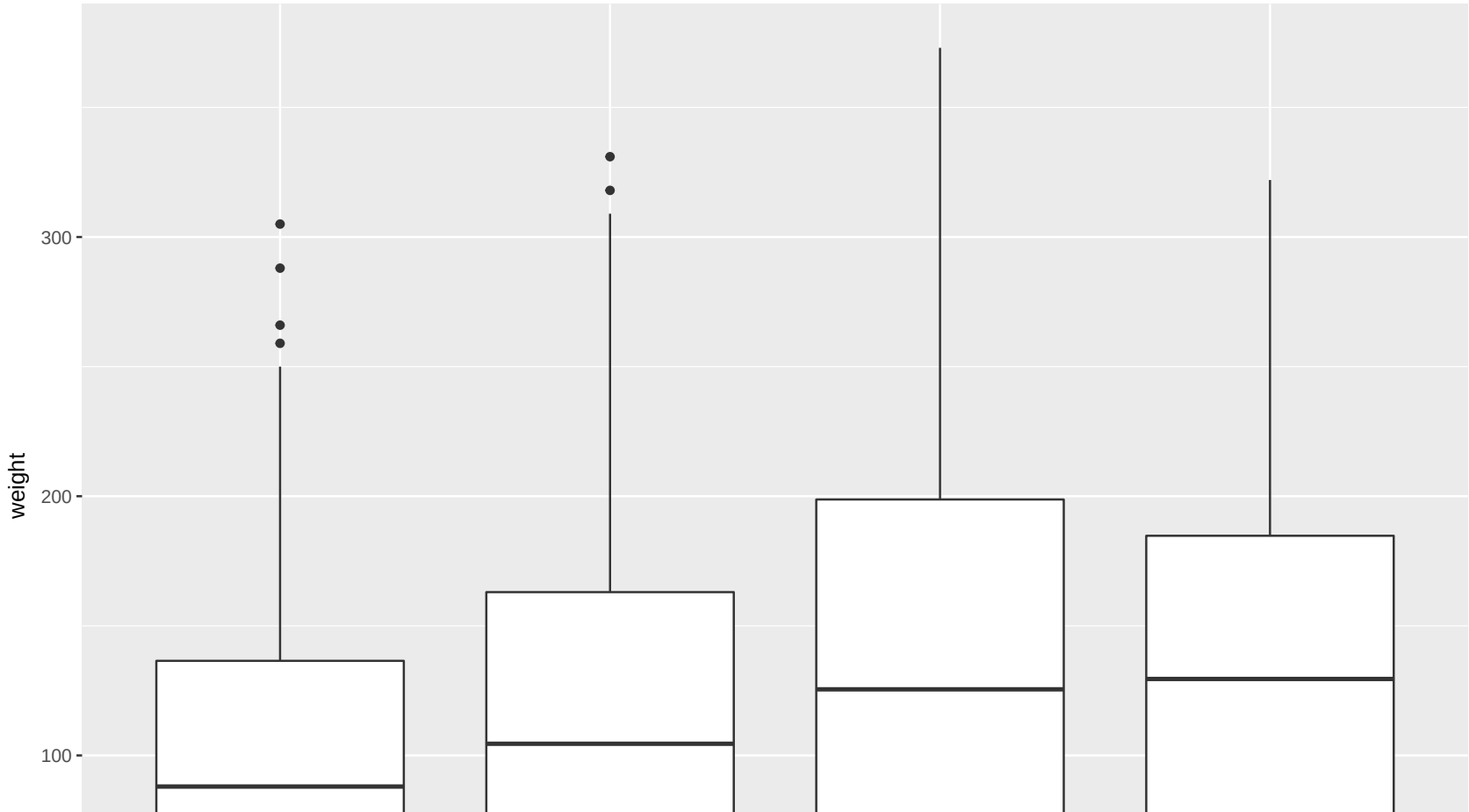
```
> library(ggplot2)
> qplot(factor(Diet), y = weight,
+       data = ChickWeight, geom = "boxplot")
```



ggplot2

The generic plotting function is ggplot:

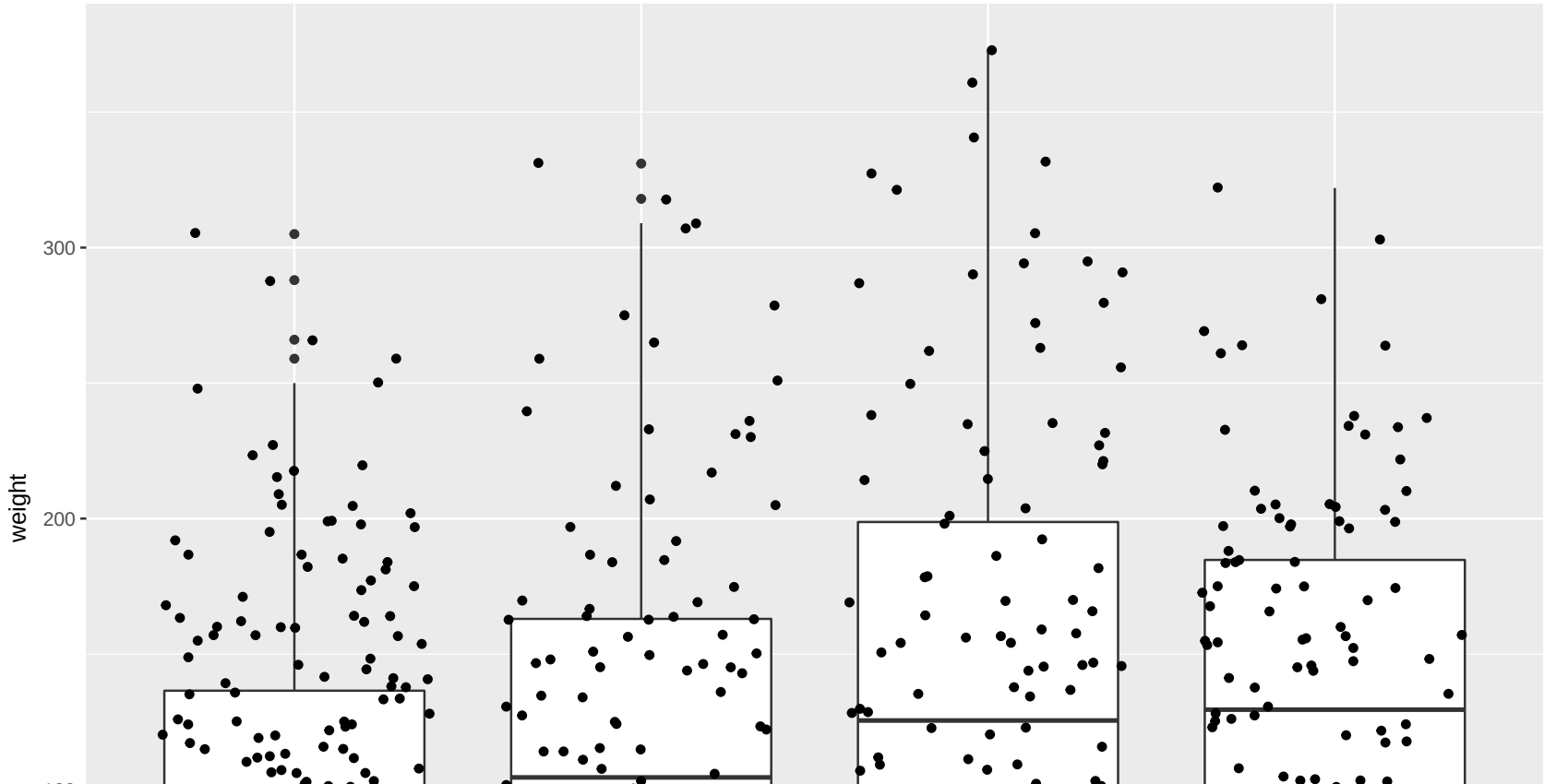
```
> g = ggplot(aes(x = Diet, y = weight), data = ChickWeight)
> g + geom_boxplot()
```



Boxplots revisited again

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

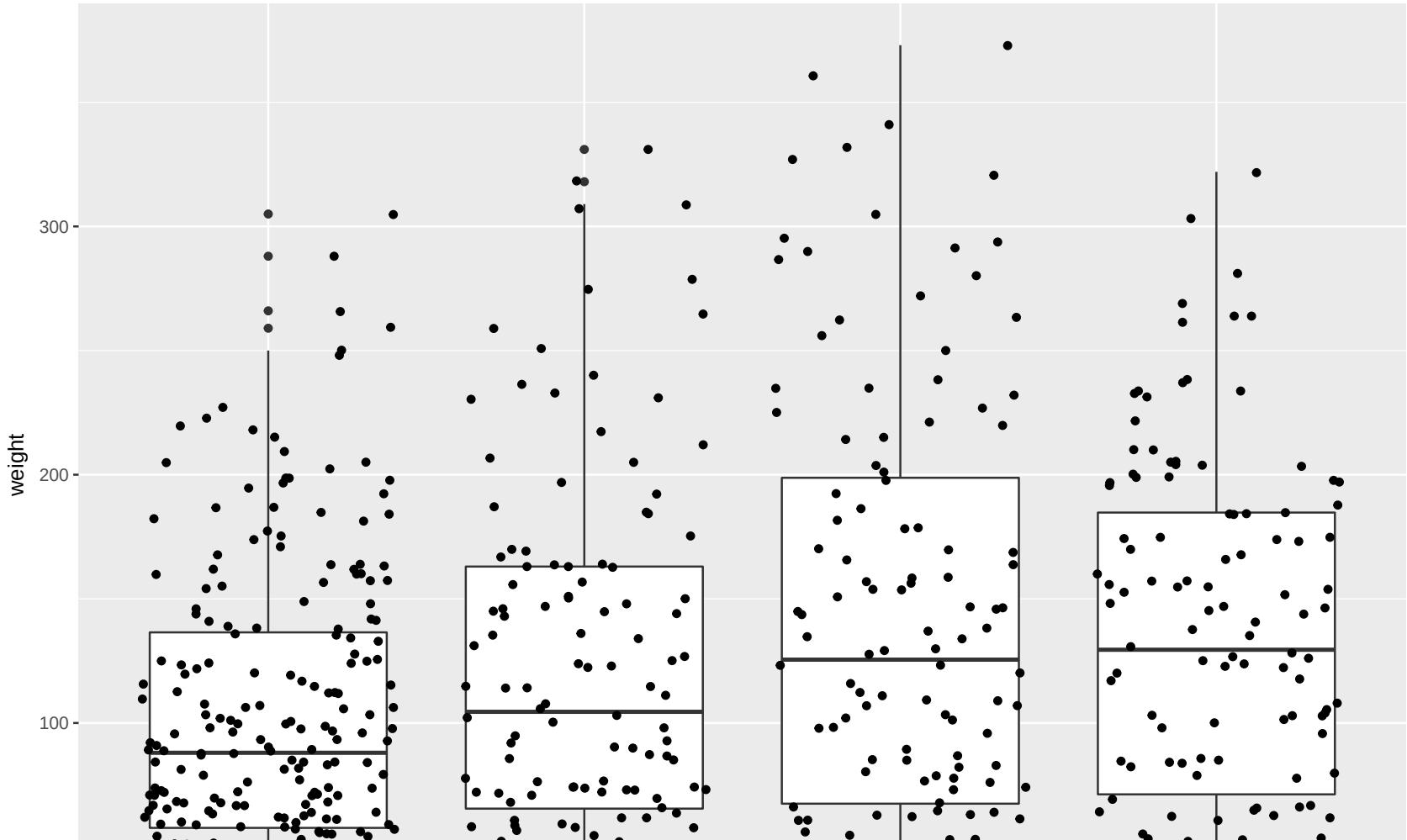
```
> qplot( factor(Diet), y = weight, data = ChickWeight,
+         geom = c("boxplot", "jitter"))
```



ggplot2: Adding 2 geoms together

To have multiple geometrics, just "add" them

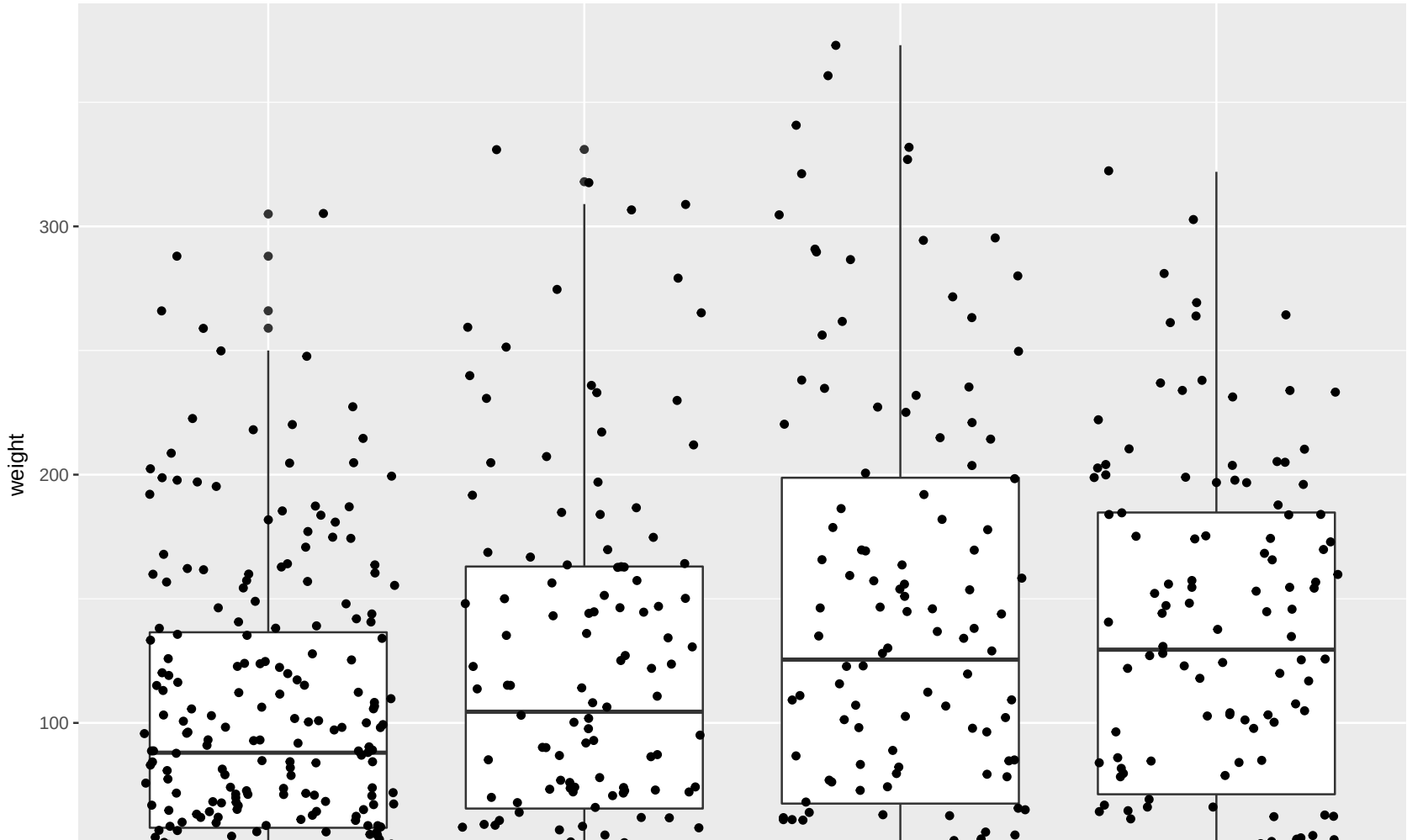
```
> g + geom_boxplot() + geom_point(position = "jitter")
```



ggplot2: Adding 2 geoms together

To have multiple geometrics, just "add" them

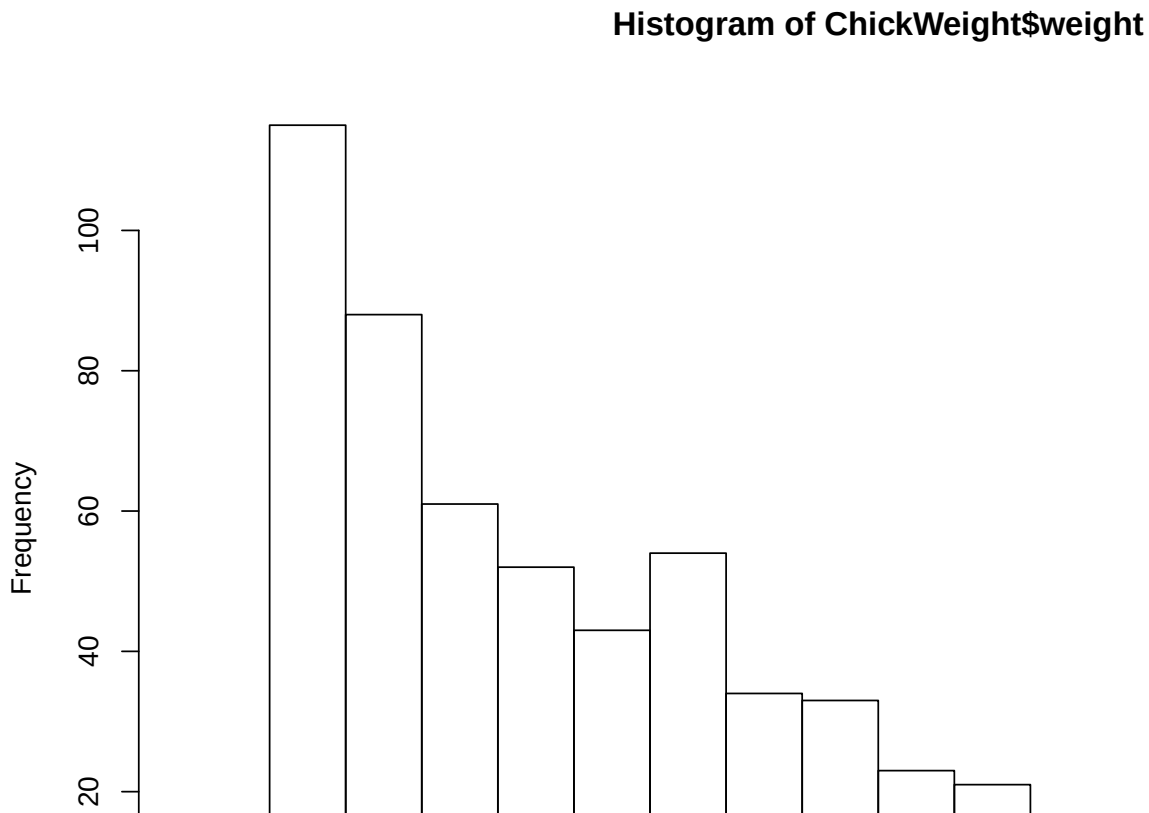
```
g + geom_boxplot() + geom_jitter()
```



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.

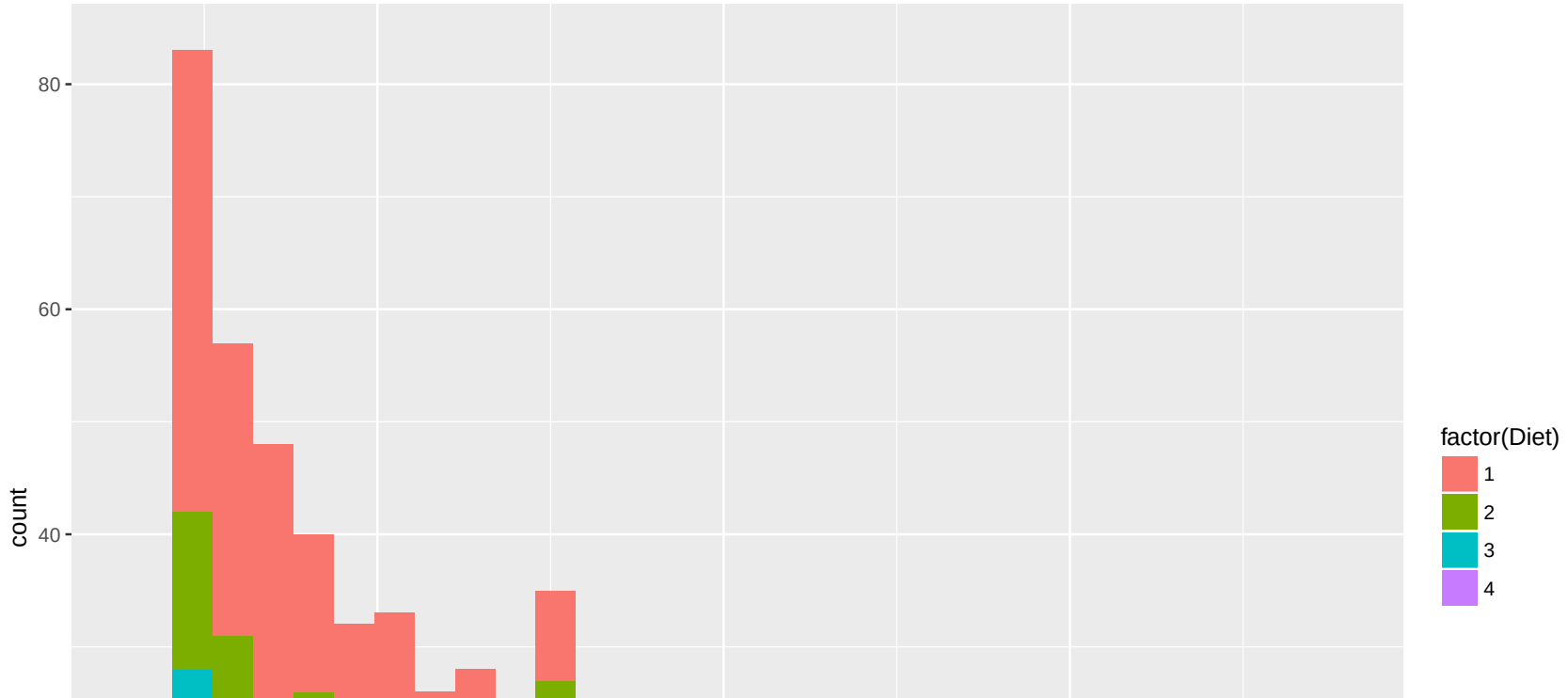
```
> hist(ChickWeight$weight, breaks = 20)
```



Multiple Histograms

```
> qplot(x = weight,
+       fill = factor(Diet),
+       data = ChickWeight,
+       geom = c("histogram"))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `bin
```

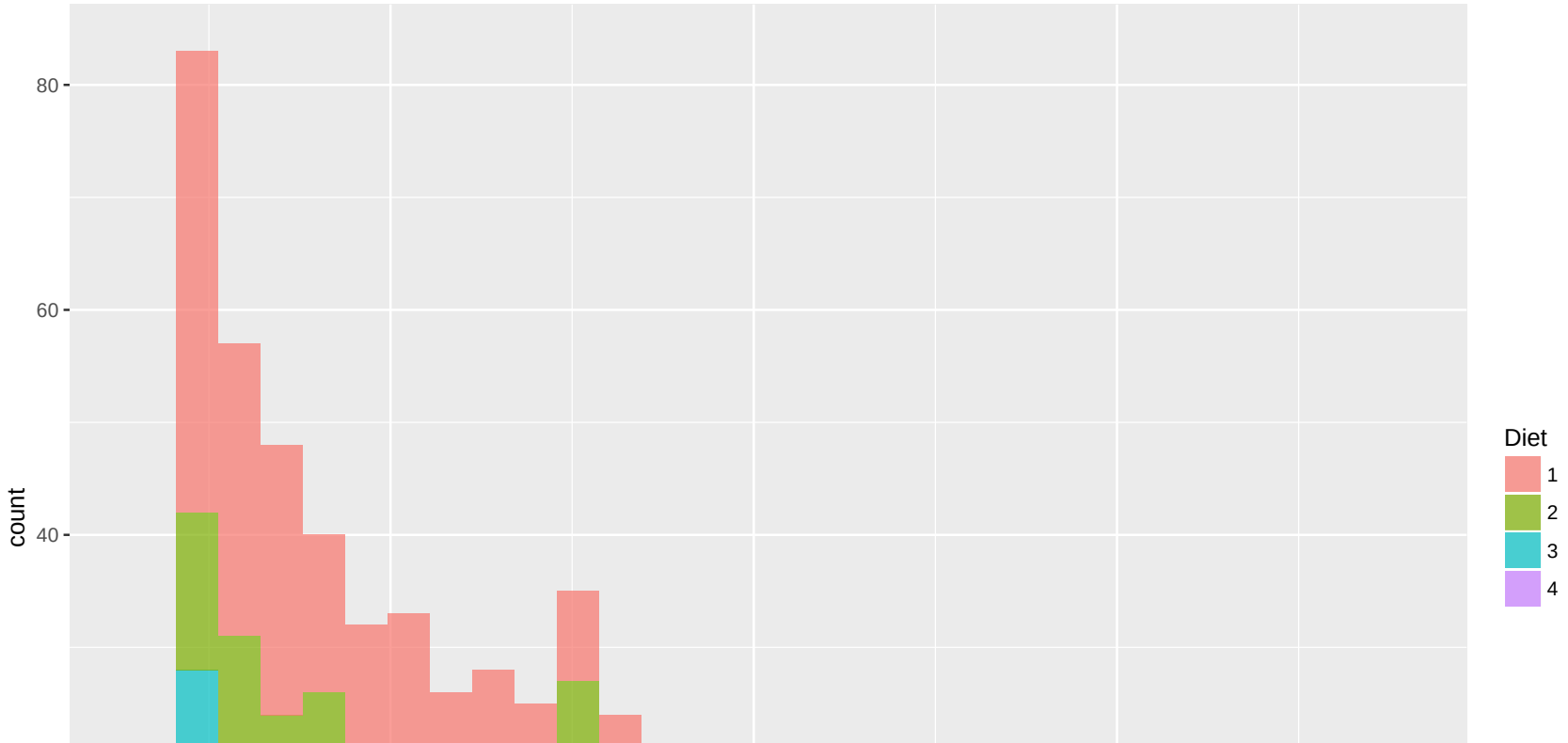


Multiple Histograms

Alpha refers tot he opacity of the color, less is

```
> qplot(x = weight, fill = Diet, data = ChickWeight,
+       geom = c("histogram"), alpha=I(.7))
```

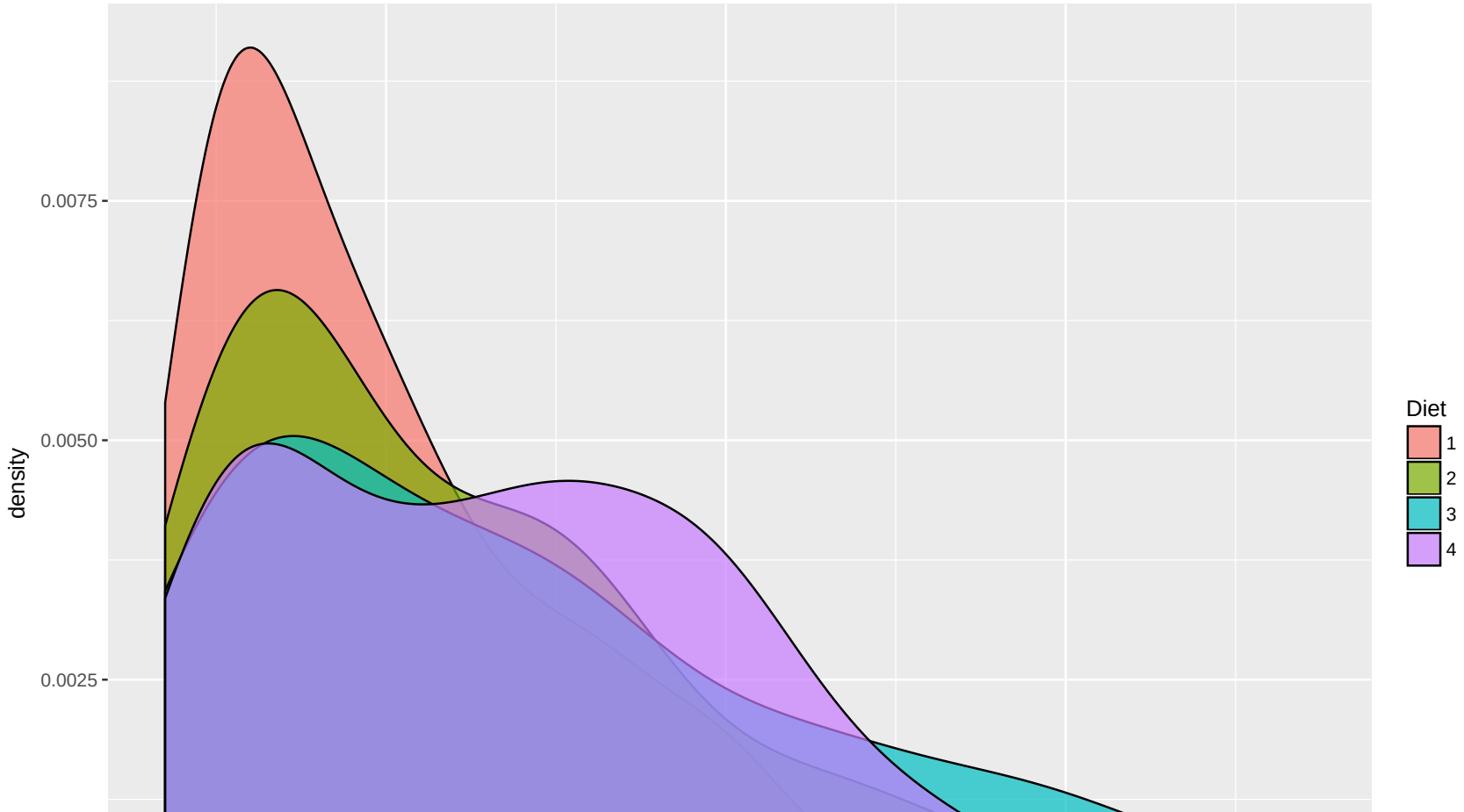
```
`stat_bin()` using `bins = 30`. Pick better value with `bi
```



Multiple Densities

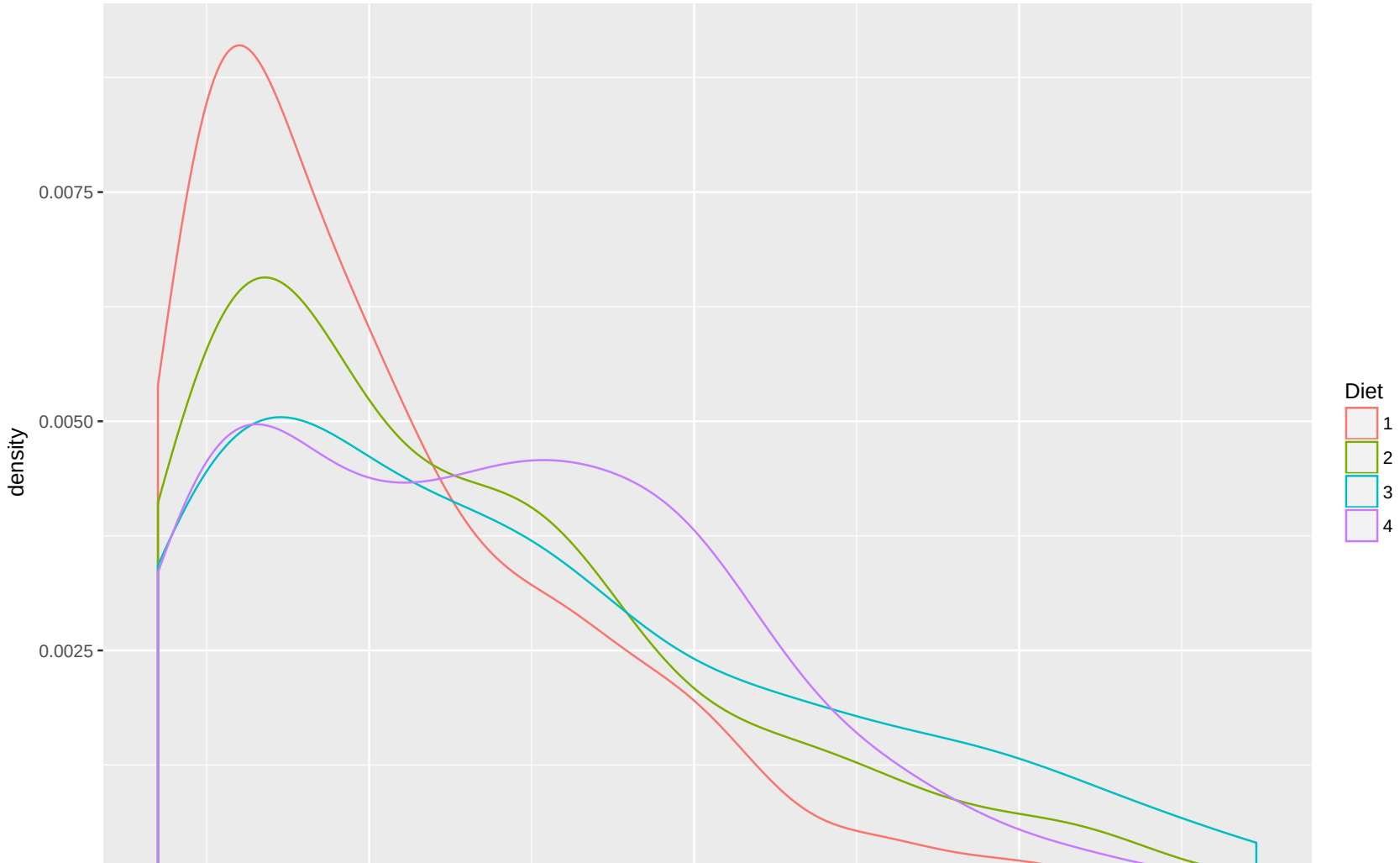
We cold also do densities

```
> qplot(x= weight, fill = Diet, data = ChickWeight,
+       geom = c("density"), alpha=I(.7))
```



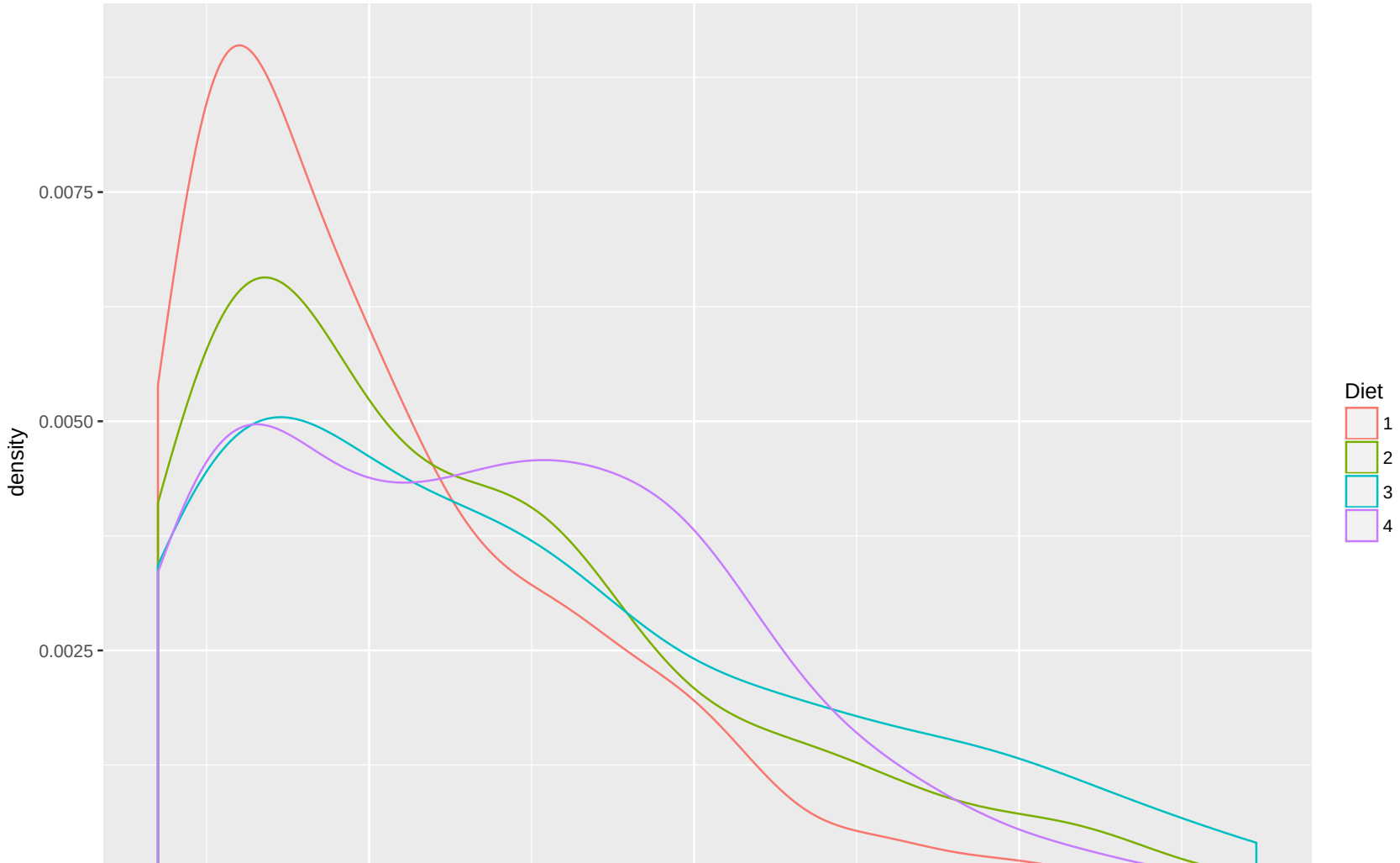
Multiple Densities

```
> qplot(x= weight, colour = Diet, data = ChickWeight,
+       geom = c("density"), alpha=I(.7))
```



Multiple Densities

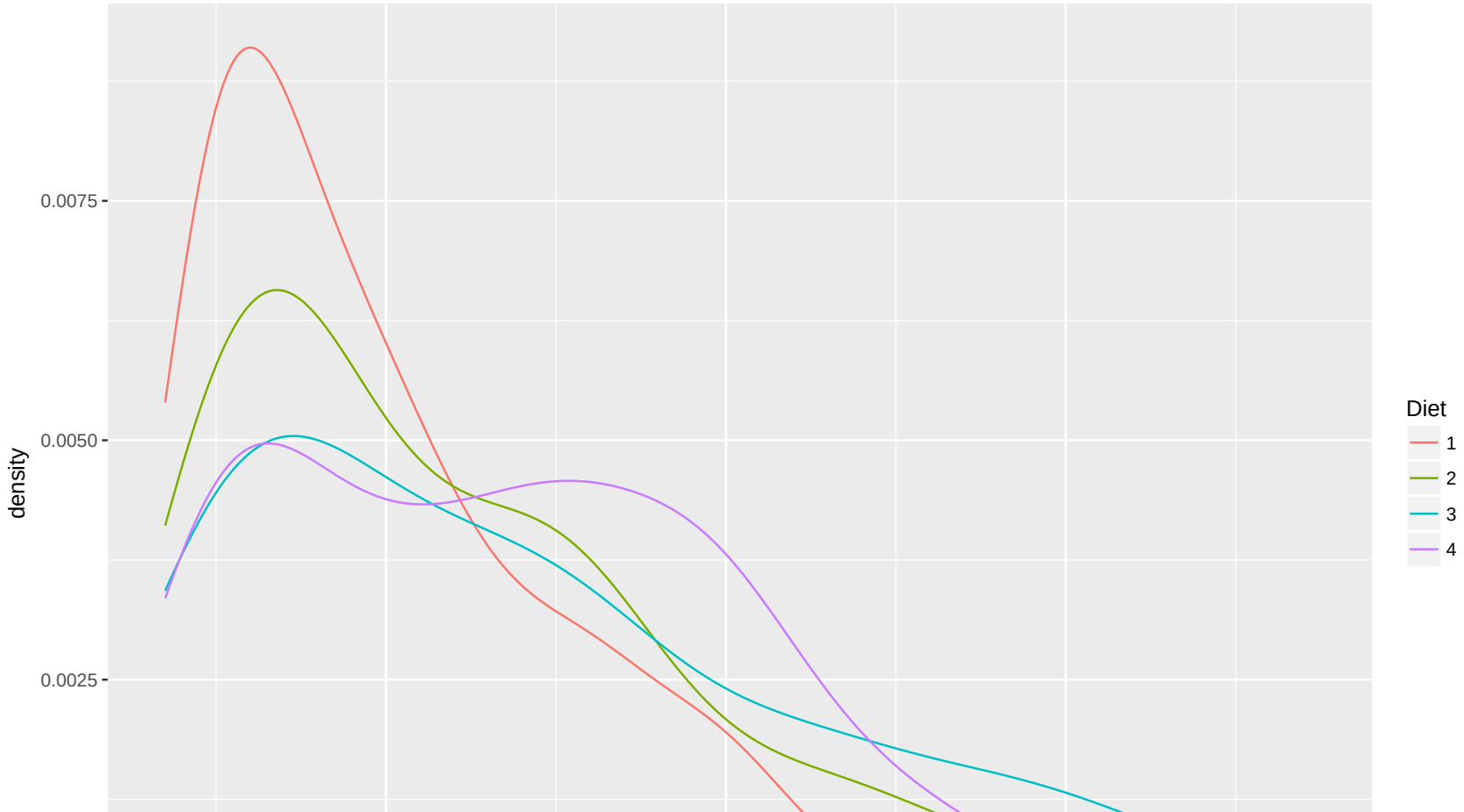
```
> ggplot(aes(x= weight, colour = Diet),
+   data = ChickWeight) + geom_density(alpha=I(.7))
```



Multiple Densities

You can take off the lines of the bottom like this

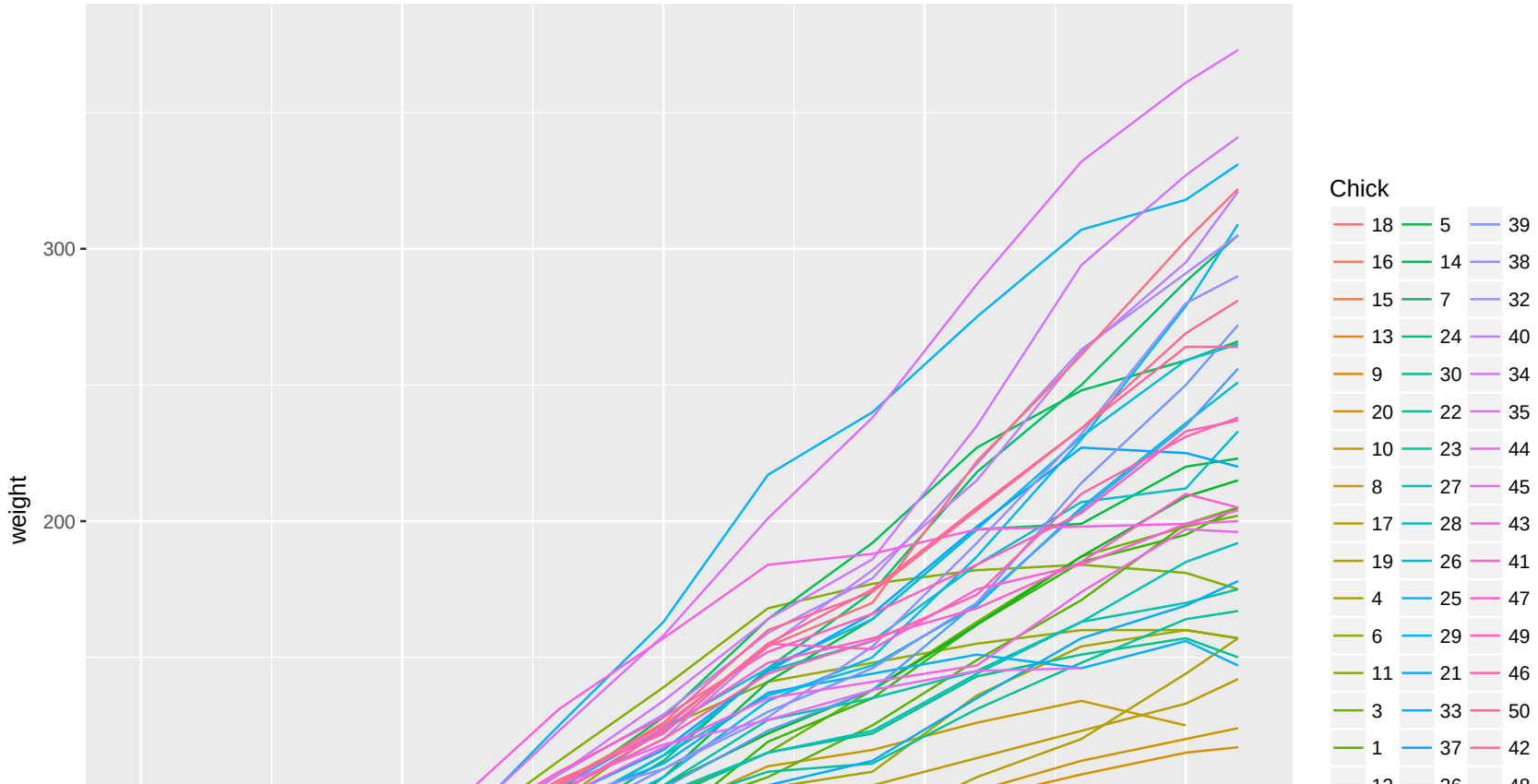
```
> ggplot(aes(x = weight, colour = Diet), data = ChickWeight
+   geom_line(stat = "density")
```



Spaghetti plot

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

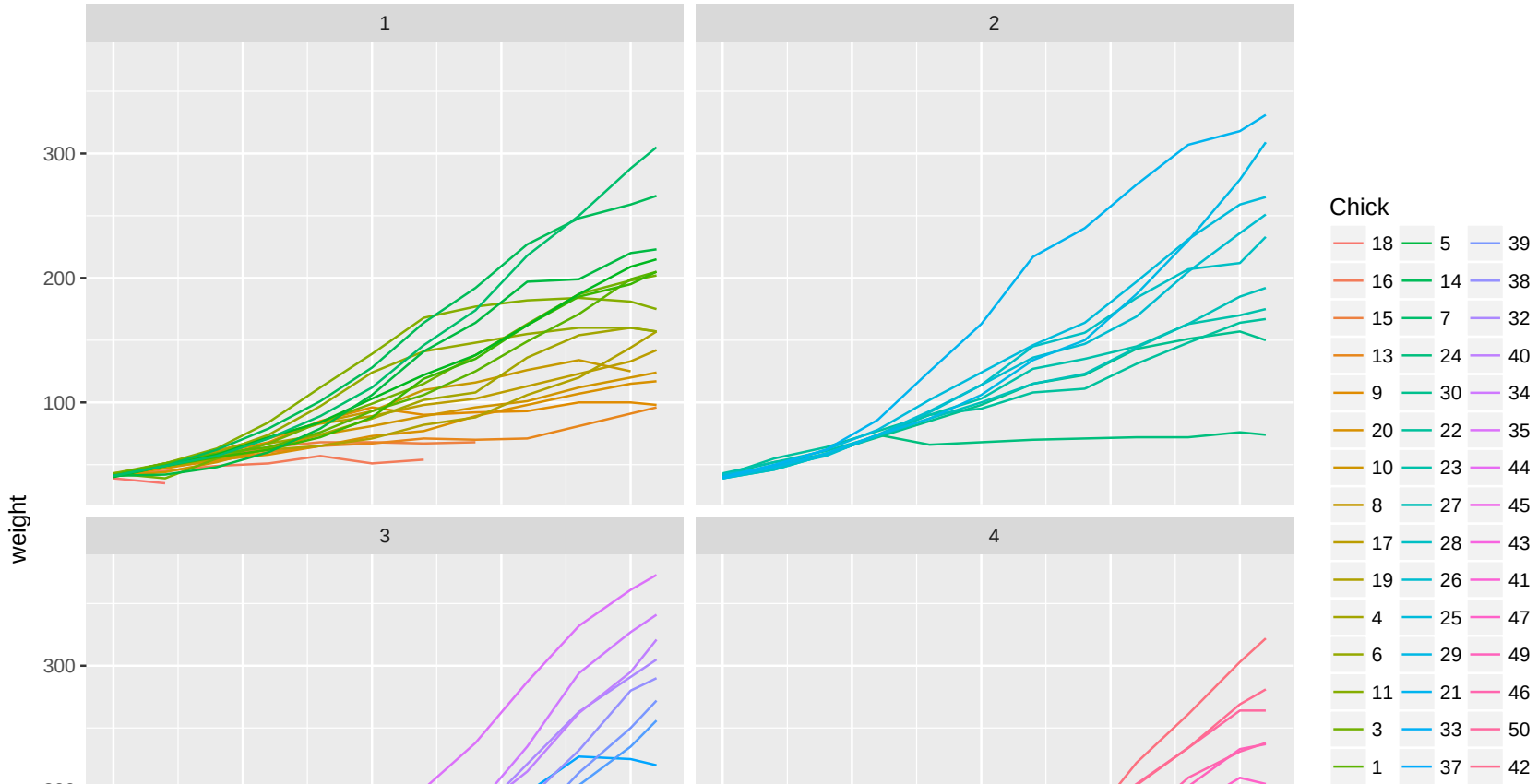
```
> qplot(x=Time, y=weight, colour = Chick,
+       data = ChickWeight, geom = "line")
```



Spaghetti plot: Facets

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

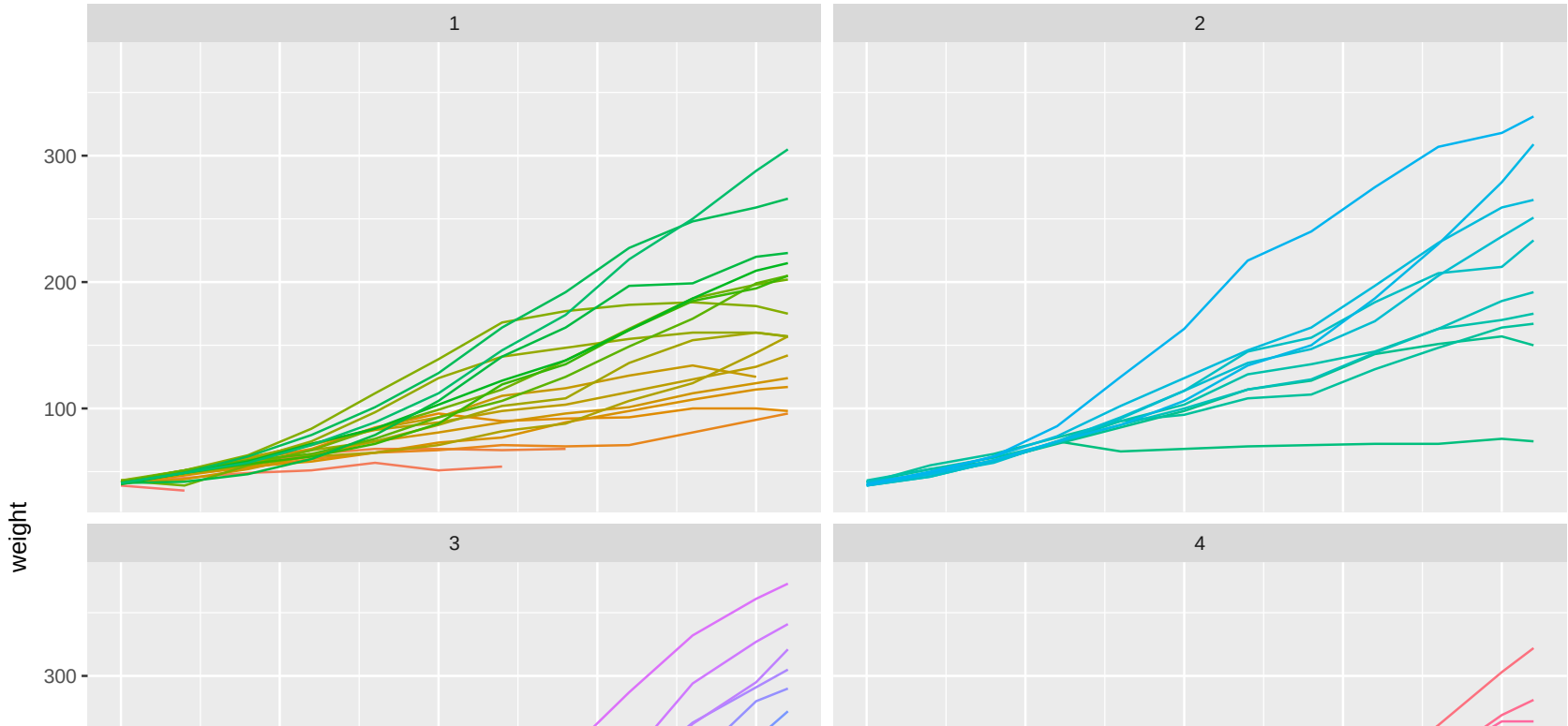
```
> qplot(x = Time, y = weight, colour = Chick,
+       facets = ~Diet, data = ChickWeight, geom = "line")
```



Spaghetti plot: Facets

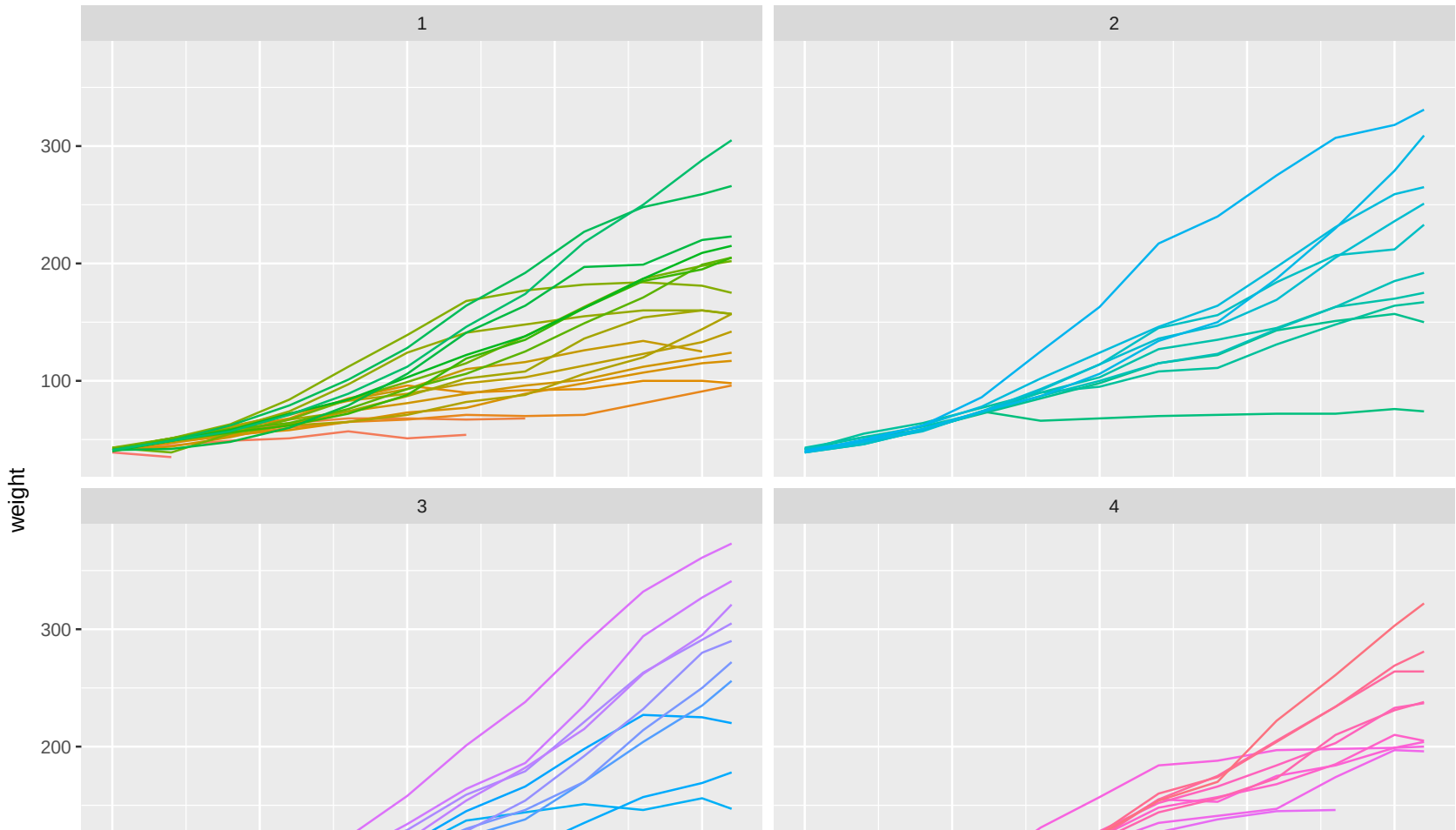
We can turn off the legend (referred to a "guide" in ggplot2). (Note - there is different syntax with the +)

```
> qplot(x=Time, y=weight, colour = Chick,
+       facets = ~ Diet,  data = ChickWeight,
+         geom = "line") + guides(colour=FALSE)
```



Spaghetti plot: Facets

```
> 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.

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

```
##         state  year deaths
## 1 Afghanistan X1760     NA
## 2     Albania X1760     NA
```

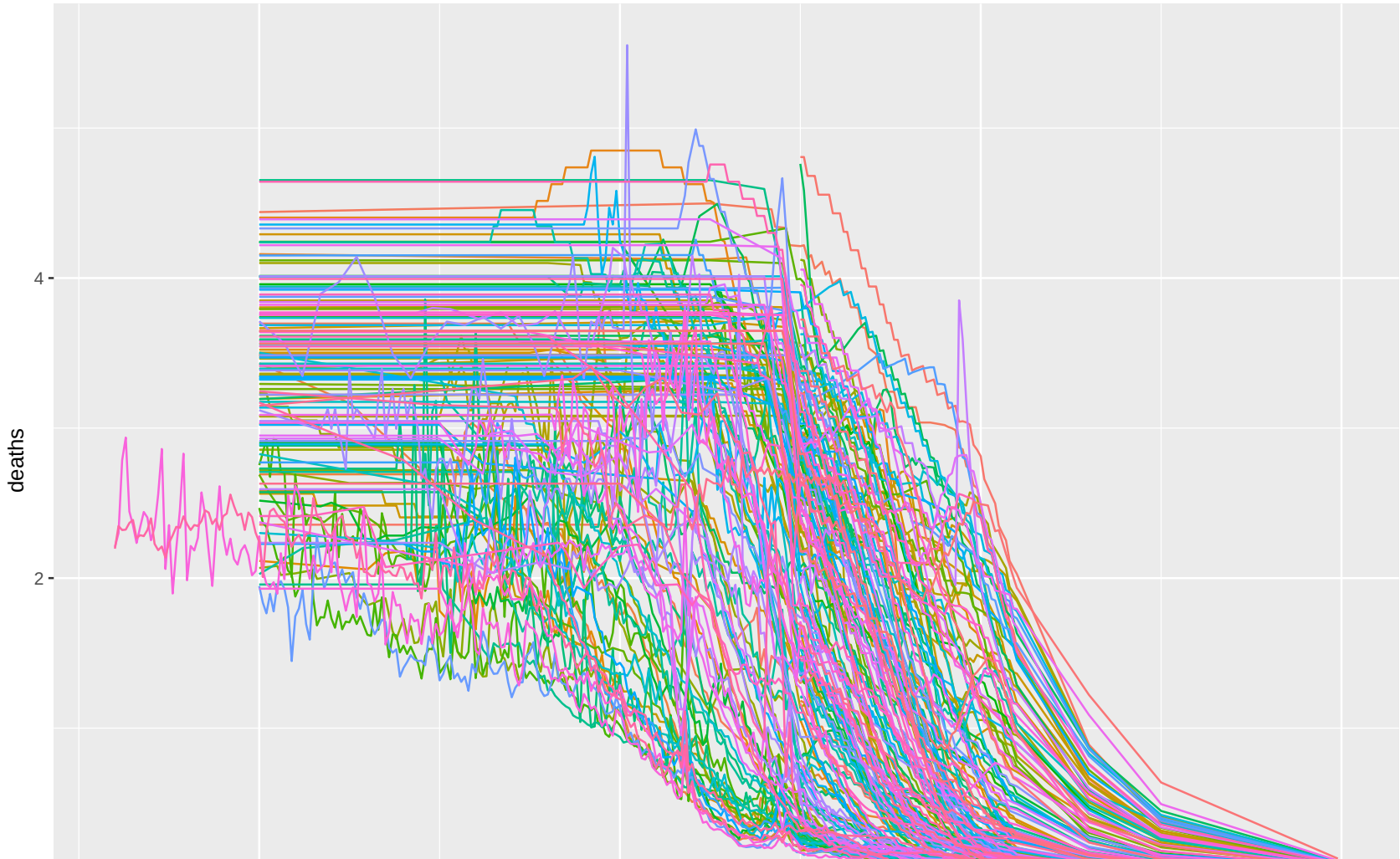
ggplot2

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

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

ggplot2

```
> 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:

```
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

```
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

```
dim(diamonds)
```

```
## [1] 53940    10
```

```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat       cut color clarity depth table price     x
##   <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl>
## 1  0.23     Ideal     E     SI2  61.5    55   326  3.95
## 2  0.21   Premium     E     SI1  59.8    61   326  3.89
## 3  0.23      Good     E     VS1  56.9    65   327  4.05
## 4  0.29   Premium     I     VS2  62.4    58   334  4.20
## 5  0.31      Good     J     SI2  63.3    58   335  4.34
## 6  0.24 Very Good     J    VVS2  62.8    57   336  3.94
```

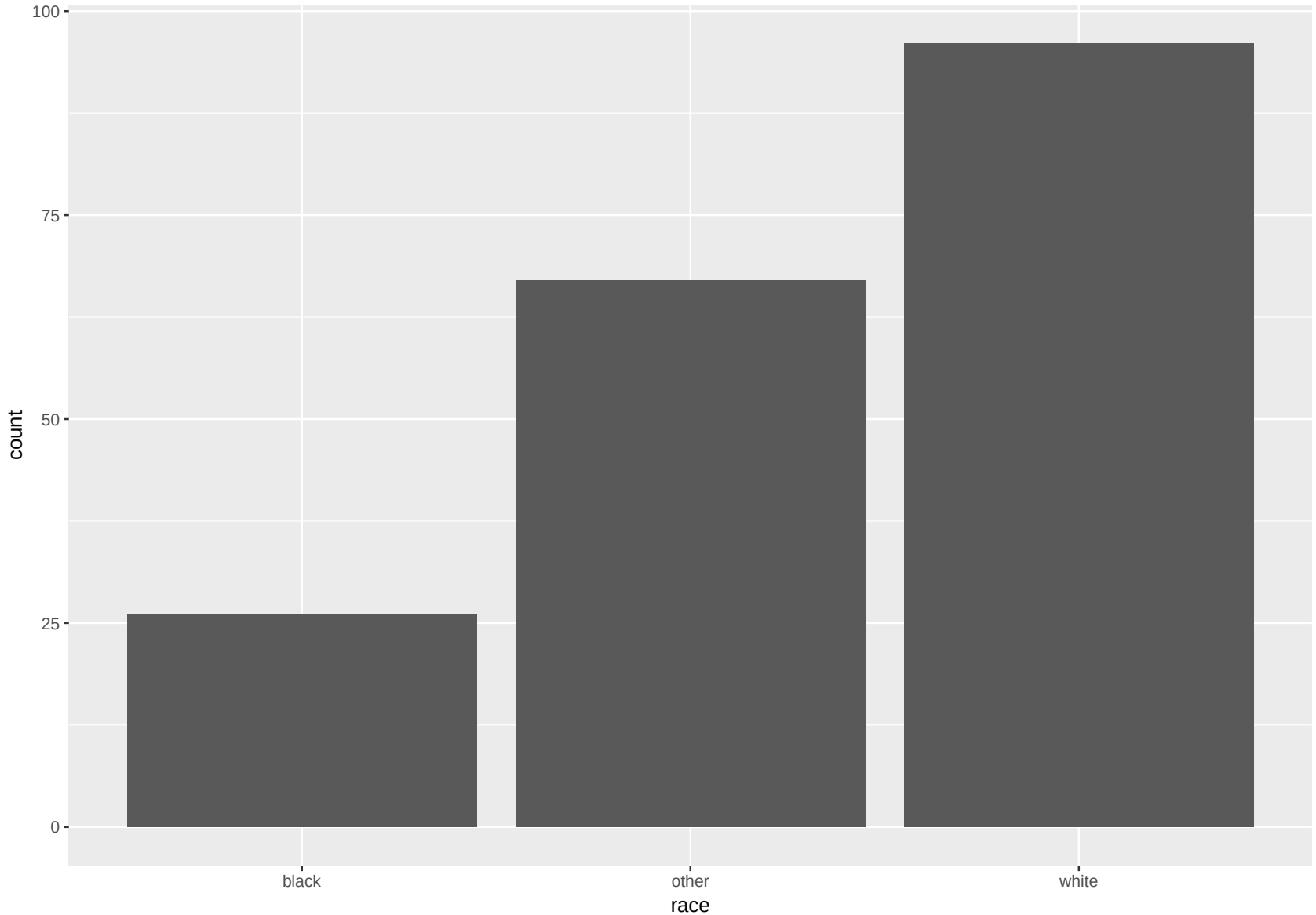
A barchart example

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.

```
# Assign better variable names
colnames(birthwt) <- c("birthwt.below.2500", "mother.age",
    "race", "mother.smokes", "previous.prem.labor", "hypert
    "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.smok
                                       c(0,1), c("no", "yes"))),
            hypertension = as.factor(mapvalues(hypertensio
                                       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(birthw
```

A bar chart

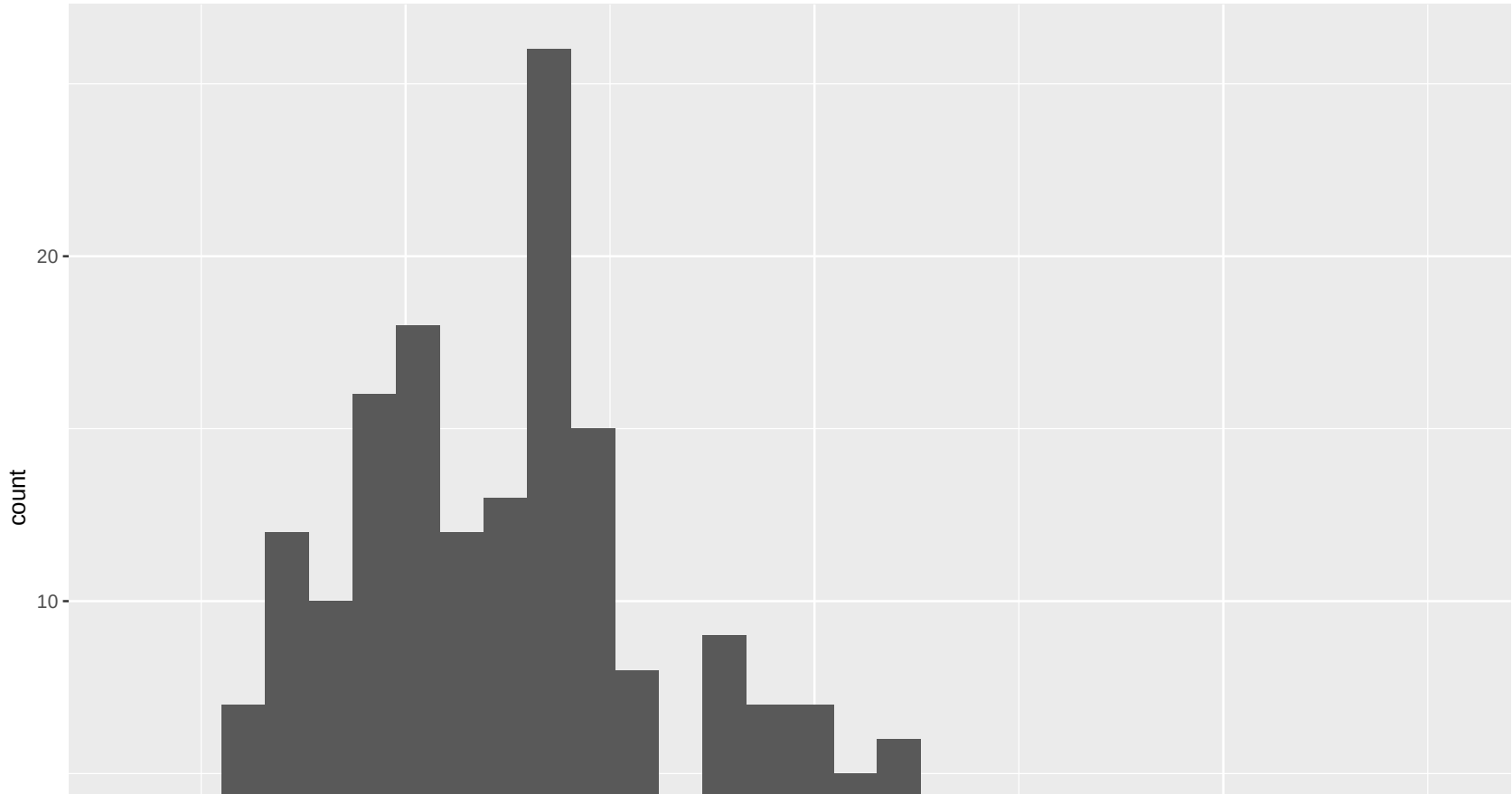
```
qplot(x = race, data = birthwt, geom = "bar")
```



Histograms and density plots

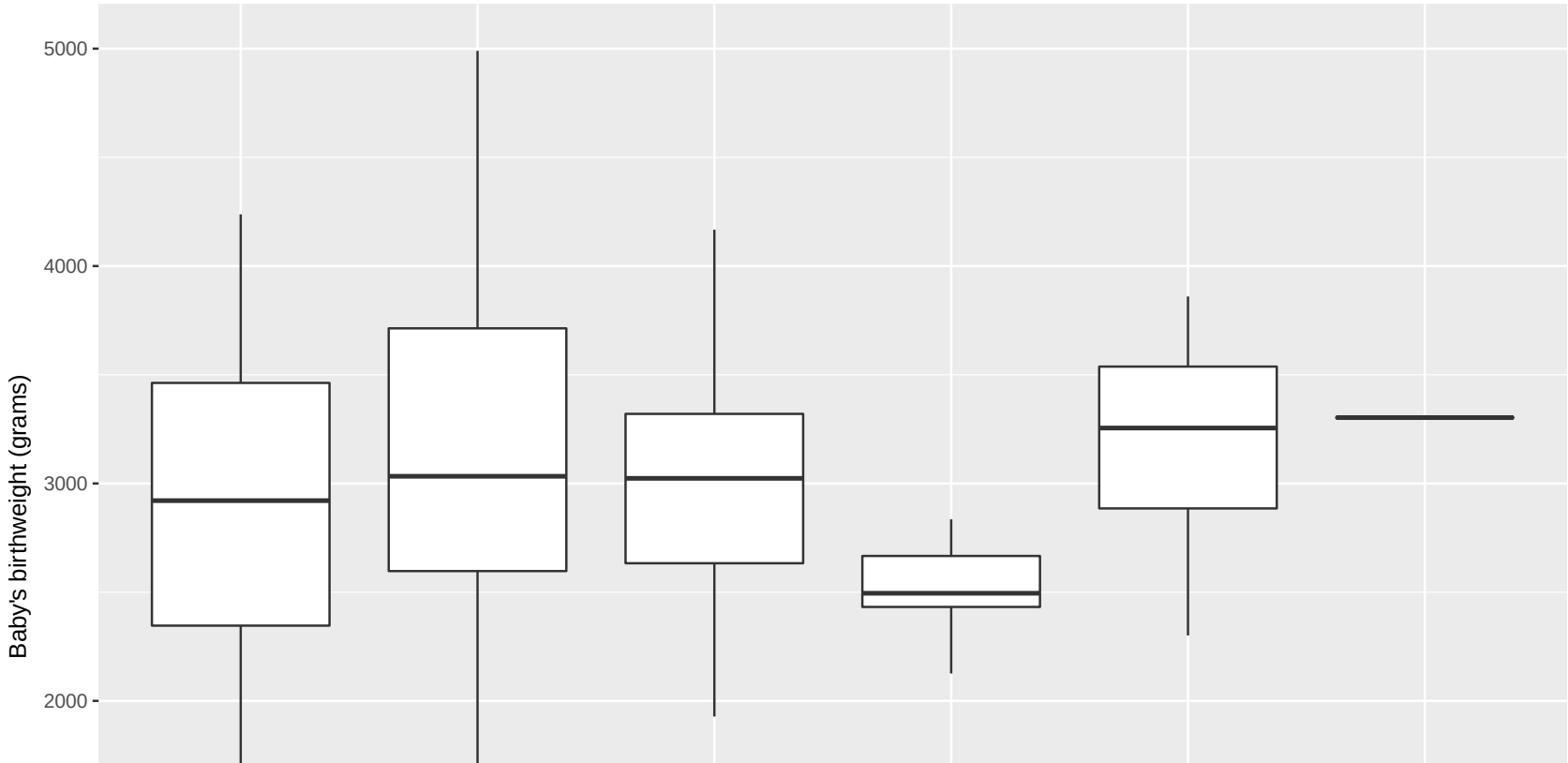
```
base.plot <- ggplot(birthwt, aes(x = mother.age)) +
  xlab("Mother's age")
base.plot + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `
```



Box plots and violin plots

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

# Box plot
base.plot + geom_boxplot()
```



Visualizing means

Previously we calculated the following table:

```
bwt.summary <- aggregate(birthwt.grams ~ race + mother.smok
bwt.summary
```

```
##    race mother.smokes birthwt.grams
## 1 black            no      2854.500
## 2 other            no      2815.782
## 3 white            no      3428.750
## 4 black           yes      2504.000
## 5 other           yes      2757.167
## 6 white           yes      2826.846
```

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

```
# Define basic aesthetic parameters
p.bwt <- ggplot(data = bwt.summary, aes(y = birthwt.grams,

# Pick colors for the bars
```

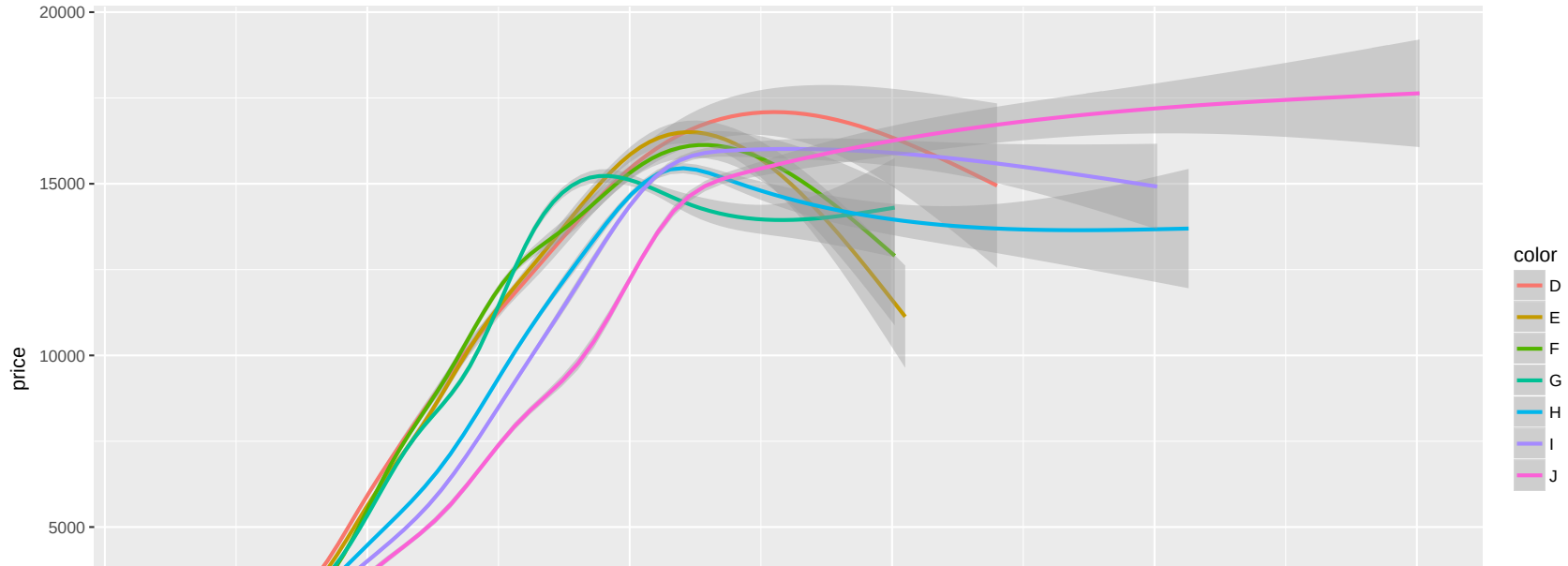
Statistical overlays in ggplot

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.

```
diamond.plot + stat_smooth()
```

```
## `geom_smooth()` using method = 'gam'
```

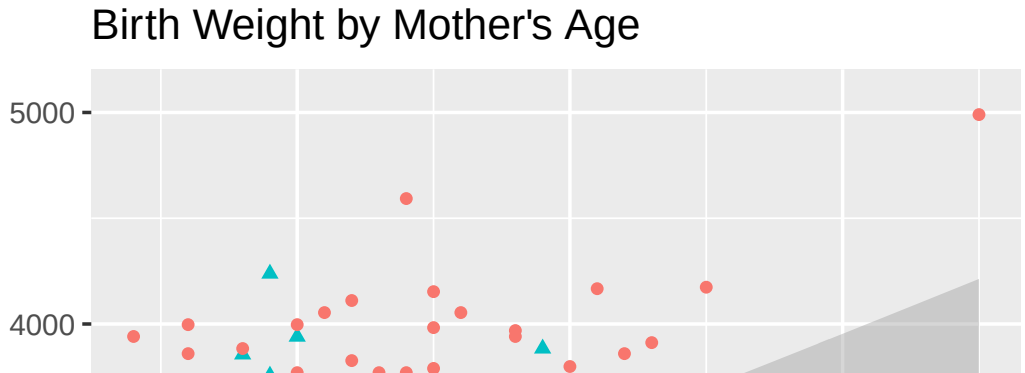


Birthweight regression overlay

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.

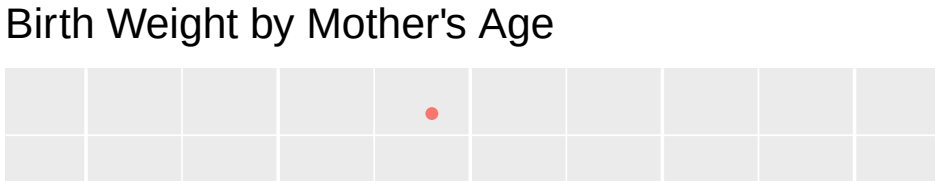
```
ggplot(birthwt, aes(x=mother.age, y=birthwt.grams, shape=mo
  geom_point() + # Adds points (scatterplot)
  geom_smooth(method = "lm") + # Adds regression lines
  ylab("Birth Weight (grams)") + # Changes y-axis label
  xlab("Mother's Age (years)") + # Changes x-axis label
  ggtitle("Birth Weight by Mother's Age") # Changes plot t
```



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.

```
# 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, shap
  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 t
```



ggplot2 also does maps

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

```
library(maps)
```

```
##
## Attaching package: 'maps'

## The following object is masked from 'package:plyr':
##
##     ozone
```

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

```
head(USArrests)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
```

map#1

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

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

```
## 'data.frame':    15537 obs. of  6 variables:
##  $ long      : num  -87.5 -87.5 -87.5 -87.5 -87.6 ...
##  $ lat       : num  30.4 30.4 30.4 30.3 30.3 ...
##  $ group     : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ order     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ region    : chr  "alabama" "alabama" "alabama" "alaban
##  $ subregion: chr  NA NA NA NA ...
```

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

# Convert everything to lower case
```