# A Short Guide to `ggplot2`

Political Science Y575
Indiana University
Last Compiled: February 29, 2016
Last Updated: January 9, 2015

Christopher D. DeSante, Ph.D.*

## Contents

**1 Why this guide?** **3**

**2 Introduction to `ggplot2`:** **3**

2.1 The Basic Grammar of Graphics: . . . 3

2.2 A Very Basic Plot: . . . 4

**3 The Most Commonly Used Geoms:** **5**

geom_point() . . . 5

geom_line() . . . 5

geom_jitter() . . . 5

geom_bar() . . . 6

geom_abline() . . . 6

geom_pointrange() . . . 6

geom_boxplot() . . . 7

geom_density() . . . 7

geom_errorbar() . . . 7

geom_text() . . . 8

geom_tile() . . . 8

**4 Other Useful Functions** **9**

4.1 Casting and Melting with `reshape2` . . . 9

Example 1: Means of Variables over Time . . . 10

Example 2: Casting/Melting with two variables: . . . 11

*Christopher DeSante is currently an Assistant Professor of Political Science at Indiana University in Bloomington, Indiana. He received his Ph.D. in Political Science (American Politics and Methodology) in 2012 from Duke University in Durham, NC. He can be reached via email at `cdesante@indiana.edu`. Most of the examples here can be found on the is.R Tumblr page, which also contains a number of fantastic examples by David B. Sparks, a fellow Duke Ph.D. and currently of the Boston Celtics. In fact, the first version of this guide was written in 2010, mostly as a way to keep all of my code to do certain things in one place so as to not pester David with so many questions.

Example 3: Adding `facets()` . . . 12
4.2 `ggsave()` . . . 17
4.3 `grid.arrange()` . . . 22

**5 Color in `ggplot2`: 23**
5.1 Custom Colors in `ggplot2`: . . . 23
scale_color_manual: . . . 23
scale_colour_gradientn . . . 24
scale_colour_brewer . . . 25

**6 Manipulating Text in `ggplot2`: 26**
6.1 Using Mathematical Expressions in Labels . . . 26
6.2 plotmath in `ggplot2` . . . 27
6.3 Custom Axis Breaks using `+ scale_y_continuous()` . . . 28
6.4 Textual Healing . . . 29
6.5 Working with `hjust()`, `vjust()` and `angle()` . . . 31

**7 The Legend(s) of `ggplot2` 32**
7.1 Troubleshooting Legends . . . 32
7.2 `ggplot()` vs. `qplot()` . . . 35

**8 Mapping Spatial Data 36**
8.1 Basic State-Level Map . . . 36
8.2 Projections . . . 38
8.3 Pulling Maps from the Web with `qmap()` . . . 39
8.4 Maps at the County Level using FIPS Codes . . . 41

# 1 Why this guide?

Plots are important and statistical graphics can convey a lot of information. On the other hand, plots can also be used to mislead readers or the masses to support whichever conclusion the statistician wishes (**??**). In your graduate training, details matter; the little things you think are not important (tables, graphics, presentation style, document style, etc.) all have *marginal effects* on conference presentations, paper publication and (ultimately) tenure.

The wise graduate student would look at the guide below as an nice, compact set of tools that can help them advance their research toward publication. The student who says something like, "why do I *need* to use R; why can't I make my charts in Excel or Stata or SPSS?" my reply is this: you don't need to do this, but a failure to do so serves as an indicator for the kind of standard to which you hold yourself. If your job is to write and publish professional papers, and someone gives you both the means and opportunity to make your papers appear more professional, then you should probably have the motivation to follow through with this process (or choose a different profession).

This monograph is solely to be used as an abbreviated guide to a single graphics package; I have compiled it because I was tired of having to look up all the code I'd used previously to solve a particular problem. Of course, there are other and better guides available both online – http://docs.ggplot2.org/current/ – and in print (**?**); also, since Hadley Wickham sometimes updates the `ggplot2` package, there's a non-zero chance none of this code works.

# 2 Introduction to `ggplot2`:

This guide uses the newer version of ggplot2, version 1.0 and the examples are run on R version R version 3.1.1 (2014-07-10) – "Sock it to Me". In the R console, it can be installed using:

```
install.packages("ggplot2", dependencies=TRUE)
library(ggplot2)
#the next line will remove the grey backgrounds
#and set the theme to black and white:
theme_set(theme_bw())
```

## 2.1 The Basic Grammar of Graphics:

All plots that you make using ggplot will utilize the following elements:

`data` – the data.frame you'll want to visualize;

`geometry` – the shapes you want to see in the plot;

`stat` – how you transform the data (counts, means, ranges, etc.) before you visualize it;

`scale/aes` – legends/axes/colors/sizes/shapes, these are the aspects that make it possible to "read" data values from the graphic;

`coordinates` – Cartesian (X/Y) or others (polar coordinates or mapping projections;

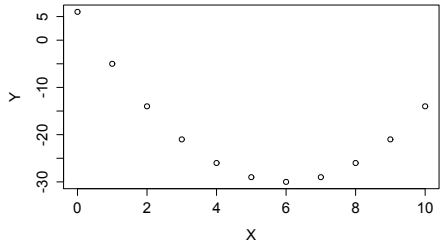
`facets` – subsetting the original data.frame in a way that makes it easier to present multiple dimensions or plots;

## 2.2 A Very Basic Plot:

To begin our plotting, let's make a very simple set of data and plot two variables against each other. One could think of this as simply plotting the equation $Y = X^2 - 12X + 6$ over (0,10). This is shown in figure 1 using the basic `plot()` command in base R.

```
X <- 0:10
Y <- (X^2 - 12*X + 6)
plot(X,Y)
```

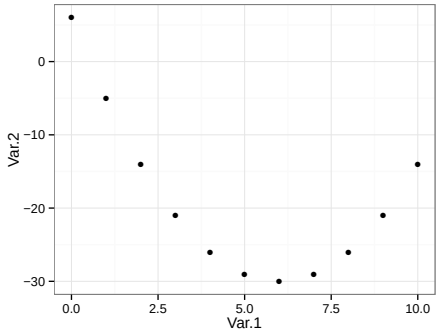
Figure 1: $Y = X^2 - 12X + 6$ over (0,10)




Below, the code to the left of the figure produces a `data.frame` consisting of one variable that is a function of the other (a sequence from 0 to 10 by 1). The `ggplot` command specifies the `data.frame` to look for the data in, and the "geom_point" command tells `ggplot2` to plot a series of points corresponding to the X and Y variables specified in the aes(thetics) portion of the command. Point is only one of many geoms (geometric objects) that one sees on the plot. Here are some examples of some others, though a full list can be seen at Hadley Wickham's ggplot2 website.

```
#This code produces figure 1:
Var.1 <- 0:10
Var.2 <- (Var.1^2 - 12*Var.1 + 6)
my.data <- data.frame(Var.1, Var.2)
my.data
#This line produces our first plot:
ggplot(data = my.data) + geom_point(aes(
x = Var.1, y = Var.2))
```

Figure 2: $Y = X^2 - 12X + 6$ over (0,10)

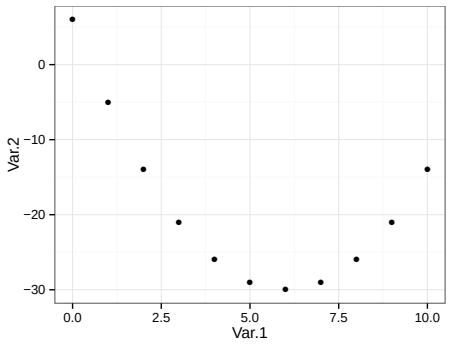


# 3 The Most Commonly Used Geoms:

### geom_point()

```
#This code produced figures 2 and 3:
Var.1 <- 0:10
Var.2 <- (Var.1^2 - 12*Var.1 + 6)
ggplot(data = my.data)+ geom_point(aes(x = Var.1, y = Var.2))
#Options for geom_line:
x, y, alpha, colour, linetype, size.
```

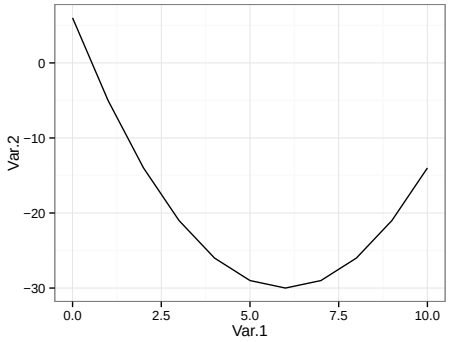
Figure 3: geom_point()




### geom_line()

```
#This code produces figure 4:
Var.1 <- 0:10
Var.2 <- (Var.1^2 - 12*Var.1 + 6)
ggplot(data = my.data) + geom_line(aes(x = Var.1, y = Var.2))
#Options for geom_line:
#x, y, alpha, colour, linetype, size.
```

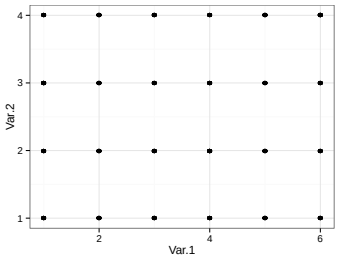
Figure 4: geom_line()




### geom_jitter()

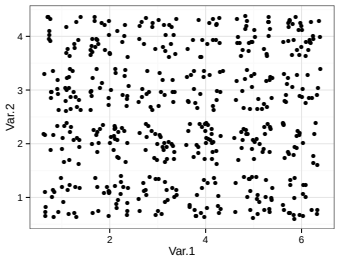
```
#This code produces figure 5:
#Notice the overplotting...
Var.1 <- sample(1:6, 500, replace = T)
Var.2 <- sample(1:4, 500, replace = T)
my.data <- data.frame(Var.1, Var.2)
ggplot(data = my.data) + geom_point(aes(x = Var.1,
                          y = Var.2))
```

Figure 5: Points without geom_jitter()




```
#This code produces figure 6:
#Notice how 'jitter' corrects overplotting...
Var.1 <- sample(1:6, 500, replace = T)
Var.2 <- sample(1:4, 500, replace = T)
my.data <- data.frame(Var.1, Var.2)
ggplot(data = my.data) + geom_jitter(aes(x = Var.1,
                          y = Var.2))
```

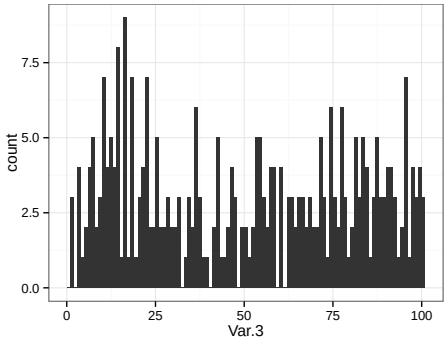
Figure 6: Points with geom_jitter()



### geom_bar()

```
#This code produced figure 7:
Var.3 <- sample(c(1:100), 300, replace=T)
bar.data <- data.frame(Var.3)
ggplot(data = bar.data) + geom_bar(aes(x = Var.3),
stat = "bin", binwidth=1)
#Options for geom_bar:
#x, y, alpha, colour, fill, linetype, size, weight.
```

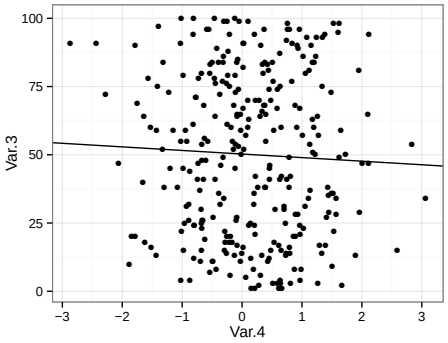
Figure 7: geom_bar()




### geom_abline()

```
#This code produced figure 8:
Var.3 <- sample(c(1:100), 300, replace=T)
Var.4 <- rnorm(300)
lm(Var.3 ~ Var.4)
#Intercept: 50.275, Slope = -1.31
abline.data <- data.frame(Var.3, Var.4)
AB <- ggplot(data = abline.data) + geom_point(aes(
  x = Var.4 , y = Var.3)) + geom_abline(
    intercept = 50.275, slope = -1.31)
#Options: alpha, colour, linetype, size
```

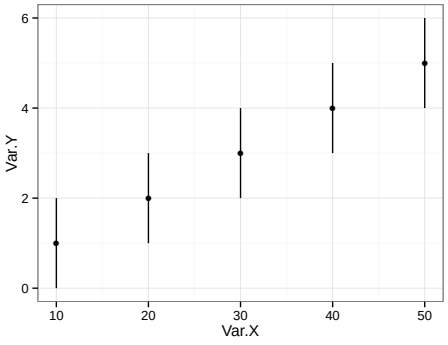
Figure 8: geom_abline()




### geom_pointrange()

```
#This code produces figure 9:
LOWS <- 0:4
Var.Y <- 1:5
HIGHS <- 2:6
Var.X <- c(10, 20, 30, 40, 50)
pointrange.data <- data.frame(LOWS, HIGHS,
    Var.Y, Var.X)
ggplot(data = pointrange.data
) + geom_pointrange(aes(x = Var.X, y = Var.Y,
      ymin = LOWS, ymax = HIGHS )  )
# x, y, ymin, ymax, alpha, colour,
# linetype, shape, size, fill
```

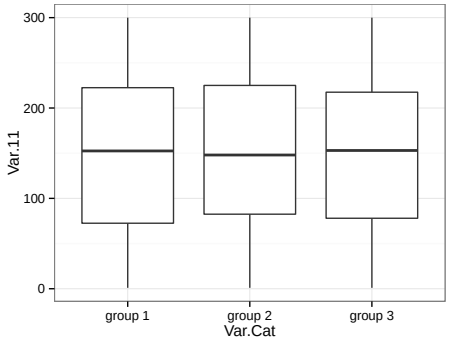
Figure 9: geom_pointrange()



### geom_boxplot()

```
#This code produced figure 10:
Var.11 <- sample(1:300, 1000, replace=T)
Var.Cat <- sample (c("group 1", "group 2",
"group 3"), 1000, replace=T)
my.data <- data.frame(Var.11, Var.Cat)
ggplot(data = my.data) + geom_boxplot(aes(
x = Var.Cat , y = Var.11))
#lower ,middle ,upper, x, ymax,
#ymin, alpha, colour, fill, linetype,
#shape, size, weight
```

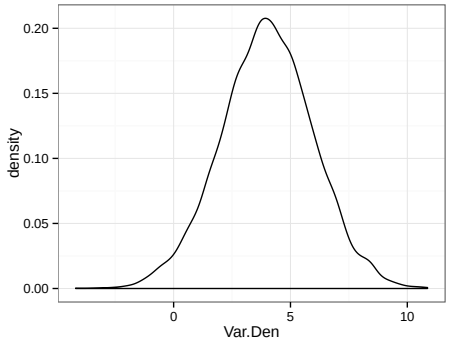
Figure 10: geom_boxplot()




### geom_density()

```
#This code produced figure 11:
Var.Den <- rnorm(10000, mean = 4, sd = 2)
Den.data <- data.frame(Var.Den)
ggplot(data = Den.data) + geom_density(
aes(x = Var.Den), stat="density")
#Aesthetics: x, y, alpha, colour,
#fill, linetype, size, weight
```

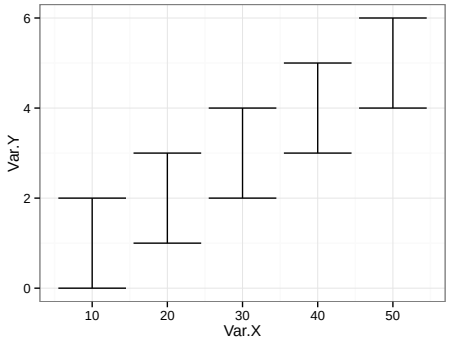
Figure 11: geom_density()




### geom_errorbar()

```
#This code produced figure 12:
LOWS <- 0:4
Var.Y <- 1:5
HIGHS <- 2:6
Var.X <- c(10, 20, 30, 40, 50)
errorbar.data <- data.frame(LOWS, HIGHS,
                    Var.Y, Var.X)
ggplot(data = errorbar.data
) + geom_errorbar(aes(x = Var.X, y = Var.Y,
         ymin = LOWS, ymax = HIGHS )  )
#Aesthetics: x, ymax, ymin, alpha,
# colour, linetype, size, width,
```

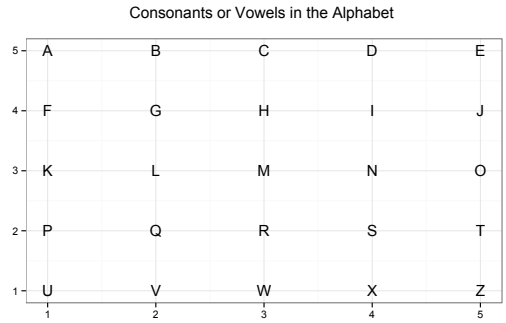
Figure 12: geom_errorbar()



### geom_text()

```
#This code produced figure 12:
X <- rep(1:5, 5)
Y <- rep(5:1, each=5)
Labels <- LETTERS[-25]
qplot(x = X, y = Y, geom = "text",
      label = Labels, xlab ="", ylab="",
      main = "Consonants or Vowels in the Alphabet \n")
```

Figure 13: geom_text()

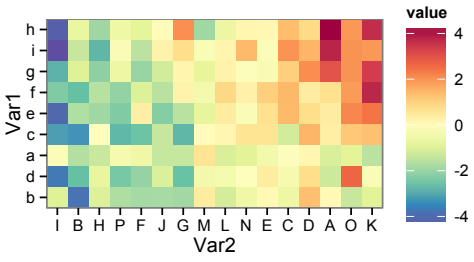



As we've seen, the plots above are all very basic and lack the use of `ggplot2`'s many other options; our final example was made by David Sparks and is an example of a heatmap; it uses the reshape package discussed in the next section. Conceptually, a heatmap shows the correlation matrix between some set of variables. As David notes, it features the lovely diverging "spectral" palette from Colorbrewer; reorders two factor variables (displayed along the axes), and brings in custom colors. This was originally posted to the is.R Tumblr by David on 9/27/2012.

### geom_tile()

```
library(reshape2)
library(RColorBrewer)
nRow <- 9
nCol <- 16

myData <- matrix(rnorm(nRow * nCol), ncol = nCol)
rownames(myData) <- letters[1:nRow]
colnames(myData) <- LETTERS[1:nCol]

# Replace with numbers that actually have a relationship:
for(ii in 2:ncol(myData)){
  myData[, ii] <- myData[, ii-1] + rnorm(nrow(myData))  }
for(ii in 2:nrow(myData)){
  myData[ii, ] <- myData[ii-1, ] + rnorm(ncol(myData))  }

# For melt() to work seamlessly, myData has to be a matrix.
longData <- melt(myData)
head(longData, 20)

# Optionally, reorder both the row and column variables in any order
# Here, they are sorted by mean value
longData$Var1 <- factor(longData$Var1,
names(sort(with(longData, by(value, Var1, mean)))))
longData$Var2 <- factor(longData$Var2,
names(sort(with(longData, by(value, Var2, mean)))))

# Define palette
myPalette <- colorRampPalette(rev(brewer.pal(11, "Spectral")), space="Lab")

zp1 <- ggplot(longData,
              aes(x = Var2, y = Var1, fill = value))
zp1 <- zp1 + geom_tile()
zp1 <- zp1 + scale_fill_gradientn(colours = myPalette(100))
zp1 <- zp1   theme_bw()
print(zp1)
```

Figure 14: geom_tile() "Heatmap"



# 4 Other Useful Functions

## 4.1 Casting and Melting with reshape2

The fundamentals for the package we will use, **reshape2**, are explained in detail by **?**.

```
ANES <- read.csv("http://pages.iu.edu/~cdesante/ANES.csv")
head(ANES)
ANES$caseid <- 1:dim(ANES)[1]
head(ANES)
dim(ANES)
```

The output is shown below in figure 15:

```
> head(ANES)
  year age cohort female race6 religion dems ftwelfare ftpoor ftaliens ftyoung pid7 trust ideo7 inerrant south dempres
1 1948  NA      7      0     1        1   NA        NA     NA       NA       NA   NA    NA    NA       NA    NA      NA
2 1948  NA      7      1     1        1   NA        NA     NA       NA       NA   NA    NA    NA       NA    NA      NA
3 1948  NA      6      1     1        2   NA        NA     NA       NA       NA   NA    NA    NA       NA    NA      NA
4 1948  NA      7      1     1        1   NA        NA     NA       NA       NA   NA    NA    NA       NA    NA      NA
5 1948  NA      6      0     1        2   NA        NA     NA       NA       NA   NA    NA    NA       NA    NA      NA
6 1948  NA      7      1     1        1   NA        NA     NA       NA       NA   NA    NA    NA       NA    NA      NA
```

Figure 15: Output

Now we can see we have 18 different variables loaded: year, age, cohort, female, race6, religion, dems, ftwelfare, ftpoor, ftaliens, ftyoung, pid7, trust, ideo7, inerrant, south, dempres, and caseid. Loading relevant packages (ggplot2, reshape2 and gridExtra), we can then take a look at what the `melt()` function does through the reshape package. Melting essentially reshapes the data into a new `data.frame` based on whichever variables you specify in the id="" option; this should be dictated by which variables you wish to analyze. If we are only interested in time trends in the ANES, we can just melt by year:

```
library(ggplot2)
library(gridExtra)
library(reshape)
anes.year <- melt(ANES, id=c("year"), na.rm=TRUE)
head(anes.year)
```

The output is shown below in figure 16:

```
> anes.year <- melt(ANES
> head(anes.year)
    year variable value
663 1952      age    25
664 1952      age    33
665 1952      age    26
666 1952      age    63
667 1952      age    66
668 1952      age    48
```

Figure 16: Output

`anes.year` has dimensions (619169 x 3). The dimensions are such that for every N variables (with complete cases) in the ID option, you return a `data.frame` that has N+2 columns and rows equal to the number of rows in the original data times the number of variables. This is a very useful way to transform data. If, for example, we want to see differences by year by gender, we can melt by year and 'female.'

```
anes.year.gender <- melt(ANES, id=c("year", "female"), na.rm=TRUE)
head(anes.year.gender)
dim(anes.year.gender)
```

So, step one is to melt according to variables you plan on using; the next step is to CAST the data in such a way that extracts useful quantities that you can then use to plot/display/summarize.

### Example 1: Means of Variables over Time

For simplicity's sake, let's truncate the data.frame so that it only includes measures of partisanship and ideology as well as the year of the survey. This is done in three steps; the first of which is optional but may help keep your R workspace organized.

1. Subsetting the data to only include our variables of interest;
2. MELT the data using year as the identifying (id) variable (figure 17 shows the resulting output);
3. CAST this data so that we can get the means over time (figure 18).

```
#Step 1 : Subset
party.and.ideology<- ANES[,c(1, 12, 14)]
head(party.and.ideology)

#Step 2 : Melt
party.and.ideology.year <- melt(party.and.ideology,
id=c("year"), na.rm=TRUE)
head(party.and.ideology.year)

#Step 3 : Cast using dcast()
party.and.ideology.means.over.time <- dcast(
         party.and.ideology.year,
                      year~variable, mean, na.rm=T)

party.and.ideology.means.over.time
head(party.and.ideology.means.over.time)
```

Figure 17: Output after sub-setting and `melt()`

```
> party.and.ideology<- ANES[,c(1, 12, 14)]
> head(party.and.ideology)
  year pid7 ideo7
1 1948   NA    NA
2 1948   NA    NA
3 1948   NA    NA
4 1948   NA    NA
5 1948   NA    NA
6 1948   NA    NA
> party.and.ideology.year <- melt(party.and.ideology,
+ id=c("year"), na.rm=TRUE)
> head(party.and.ideology.year)
    year variable value
663 1952     pid7     5
664 1952     pid7     4
665 1952     pid7     3
666 1952     pid7     6
667 1952     pid7     6
668 1952     pid7     2
```

Figure 18: Final Casted Output

```
> head(party.and.ideology.means.over.time)
  year     pid7 ideo7
1 1952 2.470693   NaN
2 1954 2.454963   NaN
3 1956 2.659763   NaN
4 1958 2.403039   NaN
5 1960 2.545936   NaN
6 1962 2.505255   NaN
>
```

Now we can plot a relatively complex computation very easily, which figure 19 shows.

```
P1 <- ggplot(data = party.and.ideology.means.over.time
) + geom_point(aes(x = year, y = pid7))
P1  + ylim(0,6)  + labs(
title="Partisanship over Time (ANES)")
```

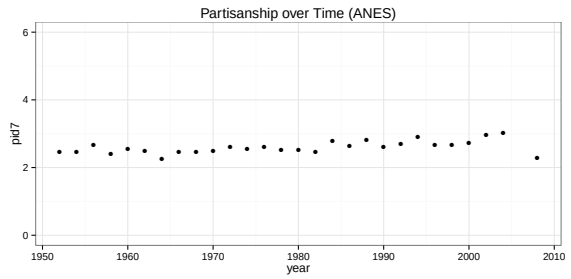



Figure 19: Quick plot after `melt()` and `dcast()`

**Example 2: Casting/Melting with two variables:**

Imagine instead we want to look at how Partisanship has shifted over these years between the South/Non-south. If we begin with just our ANES data frame, this might seem like quite a daunting task. However, if we know how to melt/cast appropriately, it is easier... First step, again, is to restrict our data to only the columns we want to utilize in our analysis. This isn't necessary, but let's keep things as simple as possible for now.

```
ANES <- read.csv("http://pages.iu.edu/~cdesante/ANES.csv")
party.and.region <- ANES[,c(1, 12, 16)]
head(party.and.region)
                  #Note the additional changes to our id=():
                  #I use complete.cases() to remove the NA values.
party.and.region.year <- melt(party.and.region[complete.cases(party.and.region),],
id=c("year", "south"), na.rm=TRUE)
head(party.and.region.year)
#Note the additional changes to our formula (below):
party.region.time  <- dcast(party.and.region.year, year+south~variable, mean, na.rm=TRUE)
party.region.time
head(party.region.time)
```

```
> head(party.region.time)
  year south     pid7
1 1952     0 2.783723
2 1952     1 1.323204
3 1956     0 2.935858
4 1956     1 1.757576
5 1958     0 2.651472
6 1958     1 1.607903
```

Figure 20: head(party.region.time)

```
                #Note the additional changes to our formula (below):
party.region.time  <- (cast(party.and.region.year, year+south~variable, mean, na.rm=T) )
party.region.time
dim(party.region.time)
P2 <- ggplot(data = party.region.time) + geom_point(aes(x = year,
                  y = pid7, colour=as.factor(south)))
P2 <-  P2  + ylim(0,6)  + labs(title="Partisanship over Time by Region")
```

Below, figure 21 shows the output:

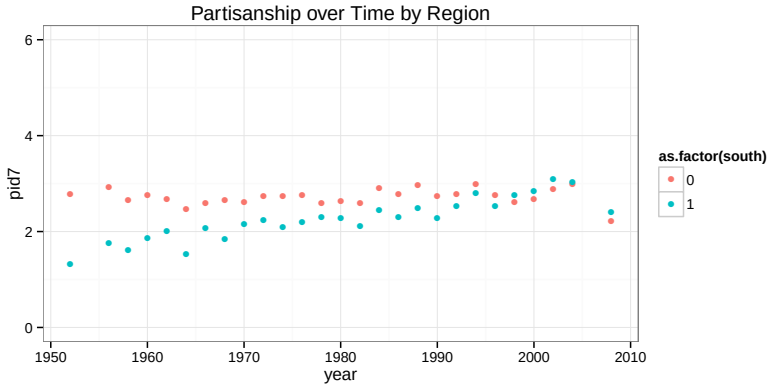



Figure 21: Plot after melting and casting by two variables

### Example 3: Adding `facets()`

While a few other examples can be found on the is-r blog, the next logical step is to add one more variable to the mix and change the display using facets. To stick with our previous examples, what if we wanted to look at the change in partisanship over time by both sex and region? Now we have four variables of interest: year, pid7, female and south.

```
party.gender.region <- ANES[,c(1, 4, 12, 16)]
head(party.gender.region)
#Note the additional changes to our id=():
party.gender.region.year <- melt(party.gender.region, id=c("year", "female", "south"), na.rm=TRUE)
head(party.gender.region.year)
#Note the additional changes to our formula (below):
party.gender.region.time  <- dcast(party.gender.region.year, year+female+south~variable, mean, na.rm=T)
#given some NAs for gender, I'm just going to clean these up quickly
party.gender.region.time <- party.gender.region.time[complete.cases(party.gender.region.time),]
head(party.gender.region.time)
#What party.gender.region.time represents is a matrix with four means per year,
# based on gender and region. So, for each year there's one for men in the south,
# men in the non-south, as well as women in each of the two regions.
#Throwing this into ggplot gives us a nice way to choose how to display this information.
```

Below, figure 22 shows the resulting melted casted, and cleaned data:

```
> head(party.gender.region.time)
  year female south     pid7
1 1952      0     0 2.744783
2 1952      0     1 1.100000
3 1952      1     0 2.818182
4 1952      1     1 1.500000
7 1956      0     0 2.810580
8 1956      0     1 1.491713
```

Figure 22: head(party.gender.region.time)

```
P3 <- qplot(data = party.gender.region.time, geom = "line",
            x = year, y = pid7,   color = factor(female) )
```

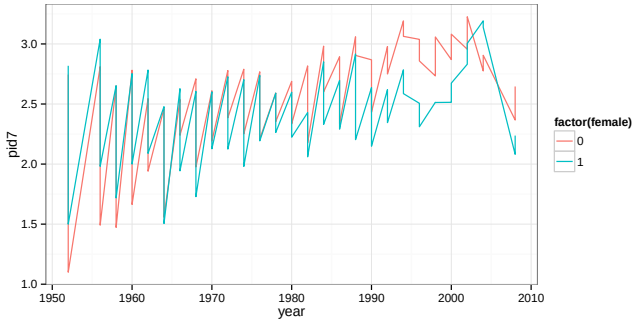


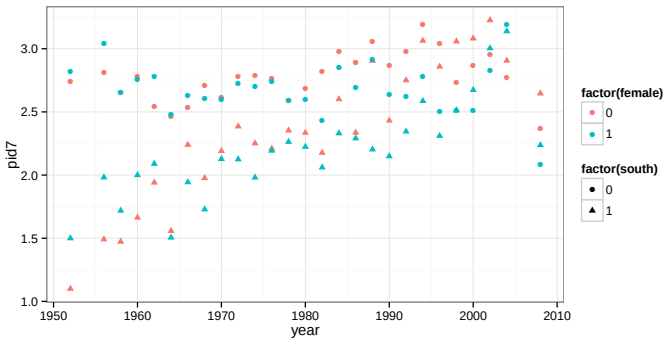

Figure 23: `P3` – What's wrong with this?

Figure 23 suffers from "over plotting;" the data are shown as zig-zags because for each year and sex there are two points being coerced into making a line. There are several ways of cleaning this up.

### Option 1: We could use `geom_point()` and add an aesthetic like shape

```
P3b <- qplot(data = party.gender.region.time,
geom = "point",
    x = year, y = pid7,
    shape = factor(south),
    color = factor(female) )
```

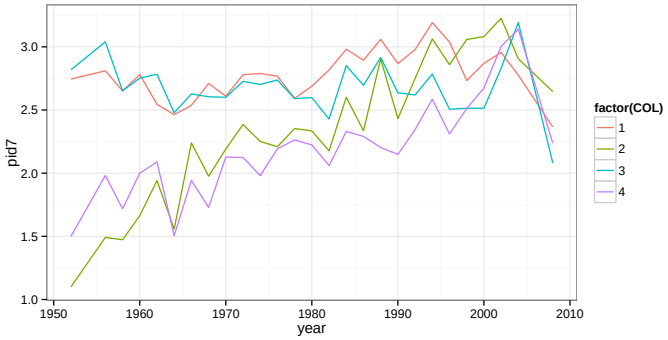
Figure 24: P3b




### Option 2: We could use custom colors

```
#NOTE: I've truncated party.gender.region.time
party.gender.region.time$COL <- c()
p...time$COL[party.gender.region.time$female==0 &
 party.gender.region.time$south==0 ] <- 1
p...time$COL[party.gender.region.time$female==0 &
     party.gender.region.time$south==1 ] <- 2
p...time$COL[party.gender.region.time$female==1 &
        party.gender.region.time$south==0 ] <- 3
p...time$COL[party.gender.region.time$female==1 &
    party.gender.region.time$south==1 ] <- 4
P3c <- qplot(data = party.gender.region.time,
             geom = "line",
             x = year, y = pid7,
             color = factor(COL) )
```

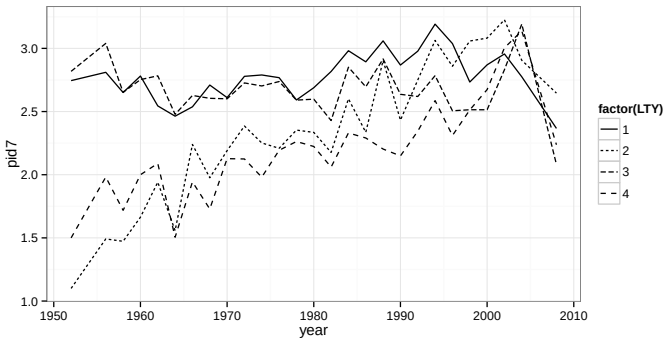
Figure 25: P3c




### Option 3: Change the linetype

```
#NOTE: I've truncated party.gender.region.time
party.gender.region.time$LTY <- c()
p...time$LTY[party.gender.region.time$female==0 &
 party.gender.region.time$south==0 ] <- 1
p...time$LTY[party.gender.region.time$female==0 &
     party.gender.region.time$south==1 ] <- 2
p...time$LTY[party.gender.region.time$female==1 &
        party.gender.region.time$south==0 ] <- 3
p...time$LTY[party.gender.region.time$female==1 &
    party.gender.region.time$south==1 ] <- 4
P3c <- qplot(data = party.gender.region.time,
             geom = "line",  x = year, y = pid7,
             linetype = factor(LTY) )
```

Figure 26: P3d

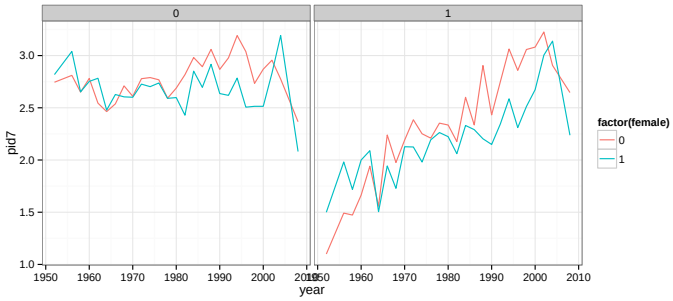


**Solution: add facets()**

Facets allow us to make different panels within a single plot, illustrating our desired effects while maintaining both clarity and appropriate legends/labels. Oftentimes it doesn't require any new variables to be created. The output, which makes two panels on the basis of region `facets = ~south)`, is shown below in figure 27.

```
P3x <- qplot(data = party.gender.region.time,
geom = "line",  x = year,
y = pid7,   color = factor(female) ,
facets = ~south)
```

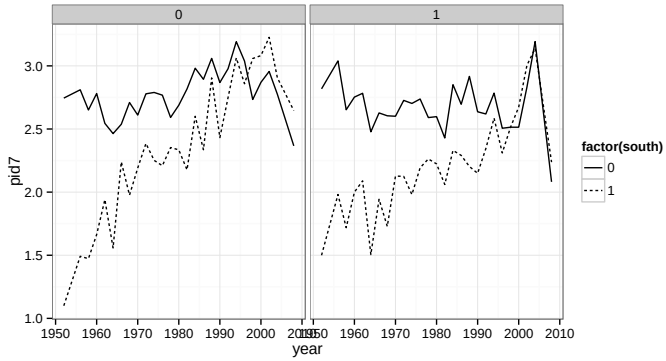
Figure 27: `facets = ~south`




Alternatively, we could facet based on sex, and change the linetype to indicate region (figure 28).

```
P3y <- qplot(data = party.gender.region.time,
geom = "line",   x = year, y = pid7,
        linetype = factor(south) ,
        facets = ~female)
```

Figure 28: `facets = ~female`




For now, let's ignore the things that could be fixed in figures 27 and 28 (axis text, legend labels, etc.). Instead, let's look at how faceting can help us illustrate patterns in very large sets of data. In our ANES data, we have data from 1948 up through 2008. Beginning in 1972, we asked respondents for both their ideological leaning and partisanship. Looking just at those folks we could classify as Democrats and Republicans, we could check whether the parties are becoming more moderate or more polarized in the 35 years of data we have. Using the `revalue()` function from the `plyr()` package, we can recode the 7 point party identification variable into 0 or 1.

```
POST72$twoparty <- revalue(factor(POST72$pid7),
                  c("0"="1", "1"="1",
                  "2"="1", "3" = NA,
                  "4"="0", "5"="0", "6"="0"))

 qplot(data = POST72[complete.cases(POST72) ,] ,
      x = ideo7 ,  facets =  twoparty ~ year,
      fill = factor(twoparty), geom= "density",
      stat = "density", position = "identity" )
```

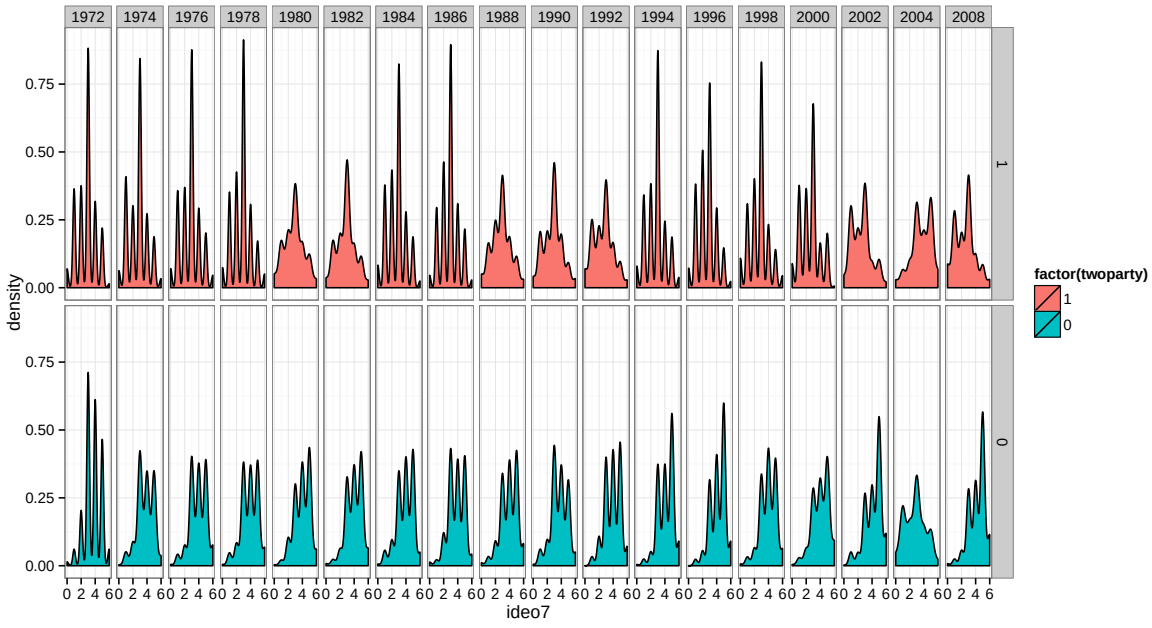



Figure 29: `facets = twoparty ~ year`

Notice here we can facet by two variables using a formula (rows ~ columns); the two different figures are shown above in figure 29 and below in figure 30; both of which could be made more aesthetically pleasing.

```
qplot(data = POST72[complete.cases(POST72) ,] ,
          x = ideo7 ,  facets =    year ~ twoparty,
          fill = factor(twoparty), geom= "density",
          stat = "density", position = "identity" )
```

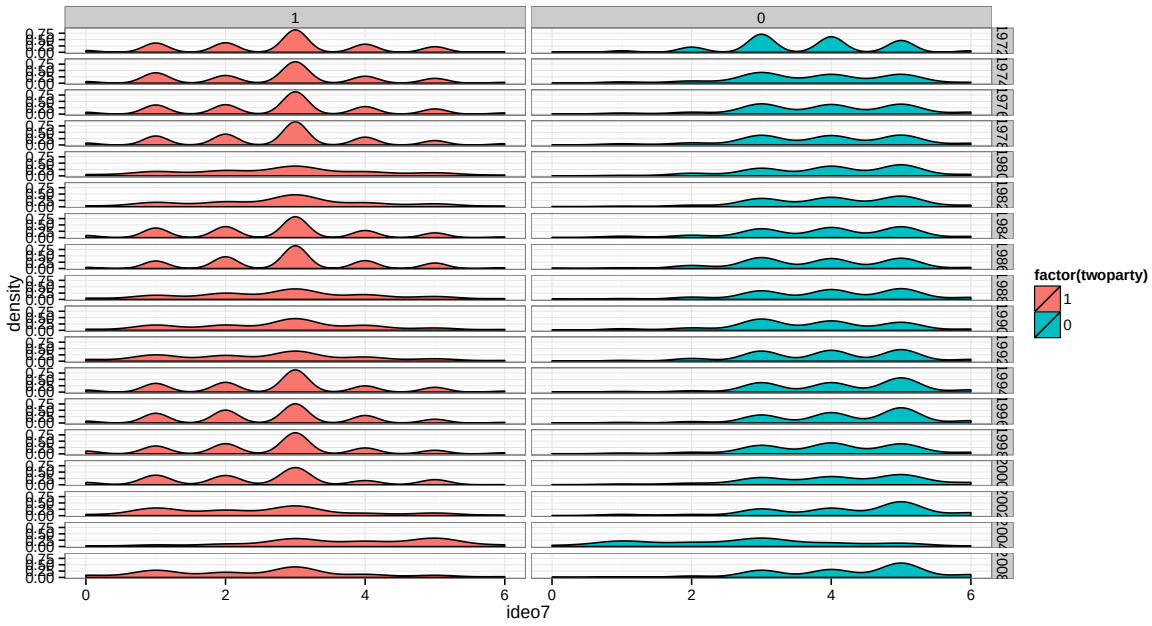
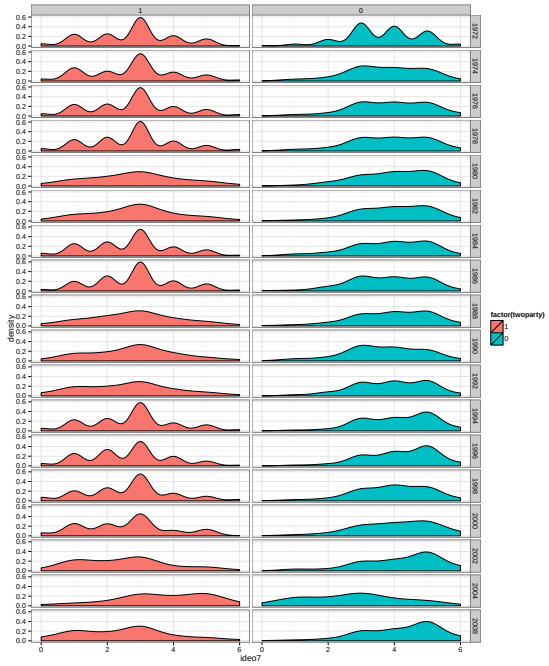



Figure 30: `facets = year ~ twoparty`

```
#NOTE: I've only added an option called "adjust"
qplot(data = POST72[complete.cases(POST72) ,] ,
       x = ideo7 ,  facets =    year ~ twoparty,
       fill = factor(twoparty), geom= "density",
      stat = "density", position = "identity",
      adjust=1.5 )
```

Changing this option allows us to see a smoother transition from year to year; though the axes labels are still quite hard to read.

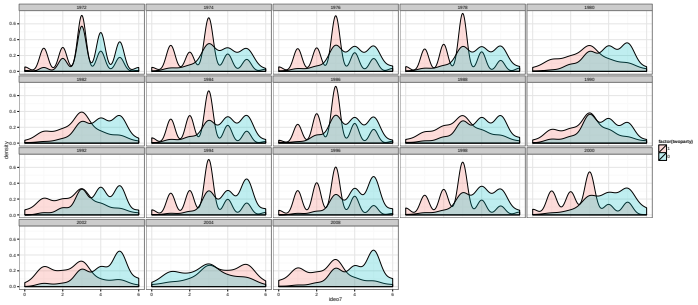
Figure 31: `adjust = 1.5`




```
#NOTE: Alpha and Adjust have changed:
qplot(data = POST72[complete.cases(POST72) ,] ,
       x = ideo7 ,  facets =    ~year ,
       fill = factor(twoparty),
       geom= "density", alpha = I(0.25),
       stat = "density", position = "identity",
       adjust=1.25 )
```

Changing the alpha level and just faceting by year, figure 32 is much easier to read, I think. Recall: this is the same data as above. I present it larger in figure 33, below.

Figure 32: `alpha = I(0.25)`



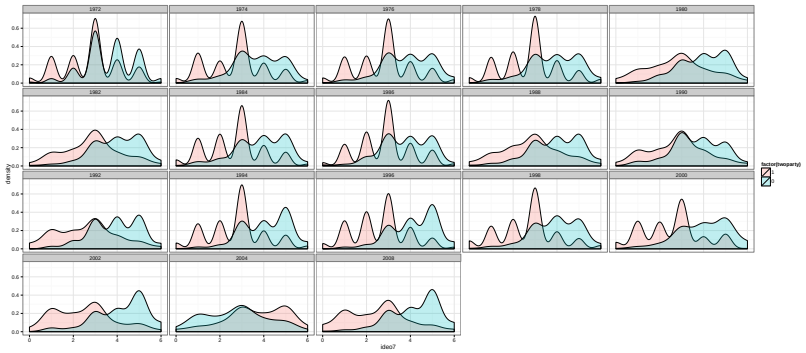




Figure 33: `facets = ~ year`

### 4.2 ggsave()

Given the trouble you might have reading important parts of the plots above, now is a good time to talk about how to export plots from R. The `ggsave()` command is a nice way to automatically save the last plot you produced to the working directory, using the following syntax. If you don't know where your working directory is, use getwd(). To set the working directory, use setwd().

```
ggsave(filename = default_name(plot), plot = last_plot(),
  device = default_device(filename), path = NULL,
  scale = 1, width = par("din")[1],
  height = par("din")[2], units = c("in", "cm", "mm"),
  dpi = 300, ...)
```

Here's an example, which saves a 6x4 inch PDF version of the plot (`aPlot`) in figure 33 in my working directory, and names it "myplot."

```
aPlot <- qplot(data = POST72[complete.cases(POST72) ,] ,
    x = ideo7 ,  facets =    ~year ,
    fill = factor(twoparty),
    geom= "density", alpha = I(0.25),
    stat = "density", position = "identity",
    adjust=1.25 )

ggsave("myplot.pdf", aPlot,
width=6, height=4, units="in")
```

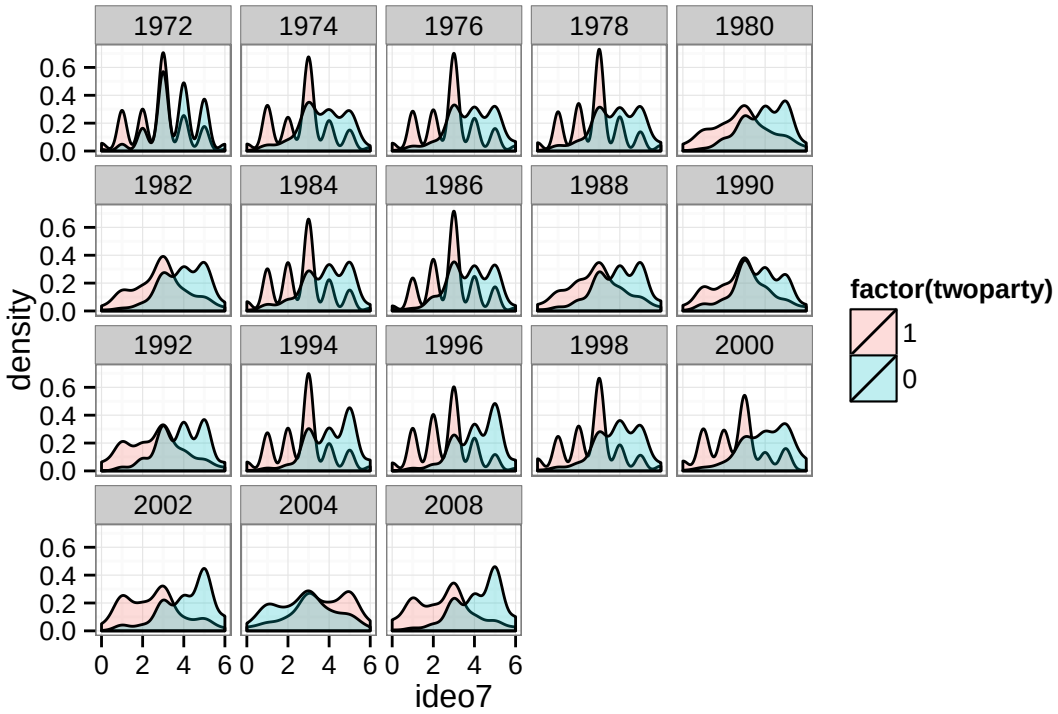



Figure 34: `facets = ~ year`

Another way to do this is to, for example, adjust the plot to the dimensions you want inside R or RStudio and then use the `scale()` option in `ggsave()` to make it larger or smaller.

```
ggsave("myplot1.pdf", aPlot, scale = 1)
```

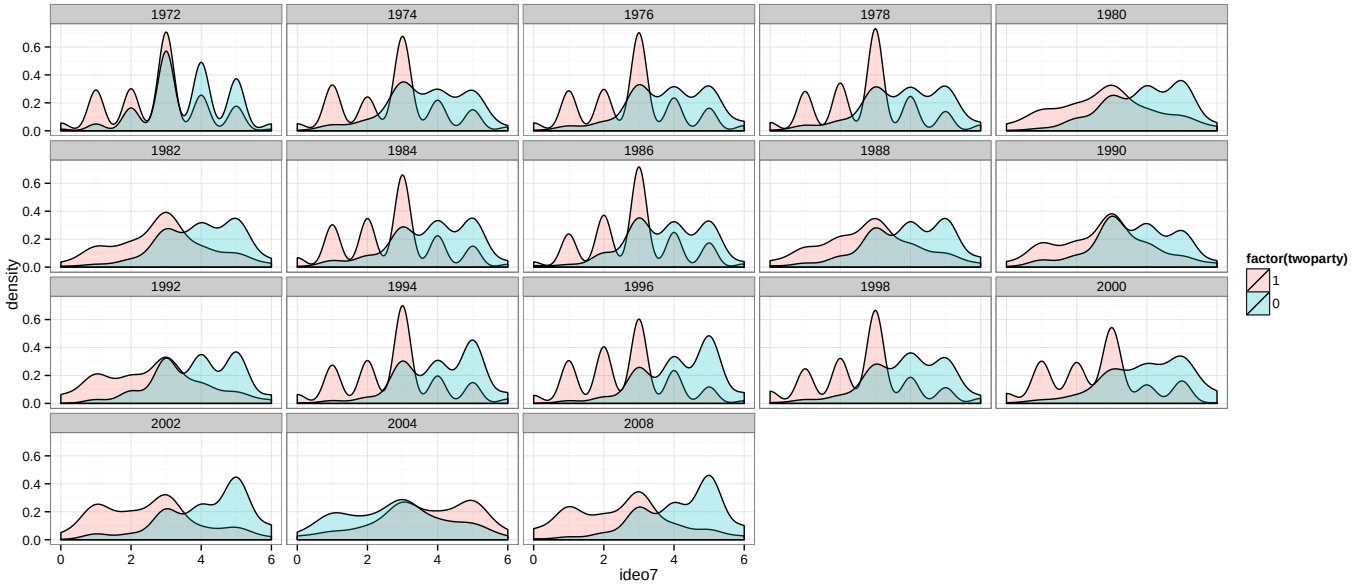



Figure 35: `ggsave("myplot1.pdf", aPlot, scale = 1)`

```
ggsave("myplot12.pdf", aPlot, scale = 0.50)
```

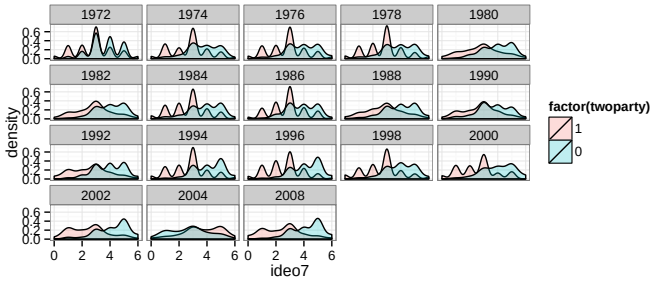



Figure 36: `ggsave("myplot12.pdf", aPlot, scale = 0.50)`

```
ggsave("myplot122.pdf", aPlot, scale = 0.250)
```

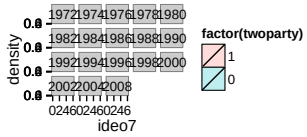



Figure 37: `ggsave("myplot12.pdf", aPlot, scale = 0.250)`

### Balancing the Scales

Because you have **two scaling commands** at your disposal; one in `ggplot2` and the other in LaTeX, your figures should always be legible and neat; all crappy graphics in this work only serve to illustrate how to do certain things, not polish all figures. Below, I show you some various options to play with the different scaling parameters using the same graphic as above, which is saved as a 8.06" x 4.11" image in R.

```
  ggsave("myplotA.pdf", aPlot, scale = 1) #ggplot2 scale option
\begin{figure}
\includegraphics[scale=1]{myplotA.pdf} % TeX scale option
\end{figure}
```

Figure 38: SCALES: `ggplot2`= 1, TEX= 1

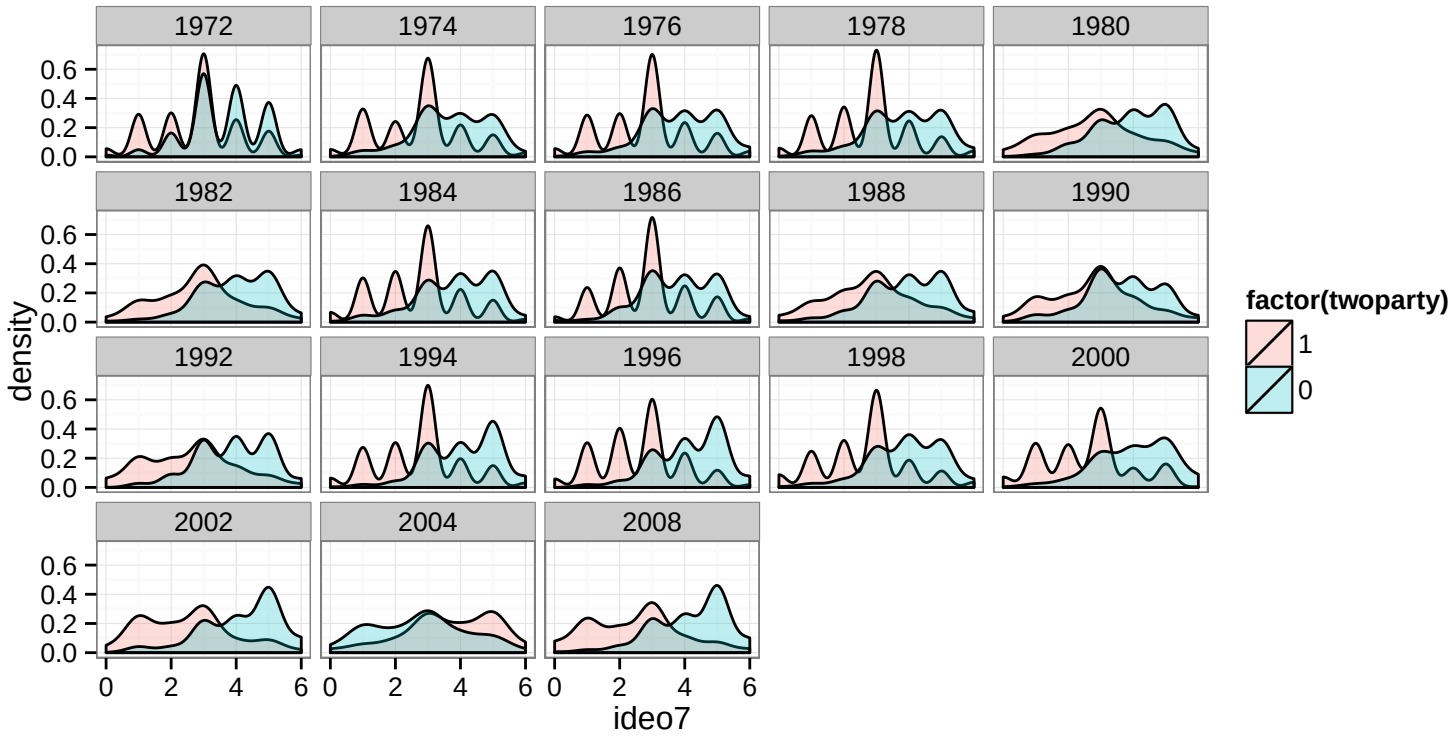



Figure 39: SCALES: `ggplot2`= 0.5, TEX= 1

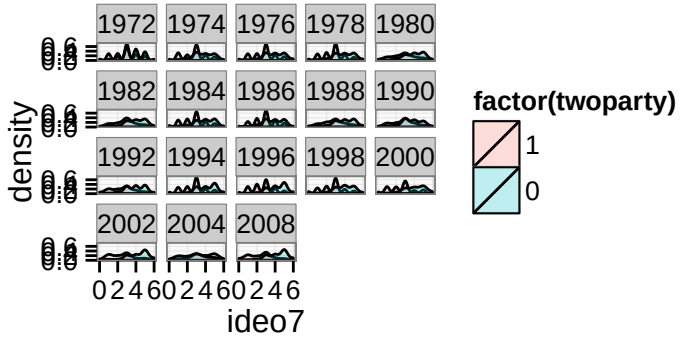


Figure 40: SCALES: `ggplot2`= 0.75, TEX= 0.75

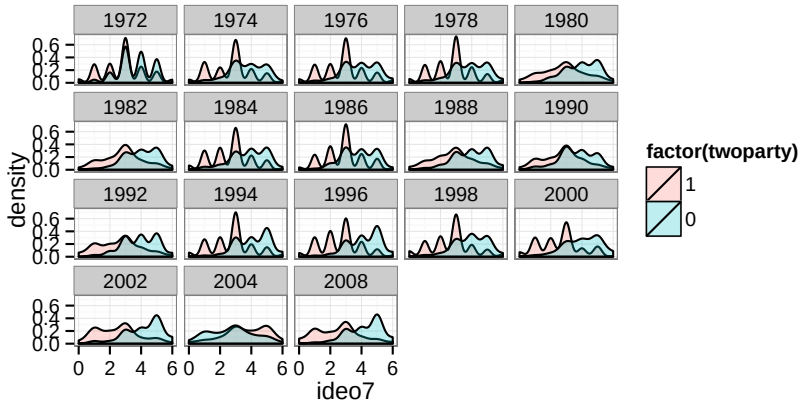



Figure 41: SCALES: `ggplot2`= 1, TEX= 0.5

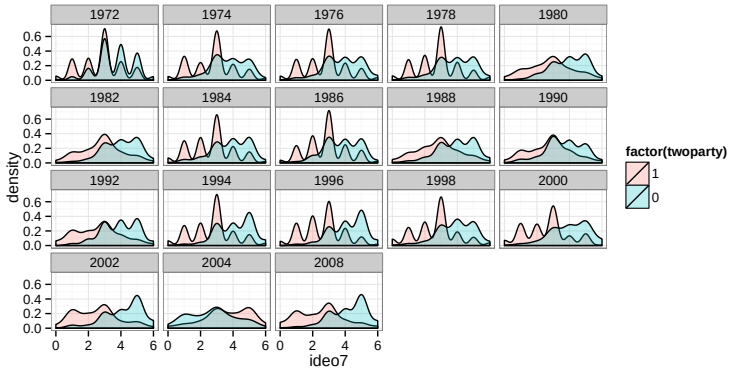



Figure 42: SCALES: `ggplot2`= 1, TEX= 0.75

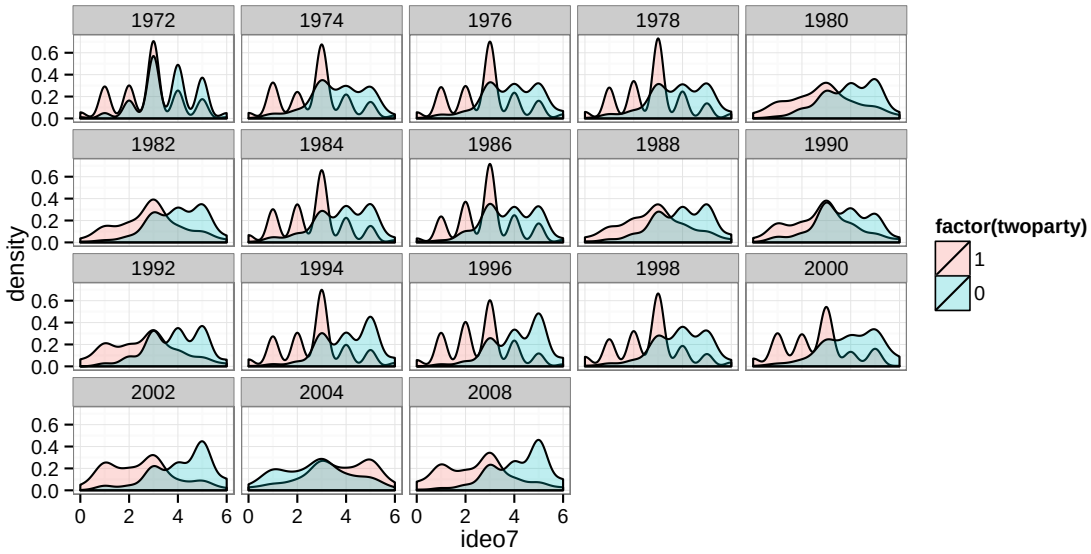



Figure 39 was terrible; figures 43 and 44 (next page) look pretty good, though!

Figure 43: SCALES: `ggplot2`= 1.25, TEX= 0.75

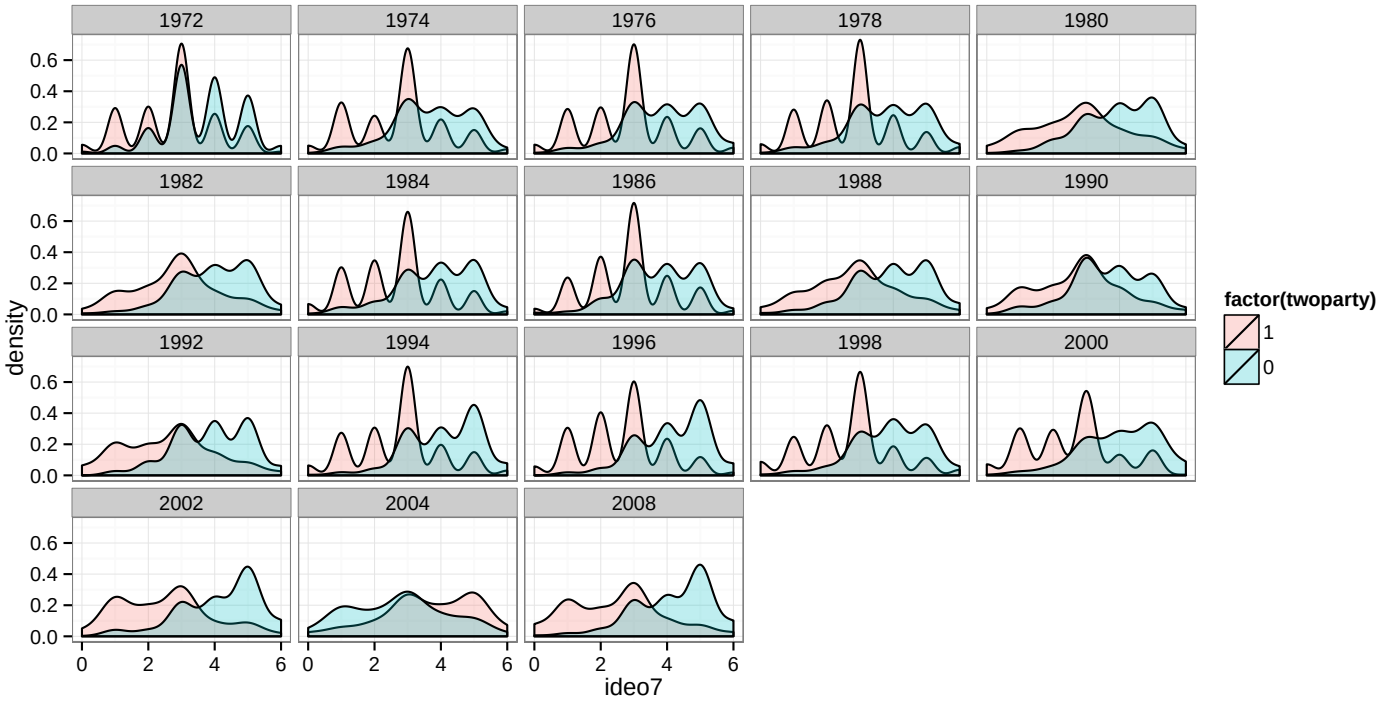
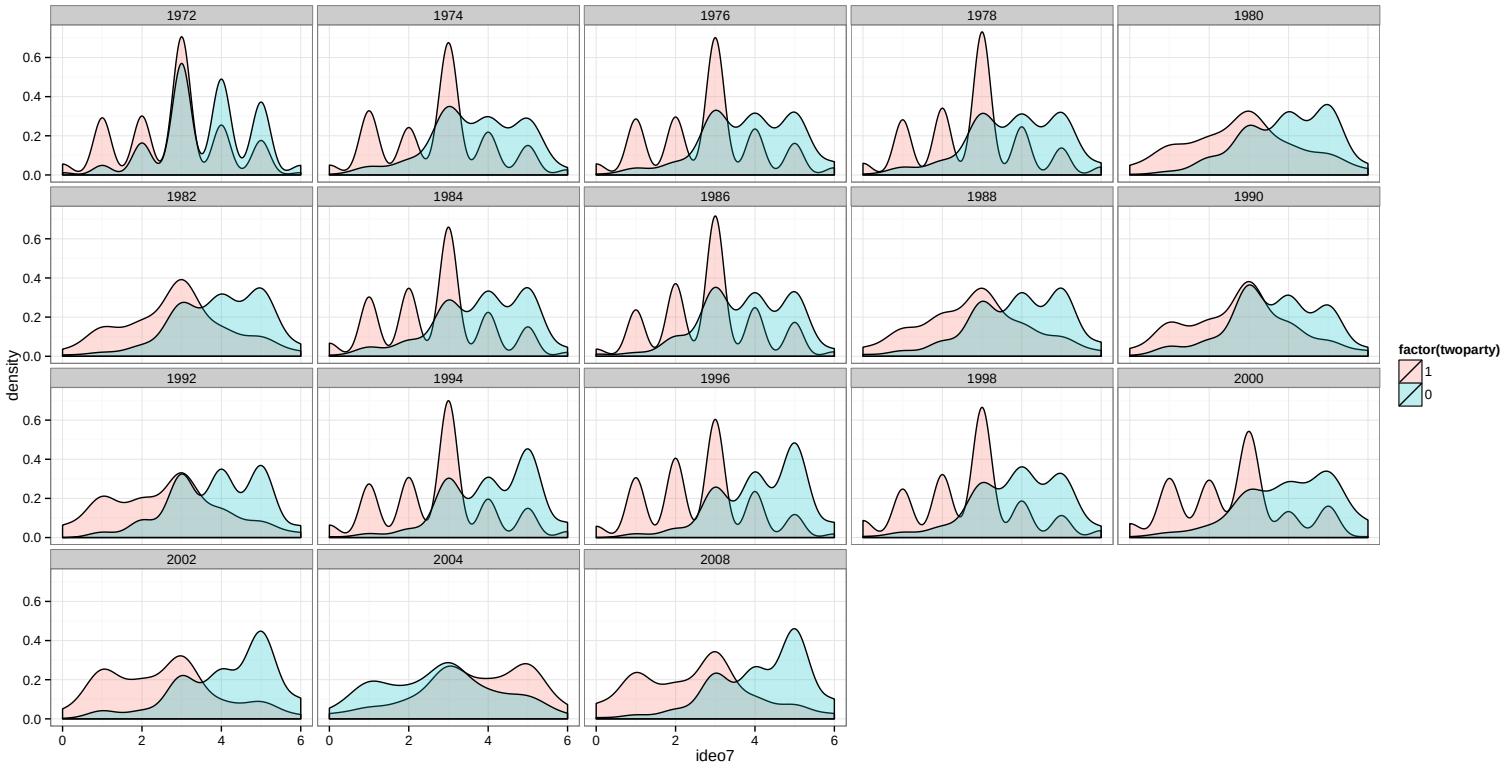



Figure 44: SCALES: `ggplot2`= 2 , TEX= 0.5



### 4.3 grid.arrange()

Using the `grid.arrange()` function from the `gridExtra` package along with `ggplot2` is an easy way to combine multiple graphical objects into a single graphic. In `ggplot2` you can simply assign a plot to an object name, as shown below, and then call the grid.arrange() command to print each of the plots in a grid whose dimensions you specify.

```
library(ggplot2)
library(gridExtra)
Var.1 <- 0:10
Var.2 <- (Var.1^2 - 12*Var.1 + 6)

PlotA <- ggplot(data = my.data) + geom_point(aes(x = Var.1, y = Var.2))

PlotB <- ggplot(data = my.data) + geom_line(aes(x = Var.1, y = Var.2))

Var.3 <- sample(c(1:100), 300, replace=T)
bar.data <- data.frame(Var.3)

PlotC <- ggplot(data = bar.data) + geom_bar(aes(x = Var.3),
                                    stat = "bin", binwidth=1)
LOWS <- 0:4
Var.Y <- 1:5
HIGHS <- 2:6
Var.X <- c(10, 20, 30, 40, 50)
pointrange.data <- data.frame(LOWS, HIGHS,
                              Var.Y, Var.X)

PlotD <-  ggplot(data = pointrange.data
) + geom_pointrange(aes(x = Var.X, y = Var.Y,
             ymin = LOWS, ymax = HIGHS ) )

grid.arrange(PlotA, PlotB, PlotC, PlotD, ncol=2)
```

Figure 45 shows the resulting output:

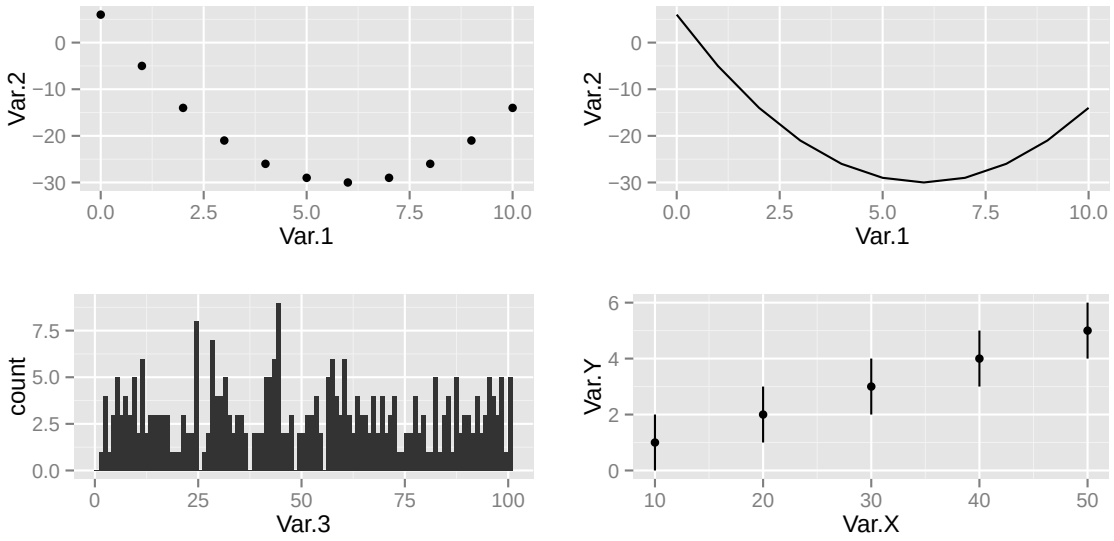



Figure 45: using `grid.arrange()`

# 5 Color in ggplot2:

So, where does ggplot2get its colors? If you've ever asked ggplot2 to color on the basis of a factor, you might have been surprised by the default color choices. The fact is, ggplot colors factors on the basis of finding evenly spaced colors around the HCL space.


Figure 46: Colors in ggplot2 (script is available here)

## 5.1 Custom Colors in ggplot2:

**scale_color_manual:**

So far we've covered Melting and Casting data using the reshape() package and today were going to look at different ways of coloring and selecting palettes for plots. For these plots, were going to use the built in diamonds dataset that comes packaged in ggplot2. Below, I go through loading the data and selecting four different palettes to color a plot from the diamonds data set using the scale_color_manual() option in ggplot.

```
library(ggplot2)
library(gridExtra)
data(diamonds)
diamonds <-     diamonds[diamonds$color < "J",]
#Script for Figure 19
#Random Colors
my.colors <- sample(colors(), 7)
RANDOM <- ggplot(data = diamonds) + geom_point(aes( x = carat, y = price, color=factor(color))
) + facet_wrap(~color) + scale_color_manual(values = my.colors)   + labs(title = "+ scale_color_manual(values=sample(colors(), 7)\n" )

custom.colors.1 <- c("red", "orange" , "yellow", "green", "blue", "blueviolet", "violet")
C1 <- ggplot(data = diamonds) + geom_point(aes( x = carat, y = price, color=factor(color))
) + facet_wrap(~color) + scale_color_manual(values = custom.colors.1) + labs(title = "+ scale_color_manual(values=custom.colors.1)\n" )

custom.colors.2 <- c("skyblue1", "skyblue2" , "skyblue3", "steelblue", "springgreen2", "springgreen3", "springgreen4")
C2 <- ggplot(data = diamonds) + geom_point(aes( x = carat, y = price, color=factor(color))
) + facet_wrap(~color) + scale_color_manual(values = custom.colors.2
) + labs(title = " + scale_color_manual(values = custom.colors.2) \n" )

#Rainbow Colors
ROYGBIV <- rainbow(6)
RAINBOW <- ggplot(data = diamonds) + geom_point(aes( x = carat, y = price, color=factor(color))
) + facet_wrap(~color) + scale_color_manual(values = ROYGBIV)  + labs(title = " + scale_color_manual(values = rainbow(6)) \n" )

grid.arrange(RANDOM, C1, C2, RAINBOW, ncol=2)
```

Figure 47 shows the resulting output:

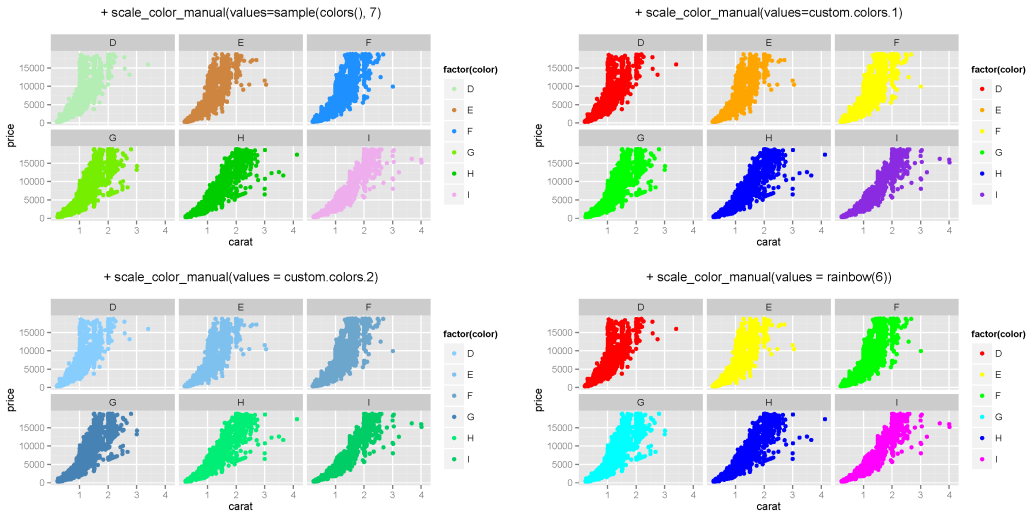



Figure 47: using `grid.arrange()` along with custom colors

### scale_colour_gradientn

If you have a continuous dependent variable, youll want to use a gradient option that gently transitions between two specified colors. Some examples are below:

```
data(diamonds)
diamonds <-     diamonds[diamonds$color < "J",]

G5 <- ggplot(data = diamonds ) + geom_point(aes( x = carat,
 y = price, color=price) ) + facet_wrap(~color)
G6 <- G5 + scale_colour_gradientn(colours=rainbow(2))
G7 <- G5 + scale_colour_gradientn(colours=c("red", "blue"))
G8 <- G5 + scale_colour_gradientn(colours=c("white", "dodgerblue"))
grid.arrange(G5, G6, G7, G8, ncol=2)
```

Figure 48 shows the resulting output:

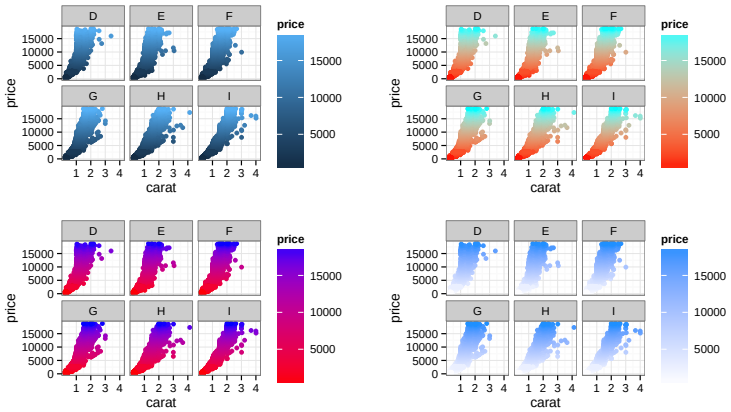



Figure 48: using `grid.arrange()` along with custom colors

### scale_colour_brewer

If you want to choose colors for factors or discrete variables, you might want to use a Brewer color scale (http://colorbrewer2.org/):

```
#Script for Figure 21
library(ggplot2)
library(gridExtra)
data(diamonds)
diamonds <-  diamonds[diamonds$color < "J",]
B1 <- ggplot(data = diamonds ) + geom_point(aes( x = carat, y = price, color=color)
) + facet_wrap(~color) + scale_colour_brewer(palette="Set1")   + labs(title = "Palette=''Set1''\n" )
B2 <- ggplot(data = diamonds ) + geom_point(aes( x = carat, y = price, color=color)
) + facet_wrap(~color) + scale_colour_brewer(palette="Set2") + labs(title = "Palette=''Set2''\n" )
B3 <- ggplot(data = diamonds ) + geom_point(aes( x = carat, y = price, color=color)
) + facet_wrap(~color) + scale_colour_brewer(palette="Blues")   + labs(title = "Palette=''Blues''\n" )
B4 <- ggplot(data = diamonds ) + geom_point(aes( x = carat, y = price, color=color)
) + facet_wrap(~color) + scale_colour_brewer(palette="Reds")   + labs(title = "Palette=''Reds''\n" )
grid.arrange(B1, B2, B3, B4, ncol=2)
```

Figure 49 shows the resulting output:

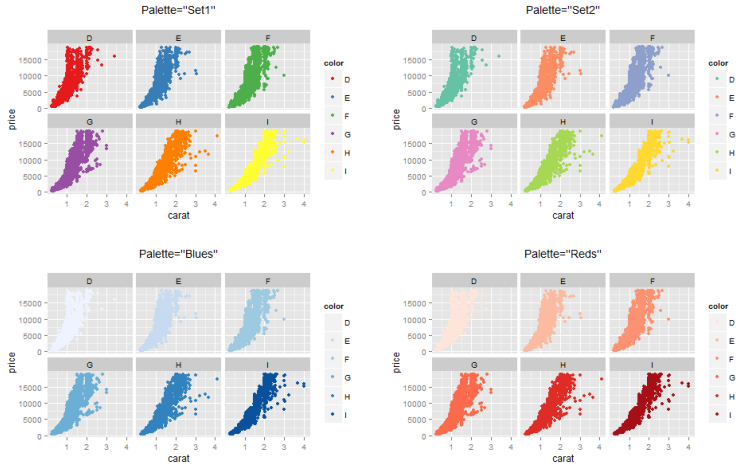



Figure 49: using `grid.arrange()` along with Brewer colors

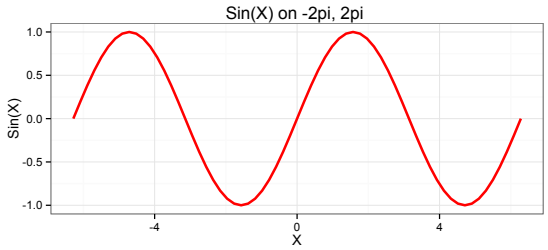
# 6 Manipulating Text in ggplot2:

## 6.1 Using Mathematical Expressions in Labels

This section relies on using the function `bquote()`, which from its help file is "an analogue of the LISP back-quote macro. `bquote` quotes its argument except that terms wrapped in .() are evaluated in the specified where environment." Since I'm sure that's less than perfectly clear, I'll attempt to illustrate it with an example.

```
X <- seq(-2*pi, 2*pi, by = pi/16)
Sin.X <- sin(X)
P1 <- qplot(X, Sin.X, geom = "line", ylab = "Sin(X)",
            main = "Sin(X) on -2pi, 2pi ",
            colour = I("red"), size = I(1))
P1
```

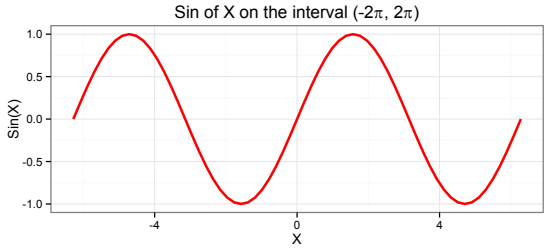
Figure 50: Sin(X) on $-2\pi, 2\pi$




Using the `bquote()` command, we can improve upon figure 50 by changing the "pi" to $\pi$ in the plot's title. Here, we use quotes to contain any regular text and asterisks (*) to delimit/separate the elements; `bquote()` knows to make the necessary changes to print common (mathematical) symbols. This improvement results in figure 51.

```
title.text <- bquote("Sin of X on the interval (-2" *
                pi  * ", 2"  * pi* ")"      )

P2 <- qplot(X, Sin.X, geom = "line", ylab = "Sin(X)",
            main = title.text ,
            colour = I("red"), size = I(1))
            P2
```

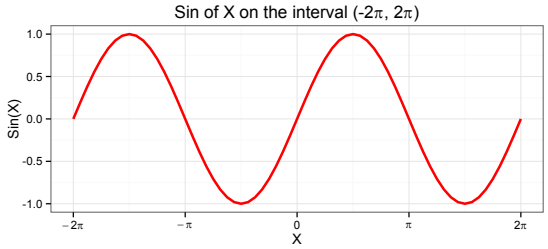
Figure 51: with an improved title




If, however, we have only mathematical symbols that we want to print, we can concatenate those symbols in a regular vector and use `expression()` to translate them accordingly; this is shown in figure 52.

```
P3 <- P2 + scale_x_continuous(breaks = seq(-2*pi,
 2*pi, by=pi),  labels = c(expression(-2*pi) ,
 expression(-pi) , 0,  expression(pi),
 expression(2*pi ) ) )
P3
```

Figure 52:

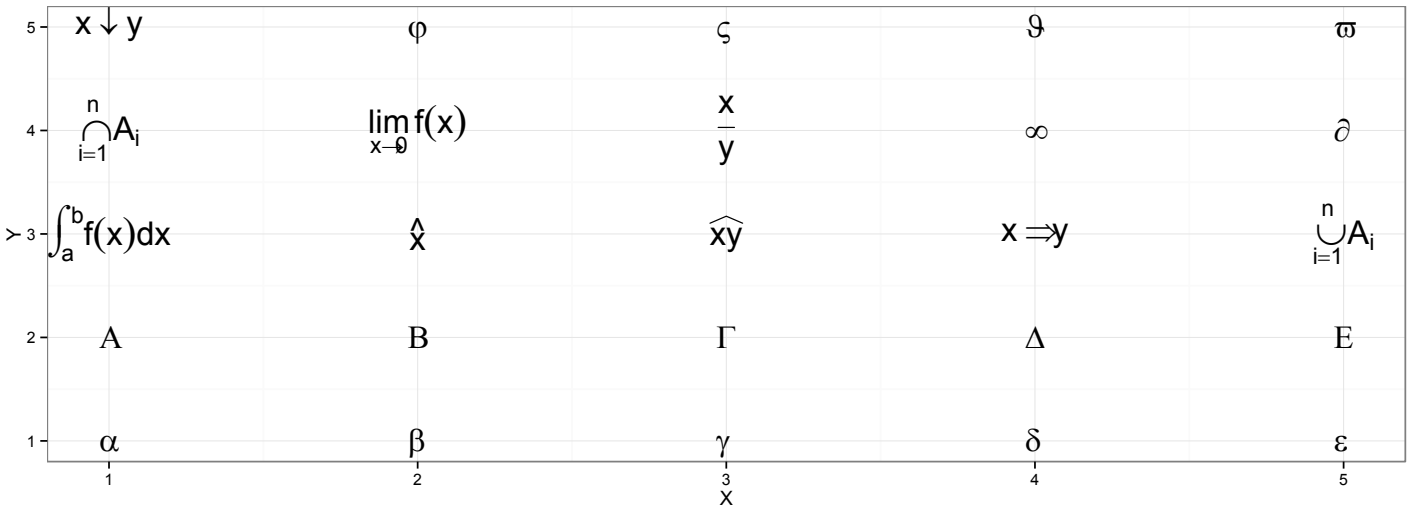


## 6.2 plotmath in ggplot2

Below are just a few of the symbols that we can print in R graphics. For a more comprehensive list, run `?plotmath` or visit this website: http://astrostatistics.psu.edu/su07/R/html/grDevices/html/plotmath.html

```
?plotmath
X <- rep(1:5, 5)
Y <- rep(1:5, each=5)
cbind(X, Y)

Greek <- c(
  expression(alpha) ,   expression(beta) ,   expression(gamma) ,   expression(delta) ,
  expression(epsilon) ,   expression(Alpha) ,   expression(Beta) ,   expression(Gamma) ,
  expression(Delta) ,   expression(Epsilon) ,   expression(integral(f(x)*dx, a, b)) ,
  expression( hat(x)) ,   expression(widehat(xy)) ,   expression( x %=>% y ) ,
  expression(union(A[i], i==1, n)) ,   expression(intersect(A[i], i==1, n)) ,
  expression(lim(f(x), x %->% 0)) ,   expression(frac(x, y)) ,
    expression(infinity ) ,   expression(partialdiff) ,
  expression(x %down% y  ) ,   expression(phi1) ,   expression(sigma1) ,
  expression(theta1) ,   expression(omega1) )

Plot <- ggplot()
Plot + geom_text(aes(x = X, y = Y, geom = "text",
 label  =  paste(Greek)), parse=TRUE, size = I(7))
```

Figure 53: Go Nuts!



## 6.3 Custom Axis Breaks using + scale_y_continuous()

These examples are available online here, but for each of the text manipulations we will do we will begin with a base plot object referred to as Plot.1. Plot.1 uses some data that were invented in order to illustrate manipulating text in ggplot2. The real data is available as the ToothGrowth dataset in R.

```
text.plots <- data.frame(
  SUPP = c (rep(c( "Control", "Vitamin C", "Orange Juice" ), each = 1500))  ,
  DOSE = rep(rep(c( "I", "II", "III" ), each=500), 3),
  LENGTH = c(rnorm(500, 4, 2),
             rnorm(500, 7, 1.2),
             rnorm(500, 8, 1.4),
             rnorm(500, 7, 1),
             rnorm(500, 8, 4),
             rnorm(500, 10, 2),
             rnorm(500, 8, 2.7),
             rnorm(500, 7, 1.8),
             rnorm(500, 6, 1.9)), stringsAsFactors = FALSE )

Plot.0 <- ggplot( text.plots, aes( x = SUPP, y = LENGTH , fill = SUPP ) ) +
  geom_boxplot( alpha = 0.6, outlier.colour = c("grey40") , outlier.size=3.5
  ) + scale_fill_manual(values=c("cadetblue", "orange", "orangered3")) +
  facet_wrap(~DOSE) + theme_bw() +labs(title="Tooth Growth in Guinea Pigs \n",
           x="\n Treatment", y="Change in Length (mm) \n") +  guides(fill = guide_legend("\n Supplement")
           ) +geom_hline(y=0, lty=2)

Plot.0


Plot.1 <- Plot.0 + scale_y_continuous(breaks=seq(-4, 24, by=4))
Plot.1
```

Below, figure 54 shows our base plot, Plot.1, which we will manipulate as an object.

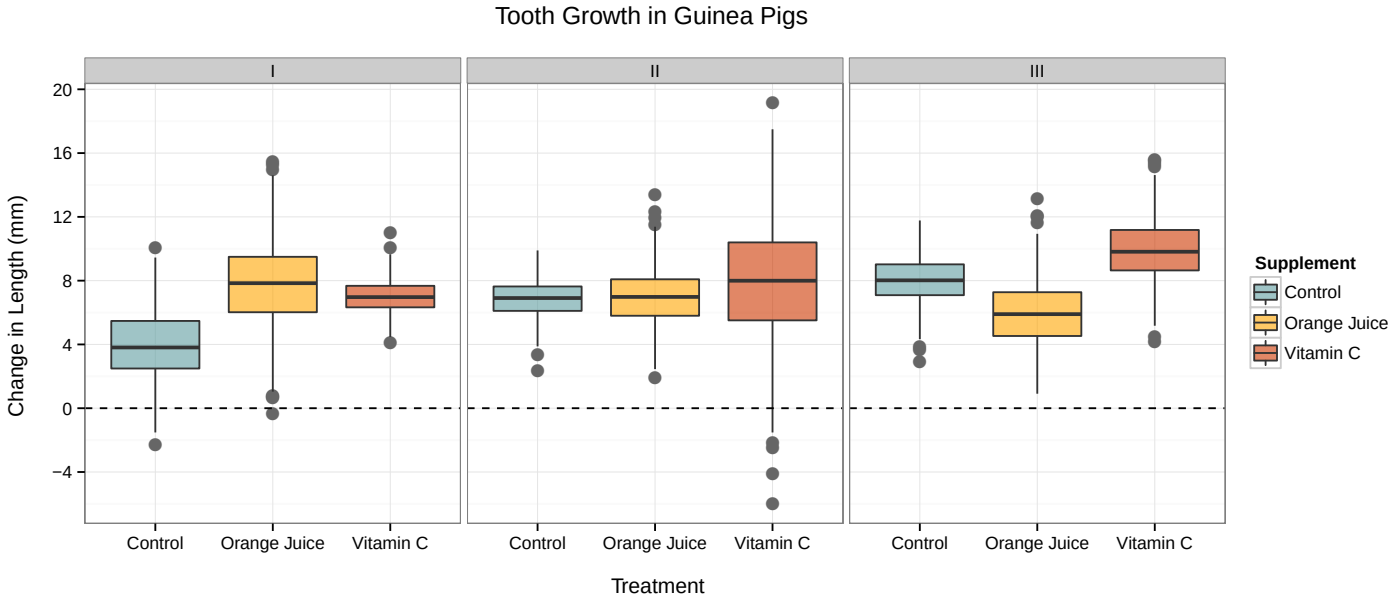


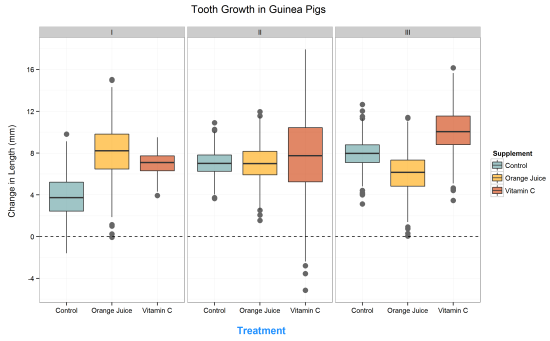

Figure 54: Plot.1

## 6.4 Textual Healing

To Adjust X Axis Label Size/Face/Color:

```
Plot.1 + theme(axis.title.x = element_text(face="bold",
               colour="dodgerblue",
            size=14) )
```

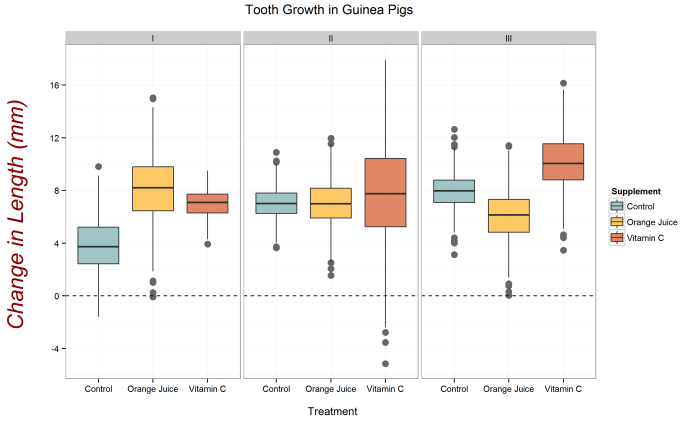
Figure 55: + theme(axis.title.x ...




To Adjust Y Axis Label Size/Face/Color:

```
Plot.1 + theme(axis.title.y =
element_text(face="italic",
               colour="darkred",
               size=24) )
```

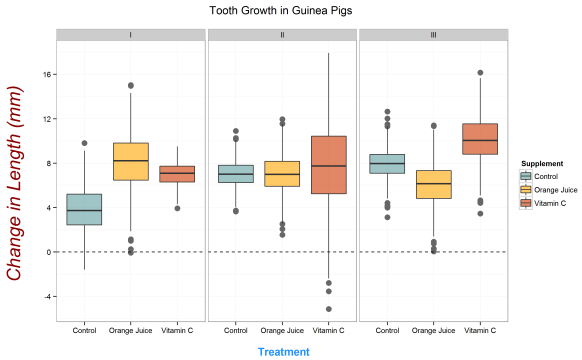
Figure 56: + theme(axis.title.y ...




When adjusting multiple parameters, you can use specify both within the "theme" command:

```
Plot.1 + theme(axis.title.x =
element_text(face="bold",
             colour="dodgerblue",
             size=14) ,
                axis.title.y =
                element_text(face="italic",
                colour="darkred",
                size=24) )
```

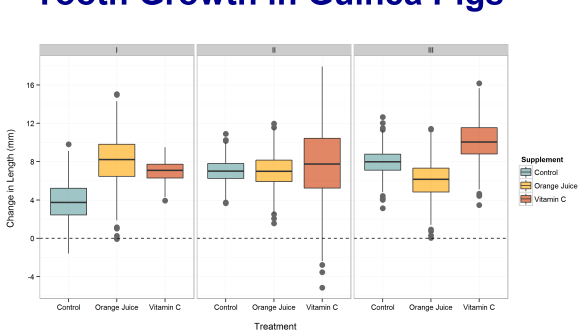
Figure 57: + multiple changes



Changing Main Title Text/Face/Color:

```
Plot.1 + theme(plot.title =
 element_text(family="sans",
                face="bold",
                colour="darkblue",
                size=44) )
```

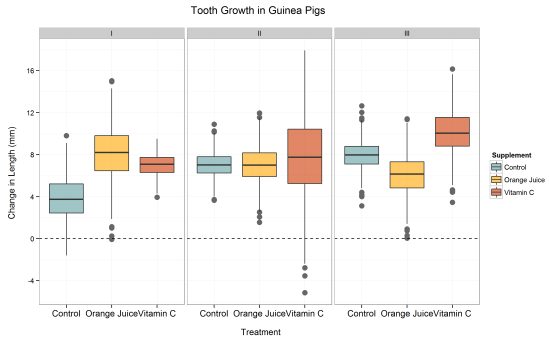
Figure 58: + theme(plot.title ...




Another option in the element_text() is angle, which is useful in certain occasions; especially when you have tight axis tick labels, like below in figure 59.

```
Plot.1  + theme(axis.text.x =
element_text(colour="black",
            size = 12,
            face = "plain"))
```

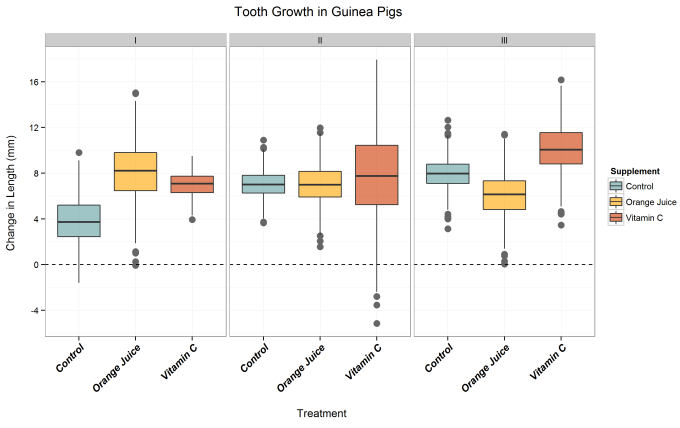
Figure 59:




By adjusting the angle at which the tick text is displayed, this problem can be solved. This is shown in figure 60; changes can be made to other labels *mutatis mutandis*[1].

```
Plot.1  + theme(axis.text.x =
element_text(colour="black",
        size = 11,
        face = "bold.italic",
        angle=45,
        vjust=1,
        hjust=1))
```

Figure 60:



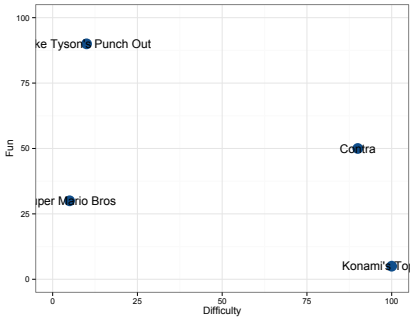

[1] "Changing those things which need to be changed;" to instead alter the y-axis text, you would change `axis.text.x` to `axis.text.y`.

## 6.5 Working with `hjust()`, `vjust()` and `angle()`

When you're working with text and things aren't lining up as beautifully as you'd like, there are some options to adjust where the text label begins/ends in relation to the point it is labeling. Figure 61 shows one such example; here I've plotted how fun I thought a Nintendo game was against how difficult it was; for the record, I never got beyond the second level on Top Gun; that game is possibly my least favorite thing ever.

```
Difficulty <- c(5, 10, 90, 100)
Fun <-  c(30, 90, 50, 5)
Video.Games <- c("Super Mario Bros",
                 "Mike Tyson's Punch Out",
                 "Contra", "Konami's Top Gun")
qplot(Difficulty, Fun, geom = "point",
 xlim=c(0,100),
         ylim=c(0,100),
        size = I(6), colour= I("dodgerblue4")
 ) +geom_text( label = Video.Games)
```

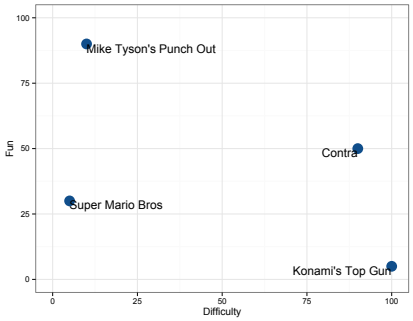
Figure 61:



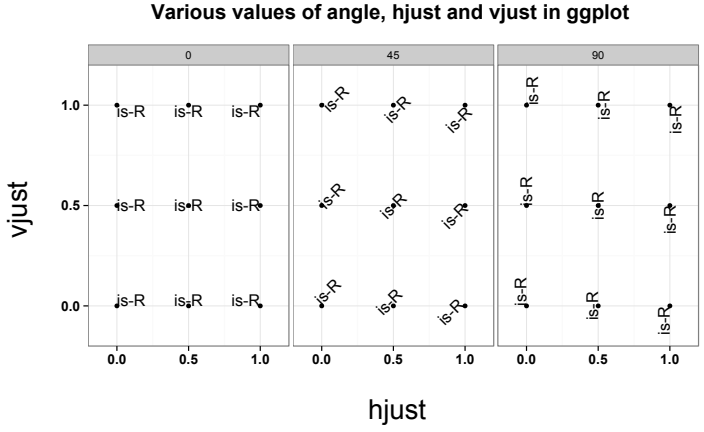

Here we can alter `hjust()`, `vjust()` and `angle()` inside the `geom_text()` command to change both the justification as well as the angle at which the text labels are printed. Of course, one solution would be to simply make the x and y axes have larger ranges, but this is supposed to be a teaching example.By making a new variable called "Konami," I can make things line up nicely, as shown in figure 62.

```
Difficulty <- c(5, 10, 90, 100)
Fun <-  c(30, 90, 50, 5)
Video.Games <- c("Super Mario Bros",
                 "Mike Tyson's Punch Out",
                 "Contra", "Konami's Top Gun")

Konami <- c(0,0,1,1) #New variable mapped to hjust
qplot(Difficulty, Fun, geom = "point", xlim=c(0,100),
      ylim=c(0,100),
      size = I(6), colour= I("dodgerblue4")
) +geom_text( label = Video.Games,
hjust=Konami, vjust=1 )
```

Figure 62:




Below is a cheat sheet showing various values for common combinations of all three options:



# 7 The Legend(s) of ggplot2

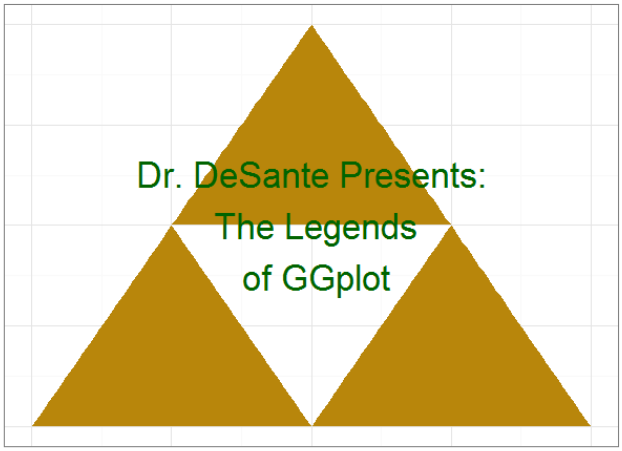


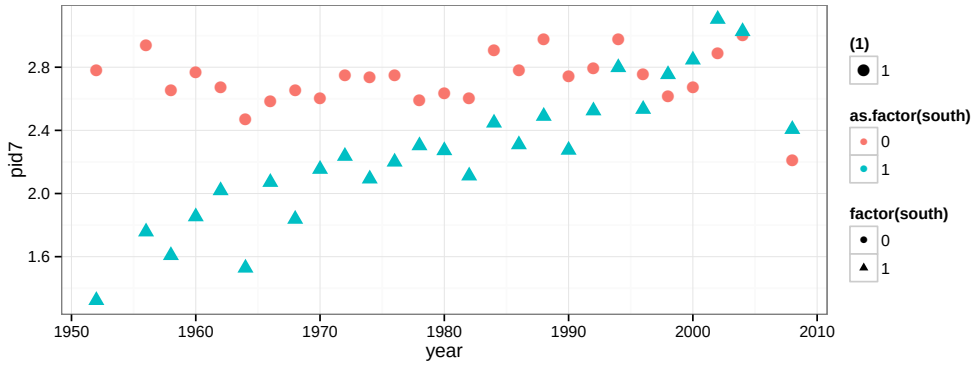

Figure 63: A Link to Your Past?

## 7.1 Troubleshooting Legends

As we discussed before, sometimes going through all the steps to make a nice graphic leaves you with extra legends or legends that don't quite look right. For example, take a look at the code below that produces figure 64.

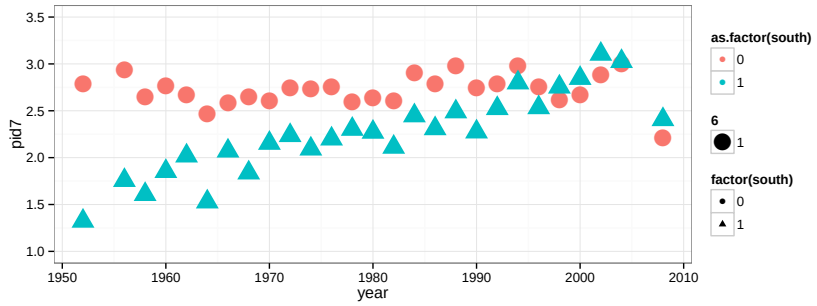
```
ANES <- read.csv("http://pages.iu.edu/~cdesante/ANES.csv")
party.and.region <- ANES[,c(1, 12, 16)]
party.and.region.year <- melt(party.and.region,
id=c("year", "south"), na.rm=TRUE)
party.region.time  <- dcast(party.and.region.year,
year+south~variable, mean, na.rm=T)

Legends.0 <- ggplot(data = party.region.time
) + geom_point(aes(x = year,
                   y = pid7,
                   size=(1),
                   colour=as.factor(south),
                   shape=factor(south)))  +theme_bw()
```

Figure 64: Legends.0




What we see is that the size variable is showing a legend with a value of "(1)" while the colors and shape which correspond to the same variable (`south`) not only are uninformative, but are redundant. Let's make some changes:

Figure 65: `Legends.1 ← Legends.0 + scale_size_area(6) + ylim(1,3.5)`




```
Legends.1 <- Legends.0  + scale_size_area(6) + ylim(1,3.5)
#Line above alters size and y-axis limits;
Legends.2 <- Legends.1 + scale_colour_manual(values=c("darkblue", "darkred"),
                            labels=c("Non-South", "South"))
#Line above alters the colors we will use as well as the labels those values to which they correspond.
Legends.3 <- Legends.2 + scale_shape_discrete(name  ="Region",
                          labels=c("Non-South", "South"))
#Line above labels the shapes we will use.
```

Figure 66: `Legends.2`

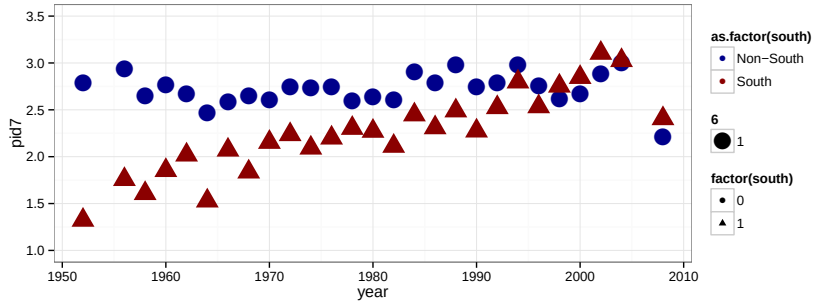
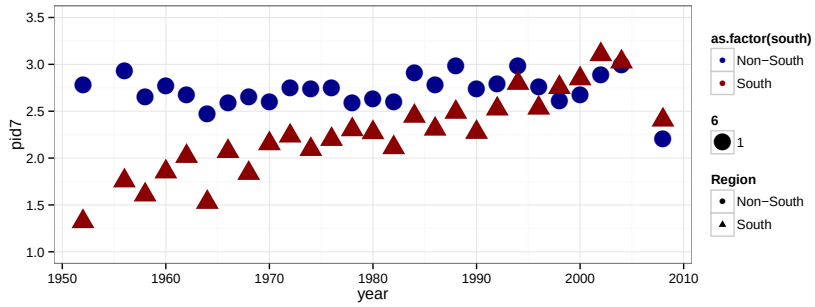


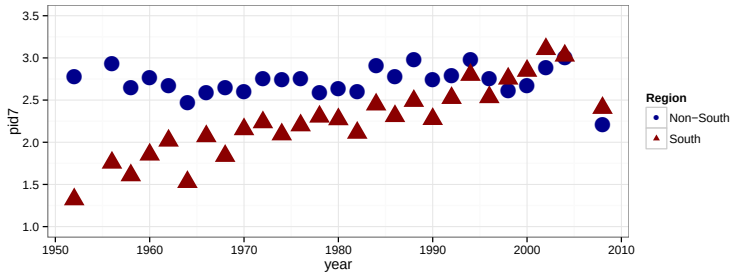
Figure 67: `Legends.3`




We still have a problem though in the resulting figure; ideally we'd have larger labels and we'd have red triangles for the south and blue circles for the non-south in our legends. We can do this as follows:

```
Legends.4 <- Legends.3 + guides(colour = "legend",
                                size = "none",
                                shape = "legend") + guides(colour = guide_legend("Region"),
                                                           shape = guide_legend("Region"))
```

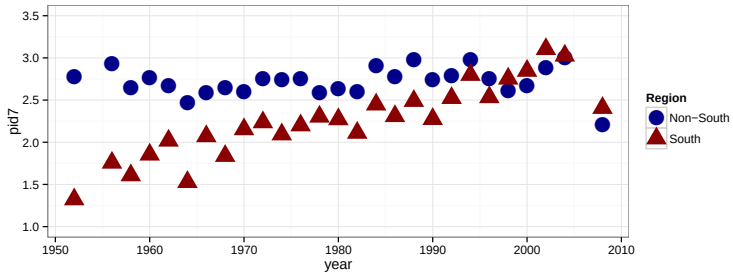
Figure 68: `Legends.4`




Our last problem is now the size of the colored shapes in the legend; we can fix this with the following code:

```
Legends.5 <- Legends.4 +  guides(colour = guide_legend("Region",
                          override.aes = list(size = 6)))
```

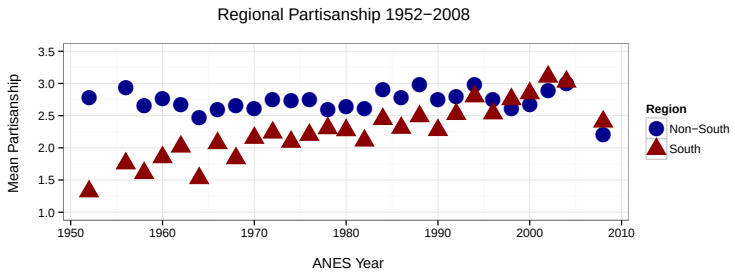
Figure 69: `Legends.5`




Once we add some axis and plot labels, we're good to go (the `\n` allows you to add a return).

```
Legends.6 <- Legends.5 + labs( list( title = "Regional Partisanship 1952-2008 \n",
                       x = "\n ANES Year",   y = "Mean Partisanship \n"))
```

Figure 70: `Legends.5`

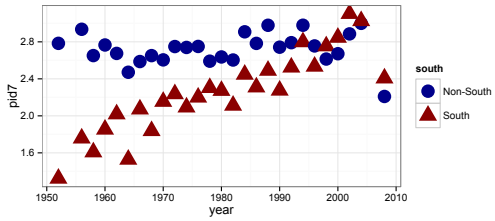



## 7.2 `ggplot()` vs. `qplot()`

Readers may have observed that I have switched between two different commands, `ggplot()` and `qplot()`, in the above examples. Honestly, I'm still not completely sure of the differences between the commands. What I know is that `qplot()` allows us to **q**uickly plot something, and `ggplot()` allows us to specify more options up front before printing the graphic. Sometimes, this is useful; it turns out we could have bypassed all of the legend troubleshooting we did above had we simply factored the "south" variable and then specified the aesthetics we wanted. Here, `ggplot()` knows that when it produces a legend it should map both colour and shape to a single legend; since we do not want a legend for size, we can move that option to outside the `aes()` and (with the addition of the custom colours) produce figure 71.

```
ANES <- read.csv("http://pages.iu.edu/~cdesante/ANES.csv")
party.and.region <- ANES[,c(1, 12, 16)]
party.and.region.year <- melt(party.and.region,
id=c("year", "south"), na.rm=TRUE)
party.region.time  <- dcast(party.and.region.year,
year+south~variable, mean, na.rm=T)
PLOT <- ggplot(data = party.region.time
) + geom_point(aes(x = year,
                   y = pid7 ,
                   colour= south ,
                   shape=  south) ,
                    size=I(6)
    )   + scale_colour_manual(values=c("darkblue", "darkred") )
print(PLOT)
```

Figure 71: Legends with `ggplot()`



# 8 Mapping Spatial Data

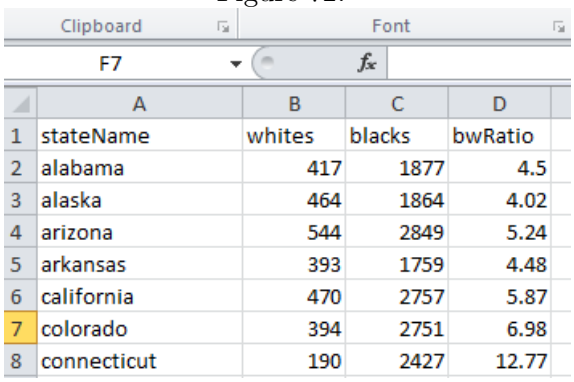
Sometimes we want to map data that are spatial in nature; county-level turnout, state-level poverty rates, etc. Below are some examples that we can use to make maps.

## 8.1 Basic State-Level Map

Let's assume you have data in a CSV file that may look like the one shown in figure 72. Notice the lower case state names; they will make merging the data much easier. The variable of interest we're going to plot is the relative incarceration rates by race (whites and blacks) across each of the fifty states (we'll remove DC once we load the data). Using the `map_data("state")` command, we can load a `data.frame` called "all_states", shown below in figure 73. In this example, my data is called "Prison" and is a CSV file of incarceration rates and poverty rates by race by state.

```
library(ggplot2)
library(maps)

Prison <- read.csv(
"http://mypage.iu.edu/~cdesante/prisons14.csv"
)
head(Prison)

all_states <- map_data("state")
all_states
head(all_states)
Prison$region <- Prison$stateName
Total <- merge(all_states, Prison, by="region")
head(Total)
Total <- Total[Total$region!="district of columbia",]
```

Figure 72:



| | A | B | C | D |
|---|---|---|---|---|
| 1 | stateName | whites | blacks | bwRatio |
| 2 | alabama | 417 | 1877 | 4.5 |
| 3 | alaska | 464 | 1864 | 4.02 |
| 4 | arizona | 544 | 2849 | 5.24 |
| 5 | arkansas | 393 | 1759 | 4.48 |
| 6 | california | 470 | 2757 | 5.87 |
| 7 | colorado | 394 | 2751 | 6.98 |
| 8 | connecticut | 190 | 2427 | 12.77 |

Figure 73: Built in State Data

```
> head(all_states)
       long      lat group order  region subregion
1 -87.46201 30.38968     1     1 alabama      <NA>
2 -87.48493 30.37249     1     2 alabama      <NA>
3 -87.52503 30.37249     1     3 alabama      <NA>
4 -87.53076 30.33239     1     4 alabama      <NA>
5 -87.57087 30.32665     1     5 alabama      <NA>
6 -87.58806 30.32665     1     6 alabama      <NA>
>
```

After merging all the data, we have a new object called "Total" which contains the geographic data as well as our variables of interest from Prison:

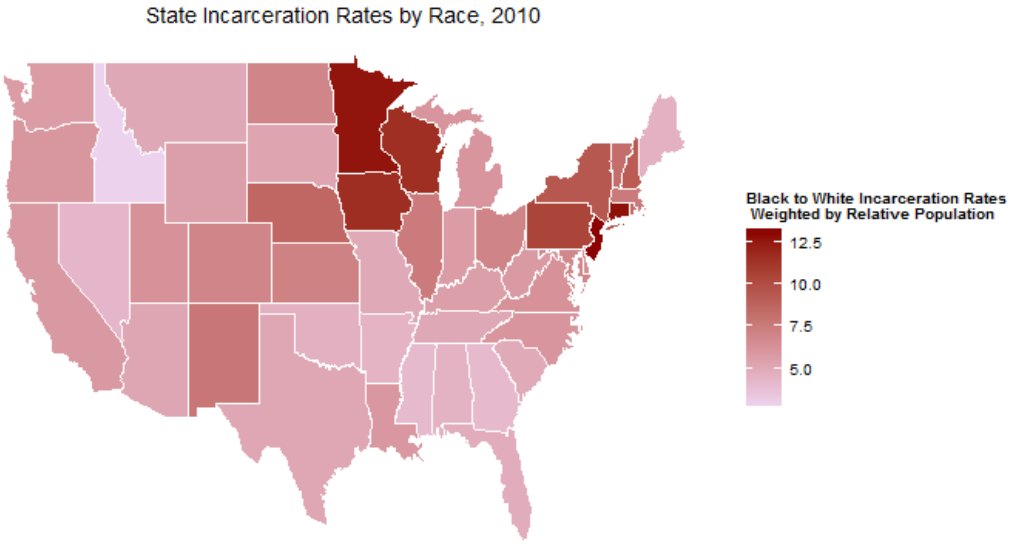
Figure 74: Data after Merge

```
> head(Total)
   region      long      lat group order subregion stateName whites blacks bwRatio wPoverty bPoverty bwPoverty bwPoverty14 prisbw14
1 alabama -87.46201 30.38968     1     1      <NA>   alabama    417   1877     4.5     12.4     30.6  2.467742         2.67 3.535055
2 alabama -87.48493 30.37249     1     2      <NA>   alabama    417   1877     4.5     12.4     30.6  2.467742         2.67 3.535055
3 alabama -87.52503 30.37249     1     3      <NA>   alabama    417   1877     4.5     12.4     30.6  2.467742         2.67 3.535055
4 alabama -87.53076 30.33239     1     4      <NA>   alabama    417   1877     4.5     12.4     30.6  2.467742         2.67 3.535055
5 alabama -87.57087 30.32665     1     5      <NA>   alabama    417   1877     4.5     12.4     30.6  2.467742         2.67 3.535055
6 alabama -87.58806 30.32665     1     6      <NA>   alabama    417   1877     4.5     12.4     30.6  2.467742         2.67 3.535055
```

Now, the first time through this might seem quite daunting. However, as long as you've followed the steps I gave earlier, and you have an object called "Total" which can be manipulated and plotted with the code below; the figure is shown in figure 75. Again, to see your options for colors, run `colors()`.

```
p <- ggplot()
p <- p + geom_polygon(data=Total, aes(x=long, y=lat, group = group, fill= bwRatio), colour="white")

p <- p + scale_fill_continuous(low = "thistle2", high = "darkred", guide="colorbar")

P1 <- p + theme_bw()  + labs(fill = "Black to White Incarceration Rates \n Weighted by Relative Population"
                             ,title = "State Incarceration Rates by Race, 2010", x="", y="")
P1 <- P1 + scale_y_continuous(breaks=c())
P1 <- P1 + scale_x_continuous(breaks=c()) + theme(panel.border =  element_blank())
```

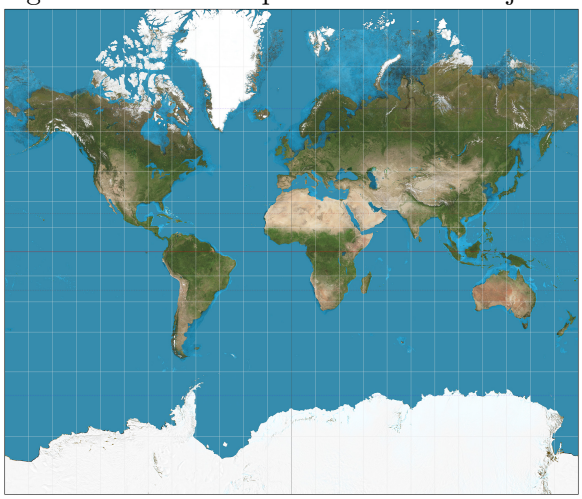
Figure 75: Our Map



## 8.2 Projections

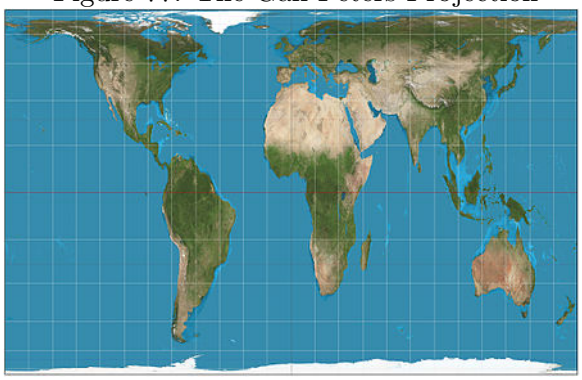
When working with maps and spatial data, you'll want to be careful how the curvature of the earth is projected into two dimensions. In case you didn't know, the world has been misrepresented for years. For more, you can see the Cartographers for Social Equality, from the West Wing. The Mercator Projection, shown in figure 76 distorts the northern countries, making Greenland appear much larger than both Africa and South America. In fact, South America is approximately nine times larger.

Figure 76: The Deceptive Mercator Projection



This is better shown in the Gall-Peters projection, in figure 77.

Figure 77: The Gall-Peters Projection



## 8.3 Pulling Maps from the Web with `qmap()`

But, that's not even the coolest thing that we can do using maps in R; using the `qmap()` from the `ggmap()` package, we can pull from already existing and easy to find maps.

```
library(ggmap)
BTown <- qmap('Bloomington, IN 47405', zoom = 16)
BTown #Where we are right now.

IU <- qmap('Indiana University', zoom = 15)
IU #Google's good at this.
```

Figure 78: `qmap('Bloomington, IN 47405', zoom = 16)`

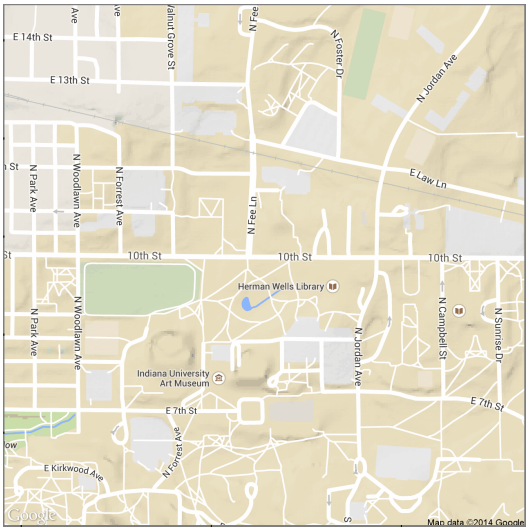
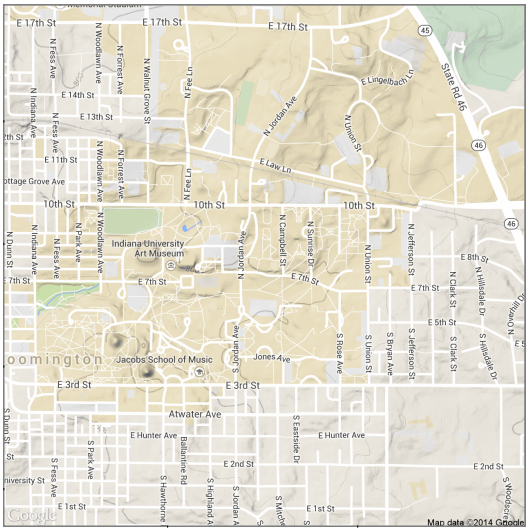
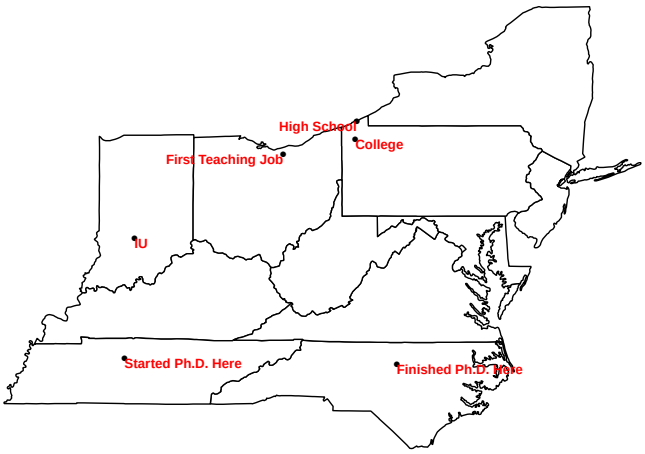



Figure 79: `qmap('Indiana University', zoom = 15)`



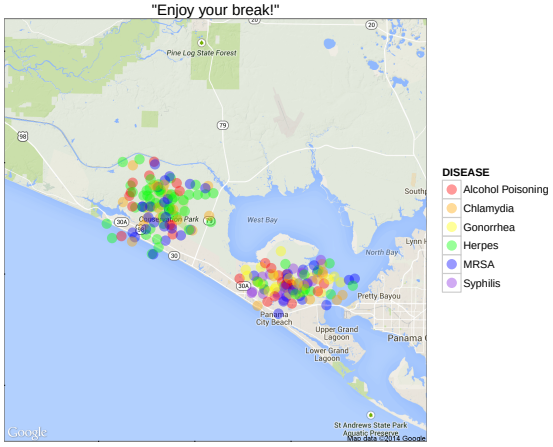
You can even use this to make custom maps, like the one below which shows just how spatial my education has been.

Figure 80: Where I've Lived:




Then one time when I was showing students how to make maps, I had a student who was giving me a hard time about where I (wasn't) going on "Spring Break." So, I made up some data (though I am not sure I told him that) and used it in my lab; guess where he was heading off to visit.

```
CDC <- read.csv("http://mypage.iu.edu/~cdesante/cdc.csv", as.is = T)
#LON = Longitude; LAT = Latitude of where the person slept the night before.
#Disease = Reason for Seeking Medical Care
attach(CDC)
MAP <- qmap('30.255096, -85.865909', zoom= 11)
PCB <- MAP
custom.colors  <- c("red", "orange" , "yellow", "green", "blue", "blueviolet", "violet")
PCB + geom_point(aes( x = LON , y = LAT , colour =  DISEASE  ), size =  4.5 , alpha = 0.40
) + scale_colour_manual(values = custom.colors)
#Final Plot:
PCB <- PCB + geom_point(aes( x = LON , y = LAT , colour =  DISEASE  ,
                        title = "Enjoy your break!"), size =  4.5 , alpha = 0.40 ,
) + scale_colour_manual(values = custom.colors)
```



## 8.4 Maps at the County Level using FIPS Codes

```
#### INDIANA COUNTY MAPS ########

install.packages("ggplot2", dependencies = TRUE)
install.packages("maps", dependencies = TRUE)
install.packages("mapdata")
install.packages("maptools")
install.packages("mapproj")
install.packages("ggmap", dependencies = TRUE)
install.packages("Hmisc", dependencies = TRUE)
library(maps)
library(mapdata)
library(maptools)
library(scales)
library(RColorBrewer)
library(ggplot2)
library(Hmisc)
theme_clean <- function(base_size = 12) {
  require(grid)
  theme_grey(base_size) %+replace%
    theme(
      axis.title      =   element_blank(),
      axis.text       =   element_blank(),
      panel.background    =   element_blank(),
      panel.grid      =   element_blank(),
      axis.ticks.length   =   unit(0,"cm"),
      axis.ticks.margin   =   unit(0,"cm"),
      panel.margin    =   unit(0,"lines"),
      plot.margin     =   unit(c(0,0,0,0),"lines"),
      complete = TRUE
    )
}

DATA <- read.csv("http://pages.iu.edu/~cdesante/indiana.csv")

head(DATA)
dim(DATA)
DATA <- DATA[, c(2,12   )]
head(DATA)
colnames(DATA)  <- c("FIPS",  "Earnings")
head(DATA)
data(county.fips)
mapcounties <- map_data("county")
head(mapcounties)

mapcounties$county <- with(mapcounties , paste(region, subregion, sep = ","))
head(mapcounties)

mergedata <- merge(mapcounties, county.fips, by.x = "county", by.y = "polyname")
```

```
head(mergedata)

Indiana <- mergedata[mergedata$region == "indiana" , ]
head(Indiana)


Final.Data <- merge(DATA, Indiana, by.x = "FIPS", by.y = "fips")
head(Final.Data)


##Colour Choices:
colors <- brewer.pal(5,"RdYlBu")
Final.Data$colorBuckets <- as.numeric(cut2( Final.Data$Earnings, g= 10) )
table(Final.Data$colorBuckets)
pal <- brewer.pal(10,"RdYlGn")
pal <- pal[10:1]
###################################

head(Final.Data)
Final.Data <- Final.Data[order(Final.Data$order),]
Indiana <- Indiana[order(Indiana$order),]

map <- ggplot(Final.Data, aes(long, lat, group= county)
) + geom_polygon(aes(fill=factor(colorBuckets)) ) +
  scale_fill_manual(values=pal)

map + geom_path(data = Indiana, colour = "white",
size = 1, alpha = .5) + labs(fill="Earnings Per\n County") +   theme_clean(
    ) +  guides(fill = guide_legend( "Earnings Decile") )
```