Training Outline

Jarad Niemi

2014-12-04

International Society for Disease Surveillance

Introduction to R for Biosurveillance

Pre-conference Training

Introduction to R

- Introduction to R
- Using R as a calculator
- Reading data into R
- Reading directly from Excel
- Descriptive statistics
- Subsetting the data
- Graphical statistics
- Getting help

Data Vizualization I (ggplot2)

- Data types
- Data frames
- Factors
- Dates
- Reshaping data frames in R
- Aggregating data frames in R
- Basics of ggplot2
- Customizing (colors, characters, lines)
- Getting help on ggplot2

Data Vizualization II (ggplot2)

- Workflow
- Workflow for constructing a graph
- Faceting
- Subsetting
- Making professional graphics
- Exporting graphs

Advanced biosurveillance

- Exporting tables
- Maps
- Packages
- Packages for Surveillance
- Functions
- R in batch
- Shiny apps

Introduction to R

Jarad Niemi

2014-12-04

Contents

Introduction to R

Detailed introduction

For an extremely detailed introduction, please see

```
help.start()
```

In this documentation, the above command will be executed at the command prompt, see below.

R interface

In contrast to many other statistical software packages that use a point-and-click interface, e.g. SPSS, JMP, Stata, etc, R has a command-line interface. The command line has a command prompt, e.g. `>`, see below.

```
>
```

This means, that you will be entering commands on this command line and hitting enter to execute them, e.g.

```
help()
```

Use the **up arrow** to recover past commands.

```
hepl()
help() # Use up arrow and fix
```

R GUI (or RStudio)

Most likely, you are using a graphical user interface (GUI) and therefore, in addition, to the command line, you also have a windowed version of R with some point-and-click options, e.g. File, Edit, and Help.

In particular, there is an editor to create a new R script. So rather than entering commands on the command line, you will write commands in the script and then send those commands to the command line using `Ctrl-R` (PC) or `Command-Enter` (Mac).

```
a = 1
b = 2
a+b
```

```
## [1] 3
```

Multiple lines can be run in sequence by selecting them and then using `Ctrl-R` (PC) or `Command-Enter` (Mac).

Intro Activity

One of the most effective ways to use this documentation is to cut-and-paste the commands into a script and then execute them.

Cut-and-paste the following commands into a **new script** and then run those commands directly from the script using `Ctrl-R` (PC) or `Command-Enter` (Mac).

```
x = 1:10
y = rep(c(1,2), each=5)
m = lm(y~x)
s = summary(m)
```

Now, look at the result of each line

```
x
y
m
s
s$r.squared
```

When you have completed the activity, compare your results to the solutions.

Using R as a calculator

Basic calculator operations

All basic calculator operations can be performed in R.

```
1+2
```

```
## [1] 3
```

```
1-2
```

```
## [1] -1
```

```
1/2
```

```
## [1] 0.5
```

```
1*2
```

```
## [1] 2
```

For now, you can ignore the [1] at the beginning of the line, we'll learn about that when we get to vectors.

Advanced calculator operations

Many advanced calculator operations are also available.

```
(1+3)*2+100^2 # standard order of operations
```

```
## [1] 10008
```

```
sin(2*pi) # the result is in scientific notation, i.e. -2.449294 x 10^-16
```

```
## [1] -2.449294e-16
```

```
sqrt(4)
```

```
## [1] 2
```

```
10^2
```

```
## [1] 100
```

```
log(10) # the default is base e
```

```
## [1] 2.302585
```

```
log(10,base=10)
```

```
## [1] 1
```

Using variables

A real advantage to using R rather than a calculator (or calculator app) is the ability to store quantities using variables.

```
a = 1
b = 2
a+b
```

```
## [1] 3
```

```
a-b
```

```
## [1] -1
```

```
a/b
```

```
## [1] 0.5
```

```
a*b
```

```
## [1] 2
```

Assignment operators =, <-, and ->

When assign variables values, you can also use arrows <- and -> and you will often see this in code, e.g.

```
a <- 1
2 -> b
c = 3 # is the same as <-
```

Now print them.

```
a
```

```
## [1] 1
```

```
b
```

```
## [1] 2
```

```
c
```

```
## [1] 3
```

Using informative variable names

While using variables alone is useful, it is much more useful to use informative variables names.

```
population = 1000
number_infected = 200
deaths = 3

death_rate = deaths / number_infected
attack_rate = number_infected / population

death_rate
```

```
## [1] 0.015
```

```
attack_rate
```

```
## [1] 0.2
```

Calculator Activity

Bayes' Rule

Suppose an individual tests positive for a disease, what is the probability the individual has the disease? Let

- D indicates the individual has the disease
- N means the individual does not have the disease
- $+$ indicates a positive test result
- $-$ indicates a negative test

The above probability can be calculated using Bayes' Rule:

$$P(D|+) = \frac{P(+|D)P(D)}{P(+|D)P(D) + P(+|N)P(N)} = \frac{P(+|D)P(D)}{P(+|D)P(D) + (1 - P(-|N)) \times (1 - P(D))}$$

where

- $P(+|D)$ is the sensitivity of the test
- $P(-|N)$ is the specificity of the test
- $P(D)$ is the prevalence of the disease

Calculate the probability the individual has the disease if the test is positive when

- the specificity of the test is 0.99,
- the sensitivity of the test is 0.95, and
- the prevalence of the disease is 0.001.

When you have completed the activity, compare your results to the solutions.

Reading data into R

In this section, we will learn how to read in csv or Excel files into R. We focus on csv files because less can go wrong.

Changing your working directory

One of the first tasks after starting R is to change the working directory. To set,

- in RStudio: Session > Set Working Directory > Choose Directory... (Ctrl + Shift + H)
- in R GUI (Windows): File > Change Dir...
- in R GUI (Mac): Misc > Change Working Directory...

Or, you can just run the following command

```
setwd(choose.dir(getwd()))
```

Make sure you have write access to this directory.

Installing and loading a package

Much of the functionality of R is contained in packages. The first time these packages are used, they need to be installed, e.g. to install a package from CRAN use

```
install.packages("plyr")
```

Once installed, they needed to loaded into the R session.

```
library(plyr)
```

Load and start this workshop

First load the package

```
library(ISDSWorkshop)
```

This package contains a function to help you get started, so run that function.

```
workshop()
```

This function did three things

1. It opened the workshop outline in a web browser
2. It created a set of .csv data files in your working directory
3. It created a set of .R scripts in your working directory.

Open an R script

As we progress through the workshop, the code for a particular module will be available in the R script for that module.

In R, open the module called `intro.R` and scroll down to the `workshop()` command. From here on out, as I run commands you should run the commands as well by using Ctrl-R (Windows) or Command-Enter (Mac) with the appropriate line(s) highlighted.

You will notice that nothing after a `#` will be evaluated by R. That is because the `#` character indicates a comment in the code. For example,

```
# This is just a comment.
1+1 # So is this
```

```
## [1] 2
```

```
# 1+2
```

Reading a csv file into R

csv stands for comma-separated value file and is a standard file format for data. To read this in to R, use

```
GI = read.csv("GI.csv")
```

This created a `data.frame` object in R called GI.

read.table

The `read.table()` function is a more general function for reading data into R and it has many options. We could have gotten the same results if we had used the following code:

```
GI2 = read.table("GI.csv",
                 header=TRUE, # There is a header.
                 sep=",")     # The column delimiter is a comma.
```

To check if the two data sets are equal, use the following

```
all.equal(GI, GI2)
```

```
## [1] TRUE
```

Exploring the data set

There are a number of functions that will provide information about a data.frame. Here are a few:

```
dim(GI)
```

```
## [1] 21244     9
```

```
names(GI)
```

```
## [1] "id"              "date"            "facility"        "icd9"
## [5] "age"             "zipcode"         "chief_complaint" "syndrome"
## [9] "gender"
```

```
head(GI)
```

```
##           id       date facility   icd9 age zipcode chief_complaint
## 1 1001301988 2005-02-28       67 787.01   7   21075        Abd Pain
## 2 1001829757 2005-02-28       67 558.90  41   20721   upset stomach
## 3 1001581758 2005-02-28      123 787.91   2   22152        diarrhea
## 4 1001950471 2005-02-28      123 787.91  71   22060        ABD PAIN
## 5 1001076304 2005-02-28      309 558.90  28   21702   LOWER AD PAIN
## 6 1001087075 2005-03-01       66 787.30  43   20762       Rctal Bld
```

```
##   syndrome gender
## 1       GI   Male
## 2       GI Female
## 3       GI   Male
## 4       GI   Male
## 5       GI Female
## 6       GI Female
```

```
tail(GI)
```

```
##               id       date facility   icd9 age zipcode
## 21239 1001403487 2008-12-30     6200 787.02   4   20169
## 21240 1001392877 2008-12-30      123 787.01   6   22153
## 21241 1001887911 2008-12-30       37 535.50  72   22033
## 21242 1001196061 2008-12-30     6200 536.20  59   20155
## 21243 1001067104 2008-12-30       67 558.90  80   22202
## 21244 1001396039 2008-12-30      123 787.03  12   20112
##                      chief_complaint syndrome gender
## 21239            LUQ ABDOMINAL PAIN       GI Female
## 21240                         N/V/D       GI   Male
## 21241               VOMITING CHILLS       GI   Male
## 21242 LEFT CHEST AND ABDOMINAL PAIN       GI   Male
## 21243                      ABD PAIN       GI Female
## 21244                      Diarrhea       GI   Male
```

Reading an Excel xlsx file into R

Reading an Excel xlsx file into R is done using the `read.xlsx` function from the xlsx R package. Unfortunately many scenarios can cause this process to not work. Thus, we do not focus on it in an introductory R course. When it works, it looks like this

```
d = read.xlsx("filename.xlsx", sheetIndex=1)
```

or

```
d = read.xlsx("filename.xlsx", sheetName="sheetName")
```

If these don't work, you can `Save as...` a csv file in Excel.

Activity

If you brought your own Excel file, open it and save a sheet as a csv file in your working directory. If you brought your own csv file, save it in your working directory. If you did not bring your own file, use the `fluTrends.csv` file in your working directory.

Try to use the read.csv function to read the file into R. There are a number of different options in the csv file that may be useful:

```
d = read.table("filename.csv", # Make sure to change filename>to your filename and
                               # make sure you use the extension, e.g. .csv.
               header = TRUE,  # If there is no header column, change TRUE to FALSE.
```

```
                sep =",",       # The column delimiter is a comma.
                skip = 0        # Skip this many lines before starting to read the file
                )
```

When you have completed the activity, compare your results to the solutions.

Descriptive statistics

When reading your data set into R, you will likely want to perform some descriptive statistics. The single most useful command to assess the whole data set is the `summary()` command:

```
summary(GI)
```

```
##        id                   date          facility         icd9
##  Min.   :1.001e+09   2007-01-31:   57   Min.   :  37   Min.   :    3.0
##  1st Qu.:1.001e+09   2007-01-29:   55   1st Qu.:  66   1st Qu.:  558.9
##  Median :1.001e+09   2007-01-16:   52   Median :  67   Median :  787.0
##  Mean   :1.001e+09   2007-02-28:   52   Mean   :1102   Mean   : 1043.2
##  3rd Qu.:1.002e+09   2007-01-24:   50   3rd Qu.: 309   3rd Qu.:  787.3
##  Max.   :1.002e+09   2007-01-17:   45   Max.   :7298   Max.   :78791.0
##                      (Other)   :20933
##       age            zipcode            chief_complaint  syndrome
##  Min.   :  0.00   Min.   :20001   Abd Pain       : 1390   GI:21244
##  1st Qu.:  8.00   1st Qu.:20747   ABD PAIN       : 1074
##  Median : 27.00   Median :21740   Vomiting       :  661
##  Mean   : 29.98   Mean   :21420   VOMITING       :  563
##  3rd Qu.: 47.00   3rd Qu.:22182   ABDOMINAL PAIN:  452
##  Max.   :157.00   Max.   :22556   vomiting       :  423
##                                   (Other)        :16681
##     gender
##  Female:10653
##  Male  :10591
##
##
##
##
##
```

Descriptive statistics for continuous (numeric) variables

To access a single column in the `data.frame` use a dollar sign ($).

```
GI$age
GI[,5] # Since age is the 5th column
```

Here are a number of descriptive statistics for *age*

```
min(GI$age)
```

```
## [1] 0
```

```r
max(GI$age)
```

```
## [1] 157
```

```r
mean(GI$age)
```

```
## [1] 29.98221
```

```r
median(GI$age)
```

```
## [1] 27
```

```r
quantile(GI$age, c(.025,.25,.5,.75,.975))
```

```
##   2.5%    25%    50%    75%  97.5%
##  0.075  8.000 27.000 47.000 81.000
```

Anything look odd here?

Descriptive statistics for categorical (non-numeric) variables

The `table()` function provides the number of observations at each level of the categorical variable.

```r
table(GI$gender)
```

```
##
## Female   Male
##  10653  10591
```

which is the same as `summary()` if the variable is not coded as numeric

```r
summary(GI$gender)
```

```
## Female   Male
##  10653  10591
```

If the variable is coded as numeric, but is really a categorical variable, then you can still use table, but summary won't give you the correct result.

```r
table(GI$facility)
```

```
##
##   37   66   67  123  255  256  259  309  390  413  420  522  703 6200 6201
## 3571 2950 4281 4408   41  575    1  325  661  668  100   67  178 1423 1928
## 7298
##   67
```

```
summary(GI$facility)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      37      66      67    1102     309    7298
```

Apparently there is only 1 observation from facility 259, was that a typo?

Subsetting the data

Rather than having descriptive statistics for the dataset as a whole, we may be interested in descriptive statistics for a `subset` of the data.

The following code creates a new `data.frame` that only contains observations from facility 37:

```
GI_37 = subset(GI, facility==37)
nrow(GI_37) # Number of rows (observations) in the new data set
```

```
## [1] 3571
```

Notice the double equal sign!

The following code creates a new `data.frame` that only contains observations with chief_complaint "Abd Pain":

```
GI_AbdPain = subset(GI, chief_complaint == "Abd Pain")
nrow(GI_AbdPain)
```

```
## [1] 1390
```

Notice that when the variable is a not numeric, we need to put level in quotes.

Alternative way to subset

```
GI_37a = GI[GI$facility==37,]
all.equal(GI_37, GI_37a)
```

```
## [1] TRUE
```

```
GI_AbdPain1 = GI[GI$chief_complaint == "Abd Pain",]
all.equal(GI_AbdPain, GI_AbdPain1)
```

```
## [1] TRUE
```

Advanced subsetting

We can subset continuous variables using other logical statements.

```r
subset(GI, age <   5)
subset(GI, age >= 60)
subset(GI, chief_complaint %in% c("Abd Pain","ABD PAIN"))
subset(GI, !(facility %in% c(37,66))) # facility is NOT 37 or 66
```

Descriptive statistics on the subset

Now we can calculate descriptive statistics on this subset, e.g.

```r
summary(GI_37$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     3.0    19.0    36.0    39.3    58.0   139.0
```

```r
summary(GI_AbdPain$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    7.00   27.00   28.61   45.00  157.00
```

Activity

Find the min, max, mean, and median age for zipcode 20032.

When you have completed the activity, compare your results to the solutions.

Graphical statistics

Here we focus on the graphical options for the base package in R. Later we will use the ggplot2 package:

- Histograms (`hist`)
- Boxplots (`boxplot`)
- Scatter plots (`plot`)
- Bar charts (`barplot`)

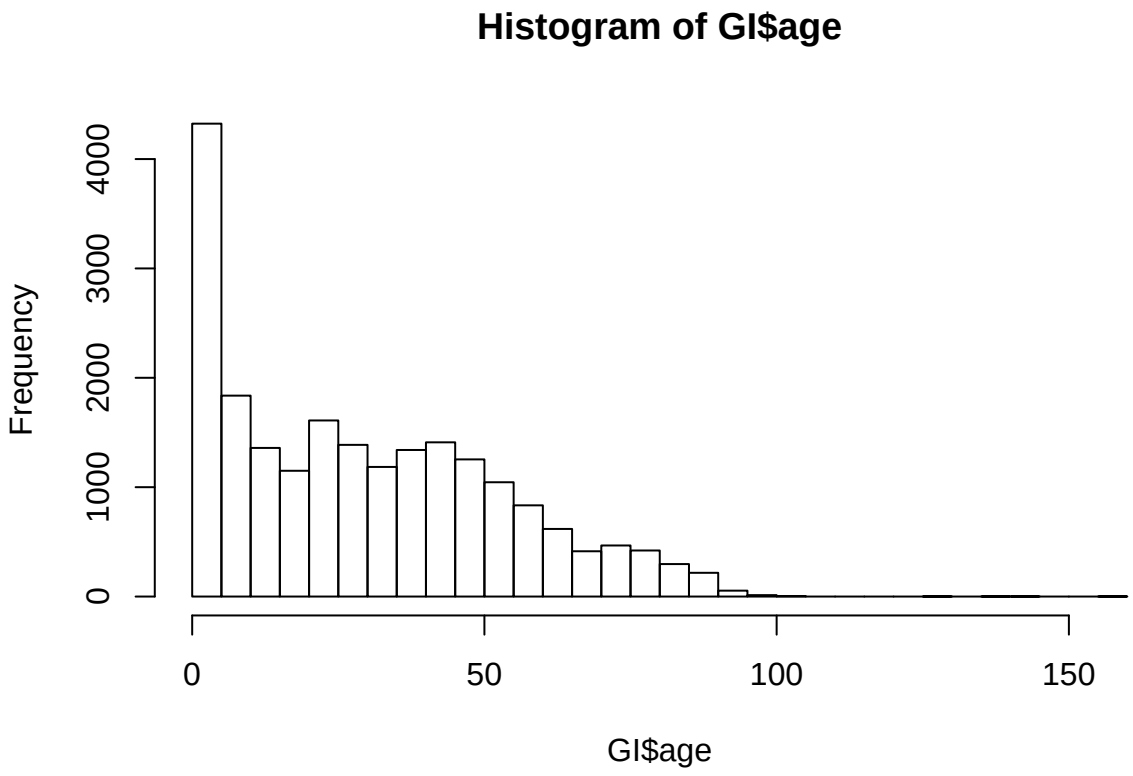
Histograms

For continuous variables, histograms are useful for visualizing the distribution of the variable

```r
hist(GI$age)
```

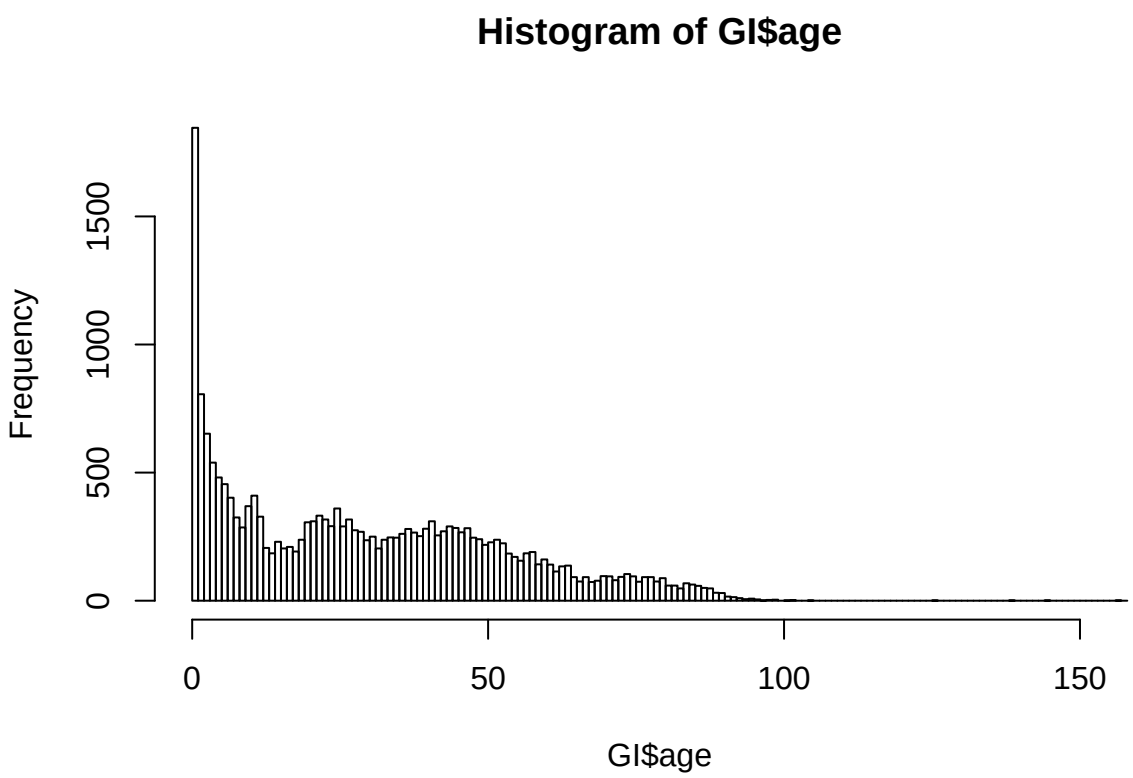
When there is a lot of data, you will typically want more bins

```
hist(GI$age, 50)
```



You can also specify your own bins

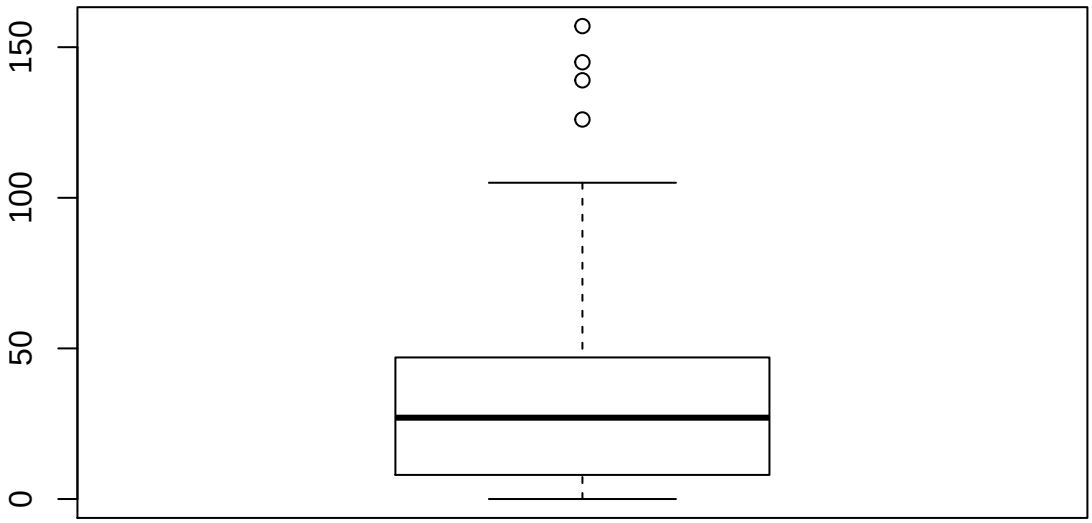
```
hist(GI$age, 0:158)
```



Boxplots

Boxplots are another way to visualize the distribution for continuous variables.

```
boxplot(GI$age)
```

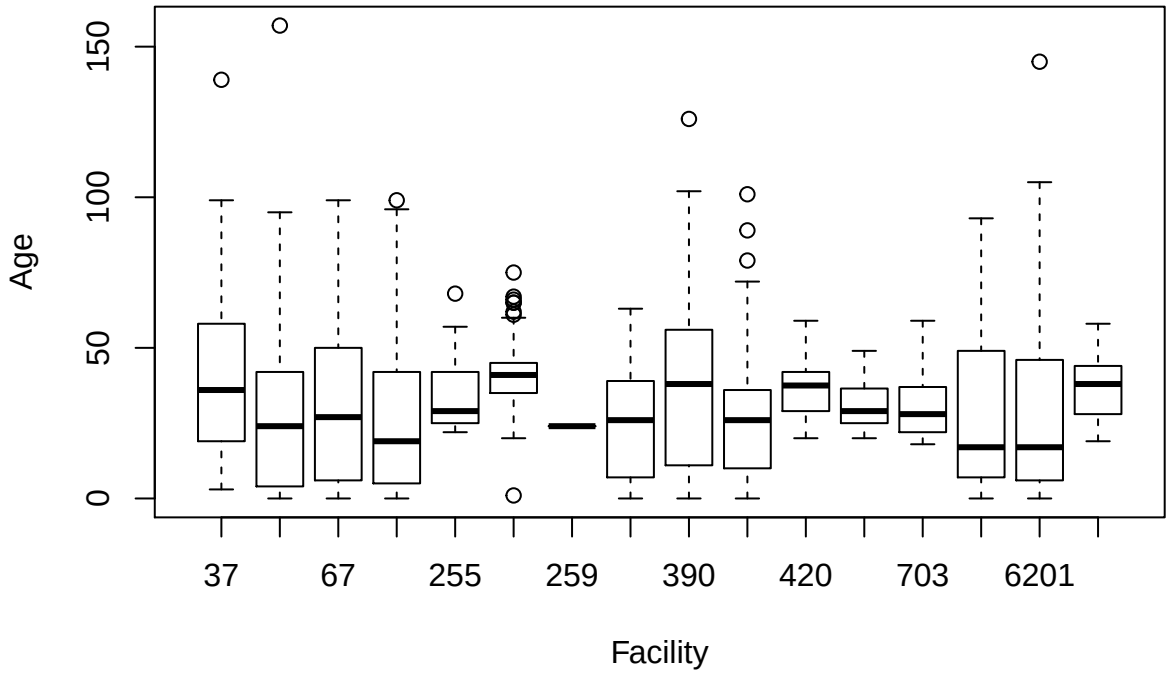


Now we can see the outliers.

Multiple boxplots

Here we create separate boxplots for each facility and label the x and y axes.

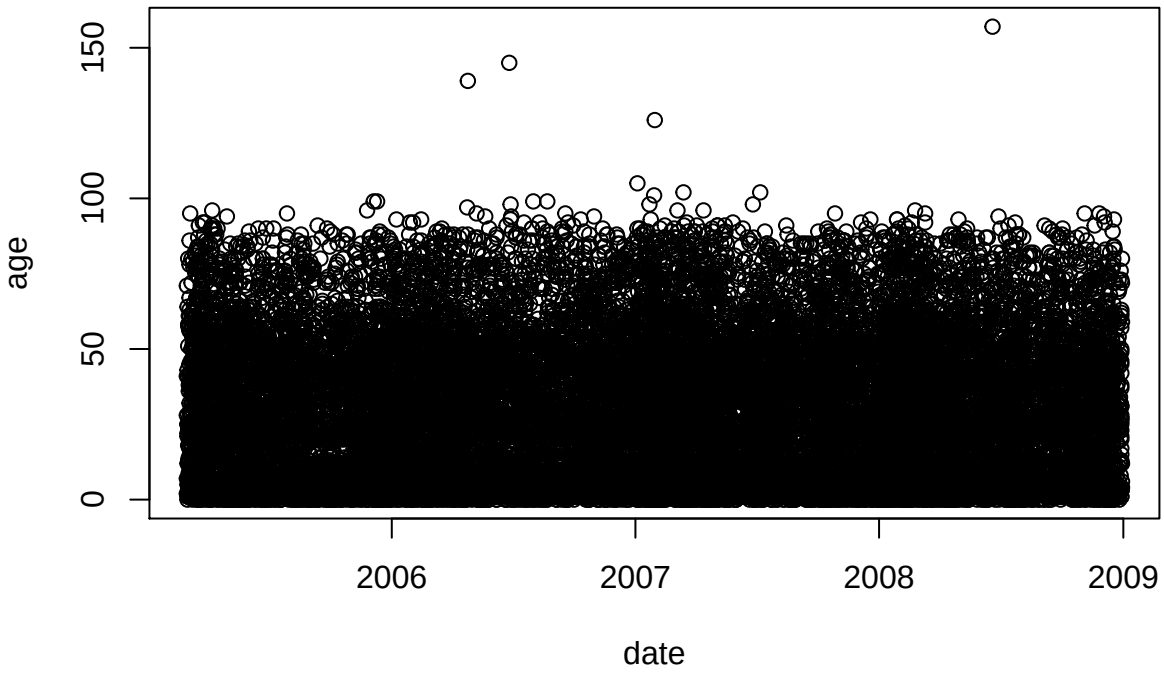
```
boxplot(age~facility, GI, xlab="Facility", ylab="Age")
```



Scatterplots

Scatterplots are useful for looking at the relationship of two continuous variables.

```
GI$date = as.Date(GI$date)
plot(age~date, GI)
```

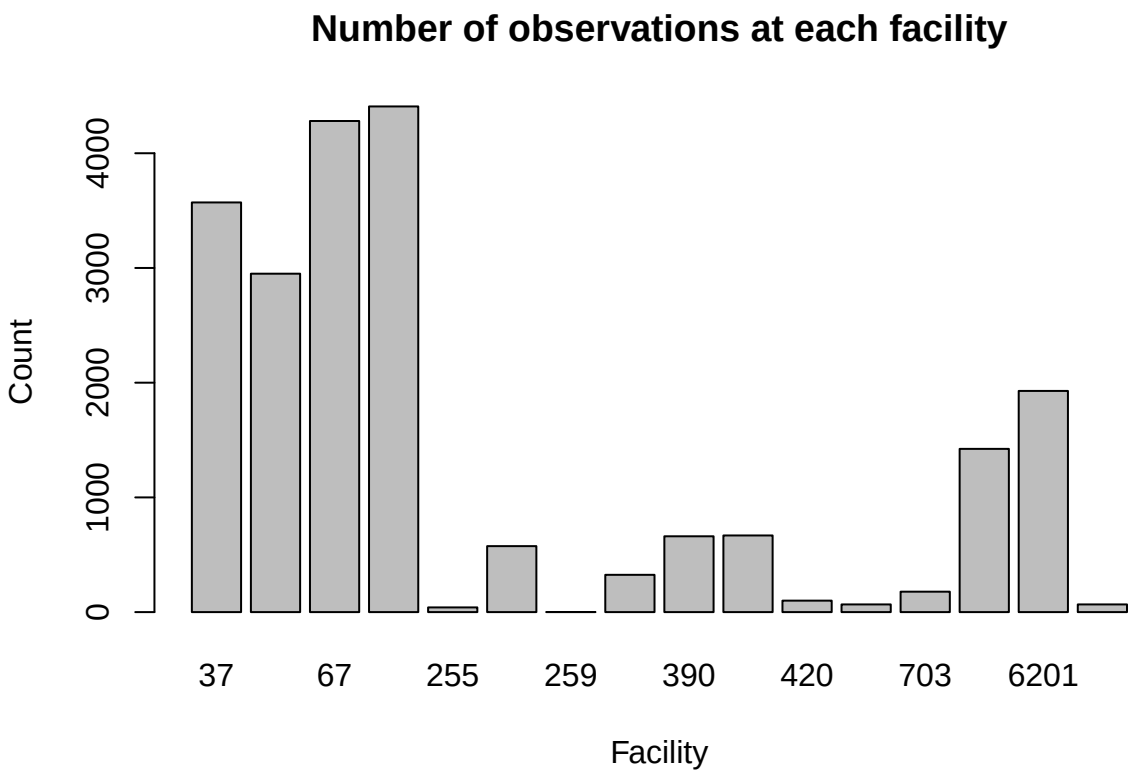


We will talk more later about dealing with dates later.

Bar charts

For looking at the counts of categorical variables, we use bar charts.

```
counts = table(GI$facility)
barplot(counts, xlab="Facility", ylab="Count", main="Number of observations at each facility")
```



Activity

Construct a histogram and boxplot for age at facility 37.

Construct a bar chart for the zipcode at facility 37.

When you have completed the activity, compare your results to the solutions.

Getting help

As you work with R, there will be many times when you need to get help.

My basic approach is

1. Use the help contained within R
2. Perform an internet search for an answer
3. Find somebody else who knows

In both cases, knowing the R keywords, e.g. a function name, will be extremely helpful.

Help within R I

If you know the function name, then you can use `?<function>`, e.g.

```
?mean
```

The structure of help is - Description: quick description of what the function does - Usage: the arguments, their order, and default values (if any) - Arguments: more thorough description about the arguments - Value: what the funtion returns - See Also: similar functions - Examples: examples of how to use the function

Help within R II

If you cannot remember the function name, then you can use `help.search("<something>")`, e.g.

```
help.search("mean")
```

Depending on how many packages you have loaded, you will find a lot or a little here.

Internet search for R help

I typically do `<something> R`, e.g.

```
calculate mean R
```

The useful sites are

- http://www.cookbook-r.com/
- http://www.r-tutor.com/r-introduction
- http://www.statmethods.net/
- http://stackoverflow.com/questions/tagged/r
- http://www.ats.ucla.edu/stat/r/

Data Visualization I (ggplot2)

Jarad Niemi

2014-12-04

Contents

Before we get back into graphics, it is important to understand some of the fundamentals behind what R is doing.

Please open the `graphics.R` script in your working directory.

Data types

Objects in R can be broadly classified as according to their dimensions:

- scalar
- vector
- matrix
- array (higher dimensionl matrix)

and according to the type of variable they contain:

- numeric
- character
- logical
- factor

Scalars

Scalars only have a single value assigned to the object in R.

```
a = 3.14159265
b = "ISDS Workshop"
c = TRUE
```

Print the objects

```
a
```

```
## [1] 3.141593
```

```
b
```

```
## [1] "ISDS Workshop"
```

```
c
```

```
## [1] TRUE
```

Vectors

The `c()` function creates a vector in R

```
a = c(1,2,-5,3.6)
b = c("ISDS","Workshop")
c = c(TRUE, FALSE, TRUE)
```

To determine the length of a vector in R use `length()`

```
length(a)
```

```
## [1] 4
```

```
length(b)
```

```
## [1] 2
```

```
length(c)
```

```
## [1] 3
```

Vector construction

Create a numeric vector that is a sequence using : or `seq()`.

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
5:-2
```

```
## [1]  5  4  3  2  1  0 -1 -2
```

```
seq(2,23,by=3)
```

```
## [1]  2  5  8 11 14 17 20 23
```

Another useful function to create vectors is `rep()`

```r
rep(1:4, times = 2)
```

```
## [1] 1 2 3 4 1 2 3 4
```

```r
rep(1:4, each  = 2)
```

```
## [1] 1 1 2 2 3 3 4 4
```

```r
rep(1:4, each  = 2, times = 2)
```

```
##  [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

Accessing vector elements

Elements of a vector can be access using brackets, e.g. [index].

```r
a = c("one","two","three","four","five")
a[1]
```

```
## [1] "one"
```

```r
a[2:4]
```

```
## [1] "two"   "three" "four"
```

```r
a[c(3,5)]
```

```
## [1] "three" "five"
```

```r
a[rep(3,4)]
```

```
## [1] "three" "three" "three" "three"
```

Alternatively we can access elements using a logical vector where only TRUE elements are accessed.

```r
a[c(TRUE, TRUE, FALSE, FALSE, FALSE)]
```

```
## [1] "one" "two"
```

You can also remove elements using a negative sign -.

```r
a[-1]
```

```
## [1] "two"   "three" "four"  "five"
```

Modifying elements of a vector

You can assign new values to elements in a vector using =.

```
a[2] = "twenty-two"
a
```

```
## [1] "one"        "twenty-two" "three"      "four"       "five"
```

```
a[3:4] = "three-four"
a
```

```
## [1] "one"        "twenty-two" "three-four" "three-four" "five"
```

```
a[c(3,5)] =c("thirty-three","fifty-five")
a
```

```
## [1] "one"          "twenty-two"   "thirty-three" "three-four"
## [5] "fifty-five"
```

Notice that the second example assigned the value to both elements in the vector.

Matrices

Matrices can be constructed using `cbind()`, `rbind()`, and `matrix()`:

```
m1 = cbind(c(1,2), c(3,4))       # Row bind
m2 = rbind(c(1,3), c(2,4))       # Column bind
m1
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
all.equal(m1, m2)
```

```
## [1] TRUE
```

```
m3 = matrix(1:4, nrow=2, ncol=2)
all.equal(m1,m3)
```

```
## [1] TRUE
```

```
m4 = matrix(1:4, nrow=2, ncol=2, byrow=TRUE)
all.equal(m3, m4)
```

```
## [1] "Mean relative difference: 0.4"
```

```
m3
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
m4
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

Accessing matrix elements

Elements of a matrix can be accessed using brackets separated by a comma, e.g. [row index, column index].

```
m = matrix(1:12, nrow=3, ncol=4)
m
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
m[2,3]
```

```
## [1] 8
```

Multiple elements can be accessed at once

```
m[1:2,3:4]
```

```
##      [,1] [,2]
## [1,]    7   10
## [2,]    8   11
```

If no row (column) index is provided, then the whole row (column) is accessed.

```
m[1:2,]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
```

Like vectors, you can eliminate rows (or columns)

```
m[-c(3,4),]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
```

Be careful not to forget the comma

```r
m[1:2]
```

```
## [1] 1 2
```

You can also construct a high-dimensional array using the `array()` function.

Cannot mix types

You cannot mix types within a vector, matrix, or array

```r
c(1,"a")
```

```
## [1] "1" "a"
```

The number 1 is in quotes indicating that R is treating it as a character rather than a numeric.

```r
c(TRUE, 1, FALSE)
```

```
## [1] 1 1 0
```

The logicals are converted to numeric (0 for FALSE and 1 for TRUE).

```r
c(TRUE, 1, "a")
```

```
## [1] "TRUE" "1"    "a"
```

Everything is converted to a character.

Activity

Reconstruct the following matrix using the `matrix()` function, then

1. Print the element in the 3rd-row and 4th column
2. Print the 2nd column
3. Print all but the 4th row

```r
m = rbind(c(1, 12, 8, 6),
          c(4, 10, 2, 9),
          c(11, 3, 5, 7))
m
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1   12    8    6
## [2,]    4   10    2    9
## [3,]   11    3    5    7
```

When you have completed the activity, compare your results to the solutions.

Data frames

A `data.frame` is a special type of matrix that allows different data types in different columns.

We have already seen a data frame with our GI data set. Let's read this data in again and take a look.

```
GI = read.csv("GI.csv")
dim(GI)
```

```
## [1] 21244     9
```

If the above didn't work, run the following first, then retry.

```
library(ISDSWorkshop)
workshop(write_scripts=FALSE, launch_index=FALSE)
```

Access `data.frame` elements

data.frames can be accessed just like matrices, e.g. [row index, column index].

```
GI[1:2, 3:4]
```

```
##   facility   icd9
## 1       67 787.01
## 2       67 558.90
```

data.frames can also be accessed by column names

```
GI[1:2, c("facility","icd9","gender")]
```

```
##   facility   icd9 gender
## 1       67 787.01   Male
## 2       67 558.90 Female
```

Different data types in different columns

The function `str()` allows you to see the structure of any object in R

```
str(GI)
```

```
## 'data.frame':    21244 obs. of  9 variables:
##  $ id             : int  1001301988 1001829757 1001581758 1001950471 1001076304 1001087075 100153694
##  $ date           : Factor w/ 1399 levels "2005-02-28","2005-03-01",..: 1 1 1 1 1 2 2 2 2 2 ...
##  $ facility       : int  67 67 123 123 309 66 6201 67 66 66 ...
##  $ icd9           : num  787 559 788 788 559 ...
##  $ age            : int  7 41 2 71 28 43 12 1 25 64 ...
##  $ zipcode        : int  21075 20721 22152 22060 21702 20762 22192 20121 20772 20602 ...
##  $ chief_complaint: Factor w/ 2936 levels " /v"," /V/D",..: 315 2600 1100 15 1741 2238 640 2480 1833
##  $ syndrome       : Factor w/ 1 level "GI": 1 1 1 1 1 1 1 1 1 1 ...
##  $ gender         : Factor w/ 2 levels "Female","Male": 2 1 2 2 1 1 2 2 2 2 ...
```

Factor

A factor is a data type that represents a categorical variable.

The default is for any character vector to be converted to a factor when read using `read.csv()` or `read.table()`.

Internally, R codes a factor as an integer and then keeps a table that contains the conversion from that integer into the actual value of the factor.

```
nlevels(GI$gender)
```

```
## [1] 2
```

```
levels(GI$gender)
```

```
## [1] "Female" "Male"
```

```
GI$gender[1:3]
```

```
## [1] Male   Female Male
## Levels: Female Male
```

```
as.numeric(GI$gender[1:3])
```

```
## [1] 2 1 2
```

Converting a numeric variable into a factor

When a categorical variable is encoded as a numeric variable in the original data set, R reads them in as numeric. To convert them to a factor use `as.factor()`.

```
GI$facility = as.factor(GI$facility)
summary(GI$facility)
```

```
##   37   66   67  123  255  256  259  309  390  413  420  522  703 6200 6201
## 3571 2950 4281 4408   41  575    1  325  661  668  100   67  178 1423 1928
## 7298
##   67
```

Converting back to the original numeric variable

To obtain the original numeric variable use `as.character()` and `as.numeric()`

```
head(as.character(GI$facility))             # This returns the levels as a character vector
```

```
## [1] "67"  "67"  "123" "123" "309" "66"
```

```
head(as.numeric(as.character(GI$facility))) # This returns the original numeric factor levels
```

```
## [1]  67  67 123 123 309  66
```

Creating your own factor

Use the `cut()` function to create a factor from a continuous variable.

```
GI$ageC = cut(GI$age, c(-Inf, 5, 18, 45 ,60, Inf))
head(table(GI$ageC))
```

```
##
##  (-Inf,5]    (5,18]   (18,45]   (45,60] (60, Inf]
##      4324      3802      7476      3133      2509
```

This created a new variable in the GI `data.frame` called *ageC*. `Inf` represents infinity.

Dates

In order to use dates properly, they need to be converted into type `Date`.

```
GI$date = as.Date(GI$date)
str(GI$date)
```

```
##  Date[1:21244], format: "2005-02-28" "2005-02-28" "2005-02-28" "2005-02-28" ...
```

`as.Date()` will attempt to read dates as "%Y-%m-%d" then "%Y/%m/%d". If neither works, it will give an error.

```
?as.Date
```

You can specify other date patterns, e.g.

```
as.Date("12/09/14", format="%m/%d/%y")
```

Activity

Create a new variable in the GI data set called `icd9code` that cuts icd9 at 0, 140, 240, 280, 290, 320, 360, 390, 460, 520, 580, 630, 680, 710, 740, 760, 780, 800, and Inf. Find the icd9code that is the most numerous in the GI data set.

When you have completed the activity, compare your results to the solutions.

Reshaping data frames in R

There are two general representations of tabular data.

Wide:

```
##   week  GI  ILI
## 1    1 246  948
## 2    2 195 1020
## 3    3 212 1024
```

which is a succinct representation of the data

Long:

```
##   week syndrome count
## 1    1       GI   246
## 2    2       GI   195
## 3    3       GI   212
## 4    1      ILI   948
## 5    2      ILI  1020
## 6    3      ILI  1024
```

which is the form most statistical software wants, i.e. there is only one column for the response (count).

Wide to long

The `reshape2` package provides functions to convert between the two representations. First, we need to load the package

```
library(reshape2)
```

Create the wide `data.frame`:

```
d = data.frame(week=1:3, GI=c(246,195,212), ILI=c(948, 1020, 1024))
```

To turn the `data.frame` into long format using `melt()`.

```
m = melt(d, id.vars    = "week",      # The variables you want to remain on the columns
         variable.name = "syndrome", # The name for the variable column
         value.name    = "count")    # The name for the response column
m
```

```
##   week syndrome count
## 1    1       GI   246
## 2    2       GI   195
## 3    3       GI   212
## 4    1      ILI   948
## 5    2      ILI  1020
## 6    3      ILI  1024
```

Long to wide

If we want to convert back, use `dcast()`

```
dcast(m, week ~ syndrome) # Notice that we do not use the count (value) column at all
```

```
## Using count as value column: use value.var to override.
```

```
##   week  GI  ILI
## 1    1 246  948
## 2    2 195 1020
## 3    3 212 1024
```

Aggregating data frames in R

The GI data set that we have is already in long format and each row is an individual. We may want to aggregate this information. To do so, we will use the `ddply()` function in the `plyr` package.

```
library(plyr)
```

For example, perhaps we wanted to know the total number of GI or ILI cases across the 3 weeks:

```
ddply(m,                  # We need to use the melted version of the data set
      .(syndrome),        # Do the following for each syndrome
      summarize,          # Summarize
      total = sum(count)) # Calculate `total` which is the sum of count for each syndrome
```

```
##   syndrome total
## 1       GI   653
## 2      ILI  2992
```

Aggregating the GI data set

Let's aggregate the GI data set by week, gender, and age category.

First, we need to create weeks

```
GI$date = as.Date(GI$date) # Make sure the dates are actually dates
GI$week = cut(GI$date,
              breaks="weeks",
              start.on.monday=TRUE)
```

Now we can summarize

```
GI_count = ddply(GI,
                 .(week, gender, ageC),
                 summarize,
                 total = length(id))    # Count the number of lines in each week-gender-ageC combinatio
nrow(GI_count)
```

```
## [1] 2008
```

```
head(GI_count, 20)
```

```
##          week gender      ageC total
## 1  2005-02-28 Female  (-Inf,5]     9
## 2  2005-02-28 Female    (5,18]     7
## 3  2005-02-28 Female   (18,45]    20
## 4  2005-02-28 Female   (45,60]     5
## 5  2005-02-28 Female (60, Inf]     5
## 6  2005-02-28   Male  (-Inf,5]    11
## 7  2005-02-28   Male    (5,18]     9
## 8  2005-02-28   Male   (18,45]    12
## 9  2005-02-28   Male   (45,60]     3
## 10 2005-02-28   Male (60, Inf]     5
## 11 2005-03-07 Female  (-Inf,5]     9
## 12 2005-03-07 Female    (5,18]     9
## 13 2005-03-07 Female   (18,45]    29
## 14 2005-03-07 Female   (45,60]    15
## 15 2005-03-07 Female (60, Inf]     8
## 16 2005-03-07   Male  (-Inf,5]    18
## 17 2005-03-07   Male    (5,18]    12
## 18 2005-03-07   Male   (18,45]    18
## 19 2005-03-07   Male   (45,60]     9
## 20 2005-03-07   Male (60, Inf]     6
```

Activity

Aggregate the GI data set by gender, ageC, and icd9code (the ones created in the last activity).

When you have completed the activity, compare your results to the solutions.

Basics of ggplot2

Once you have your data in long format, you can use the `ggplot2` package for graphics.

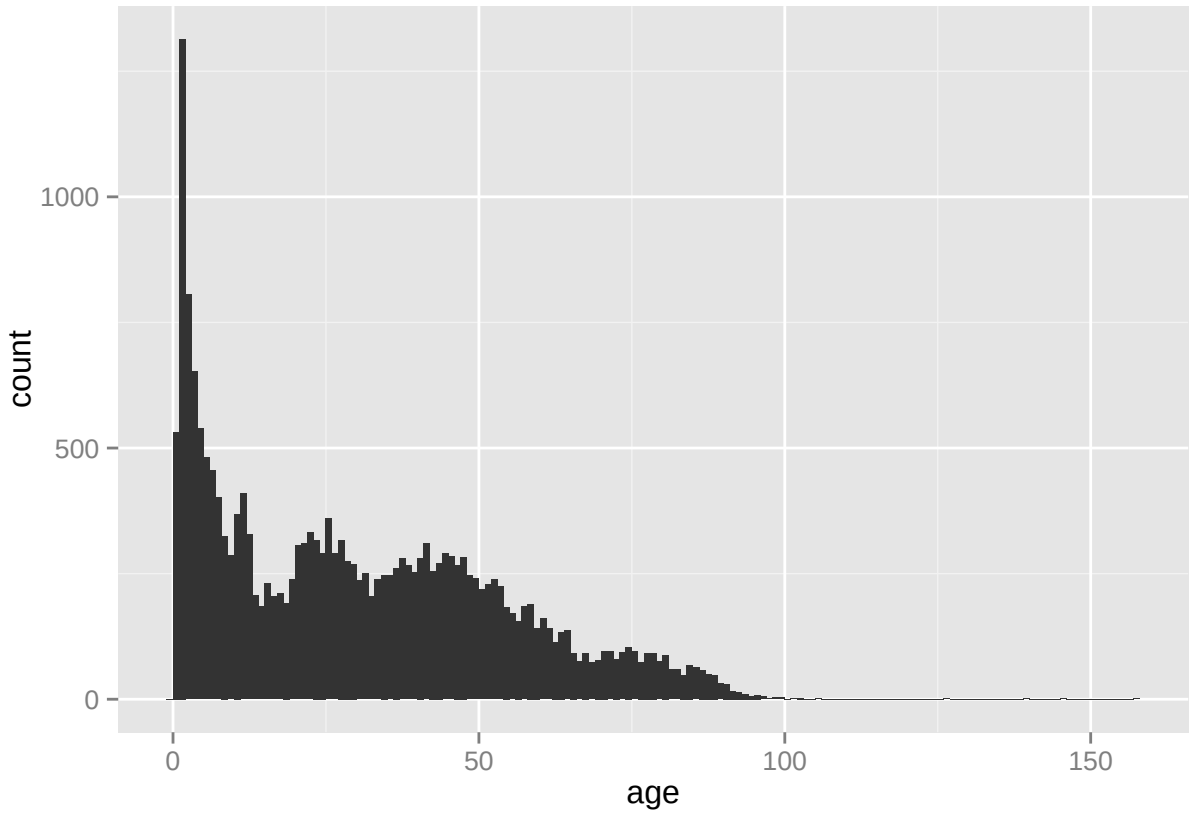
Load the `ggplot2` package

```
library(ggplot2)
```

Histogram

A basic histogram in ggplot

```
ggplot(data = GI, aes(x = age)) + geom_histogram(binwidth=1)
```



For something that looks more similar to the histogram we saw before, you can use

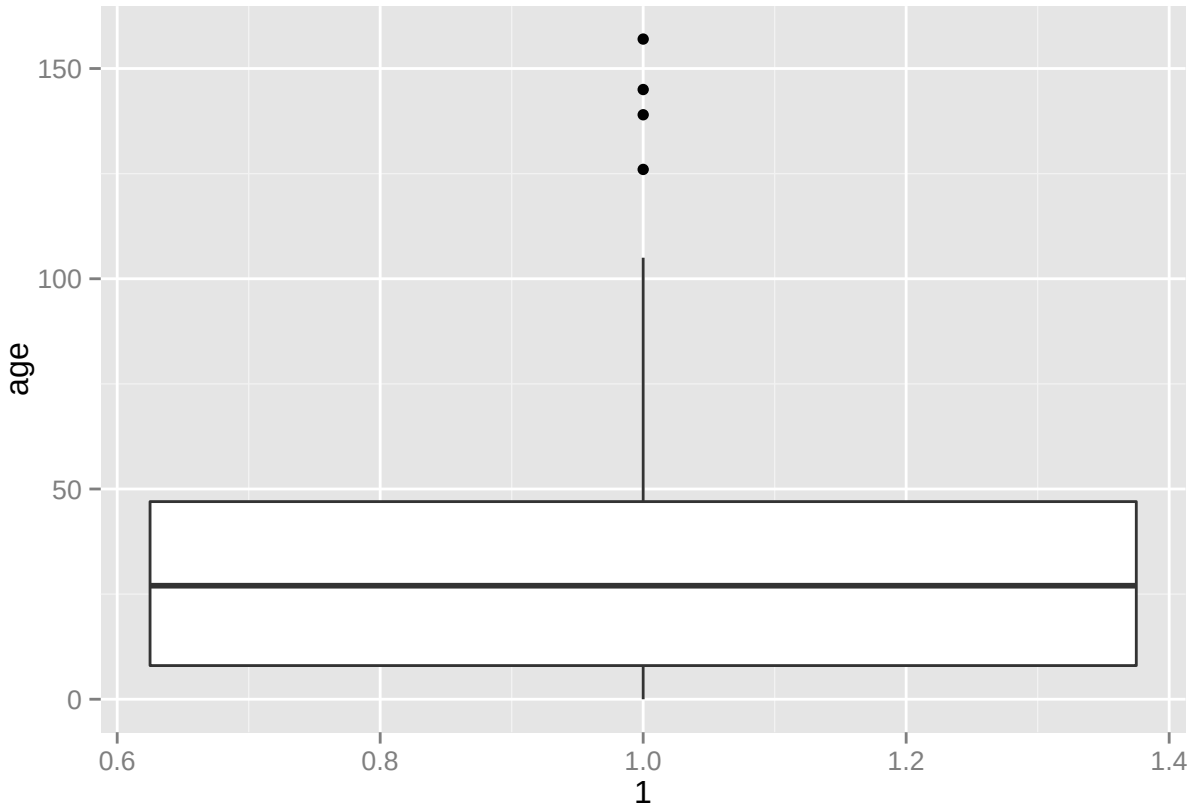
```
qplot(data = GI, x=age, geom="histogram", binwidth=1)
```

‘‘‘

Boxplots

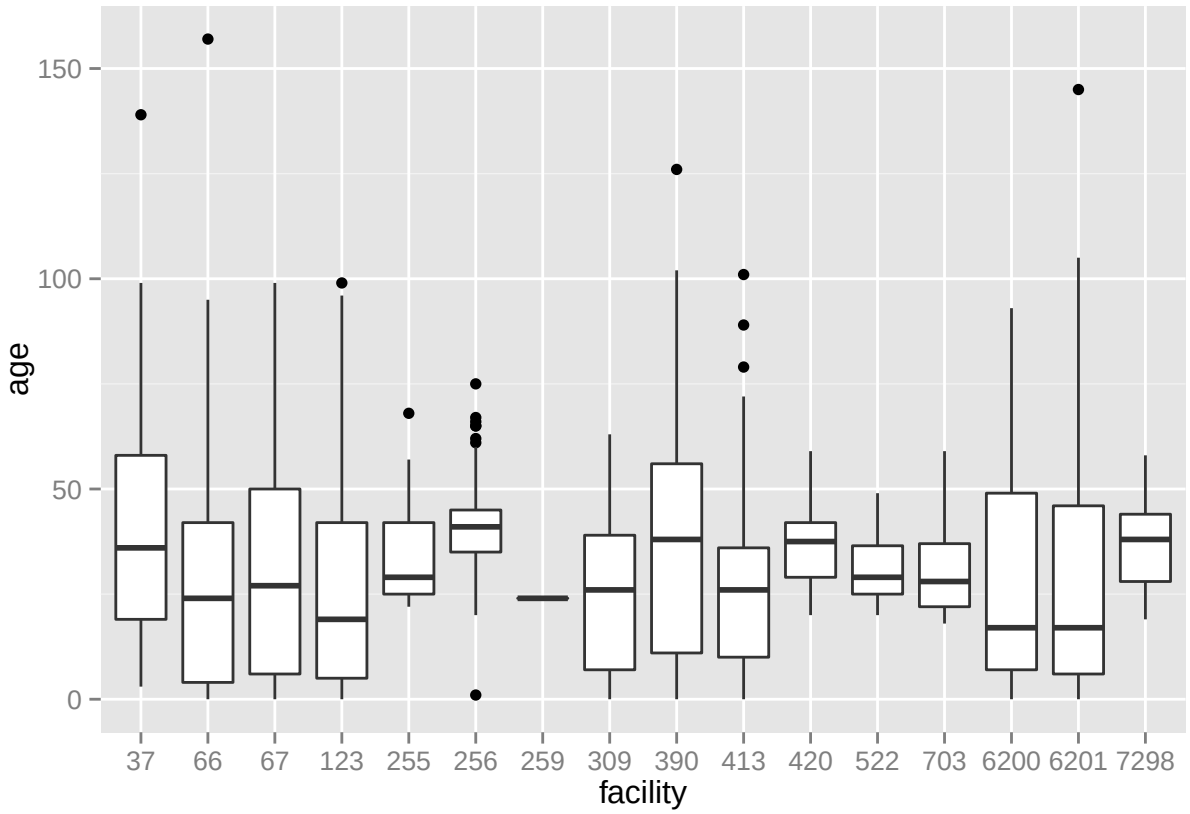
A basic boxplot

```
ggplot(data = GI, aes(x = 1, y = age)) + geom_boxplot()
```



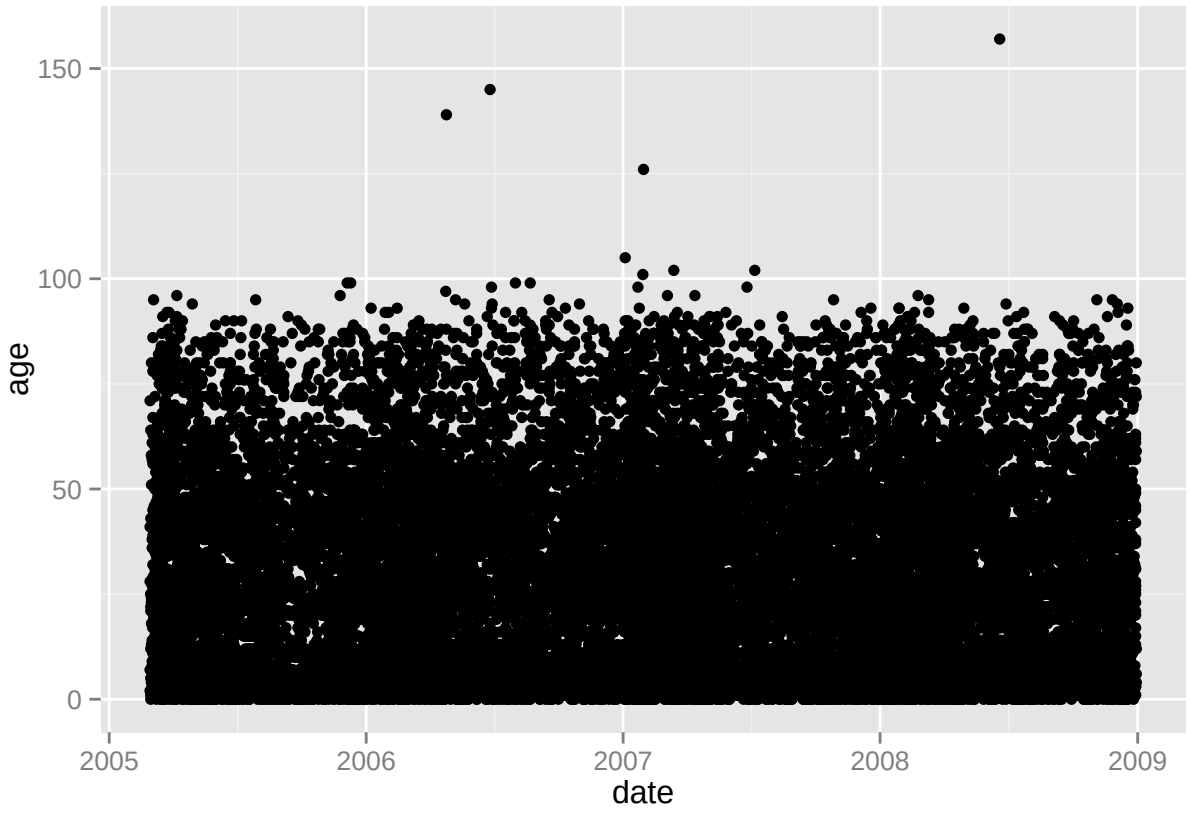
Multiple boxplots

```
ggplot(GI, aes(x = facility, y = age)) + geom_boxplot()
```



Scatterplots

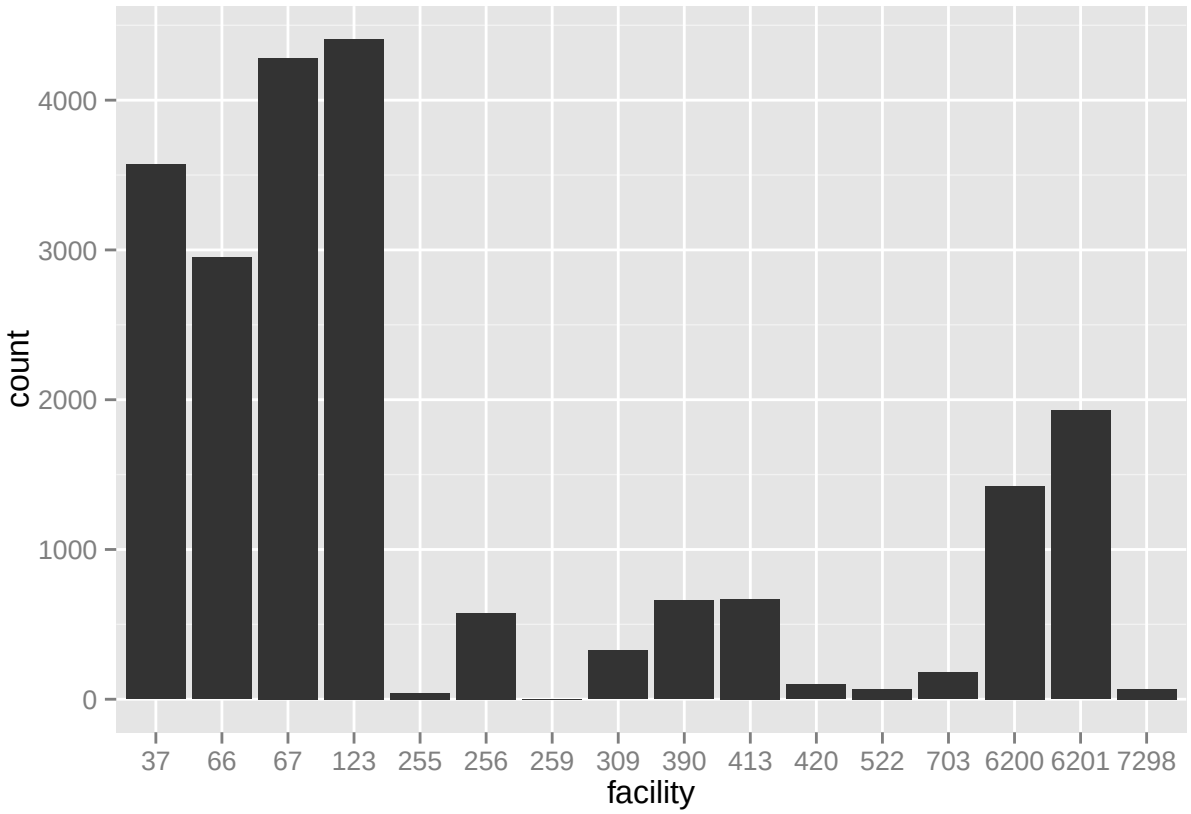
```
ggplot(GI, aes(x=date, y=age)) + geom_point()
```



Bar charts

With ggplot, there is no need to count first.

```
ggplot(GI, aes(x=facility)) + geom_bar()
```



Activity

Construct a histogram and boxplot for age at facility 37 using ggplot2.

Construct a bar chart for the zipcode at facility 37 using ggplot2

When you have completed the activity, compare your results to the solutions.

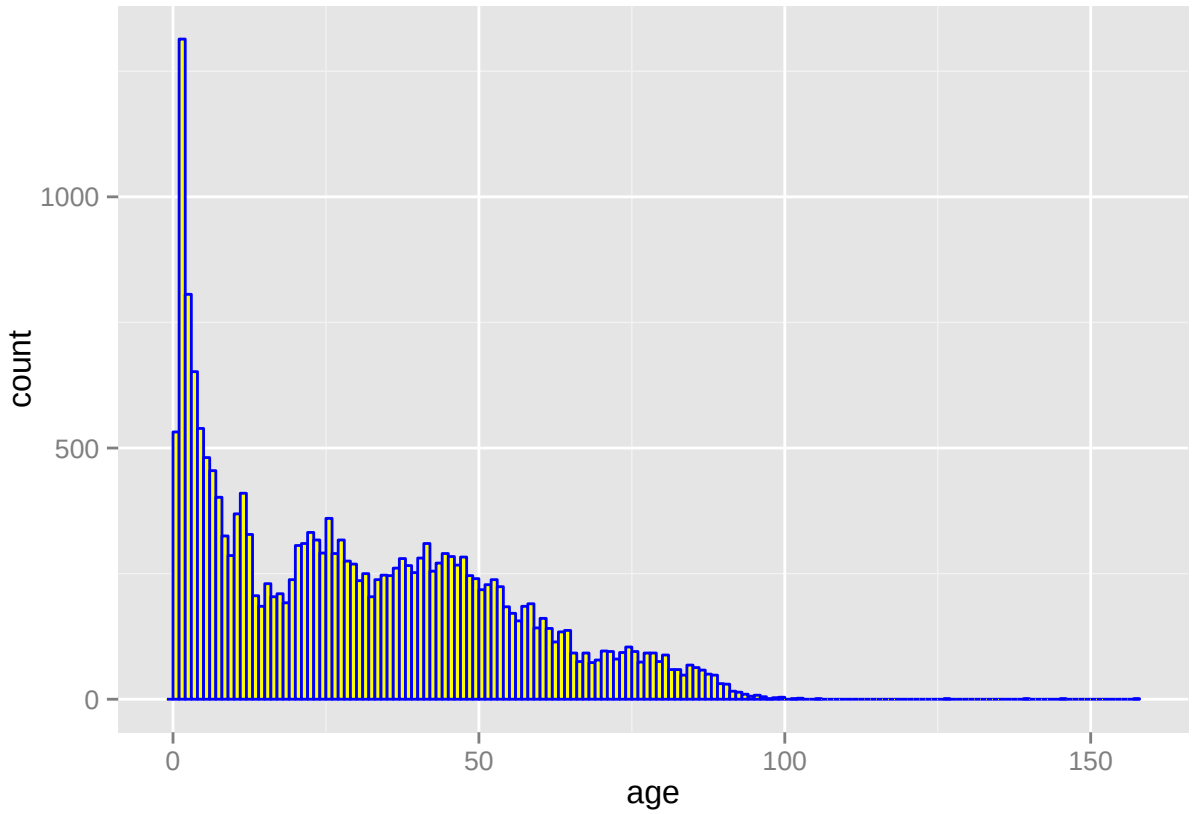
Customizing ggplot2 plots

There are many ways to customize the appearance of ggplot2 plots:

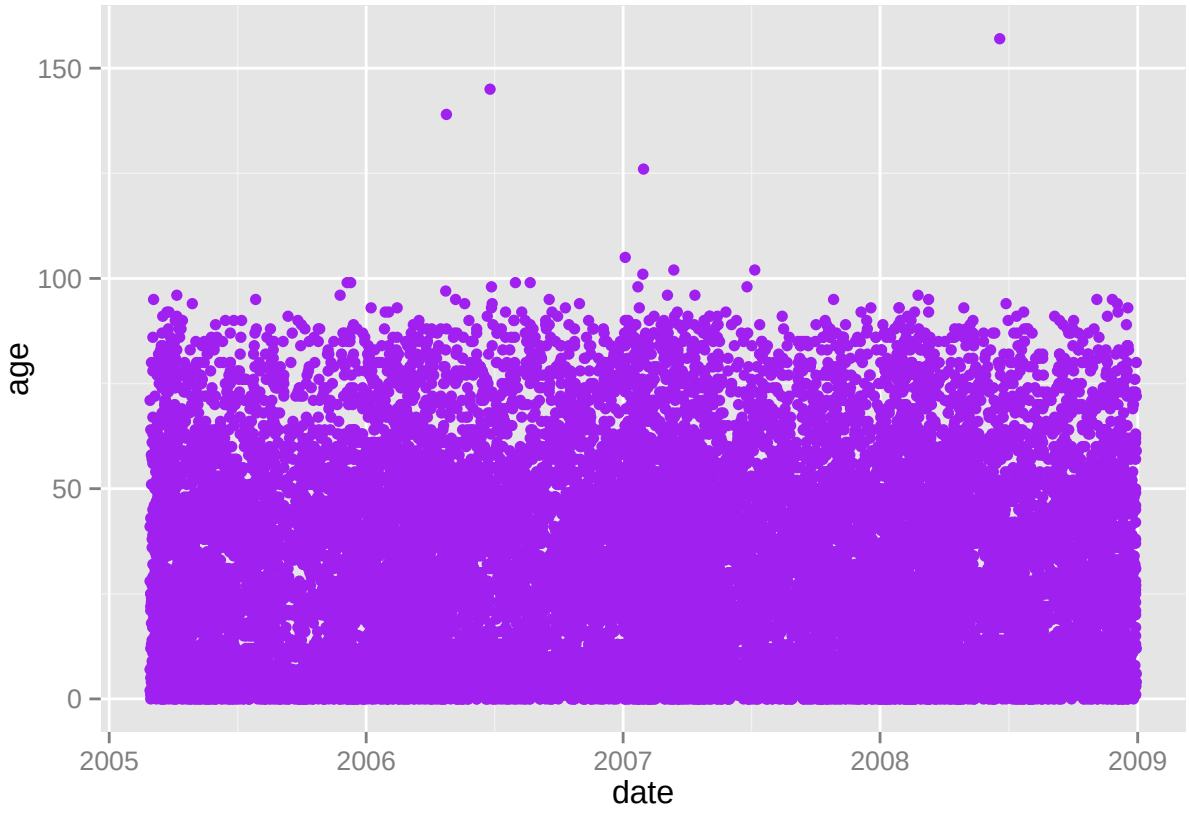
- Colors
- Labels
- Titles
- Characters
- Line types
- Themes

Colors

```
ggplot(GI, aes(x = age)) + geom_histogram(binwidth=1, color='blue',   fill='yellow')
```

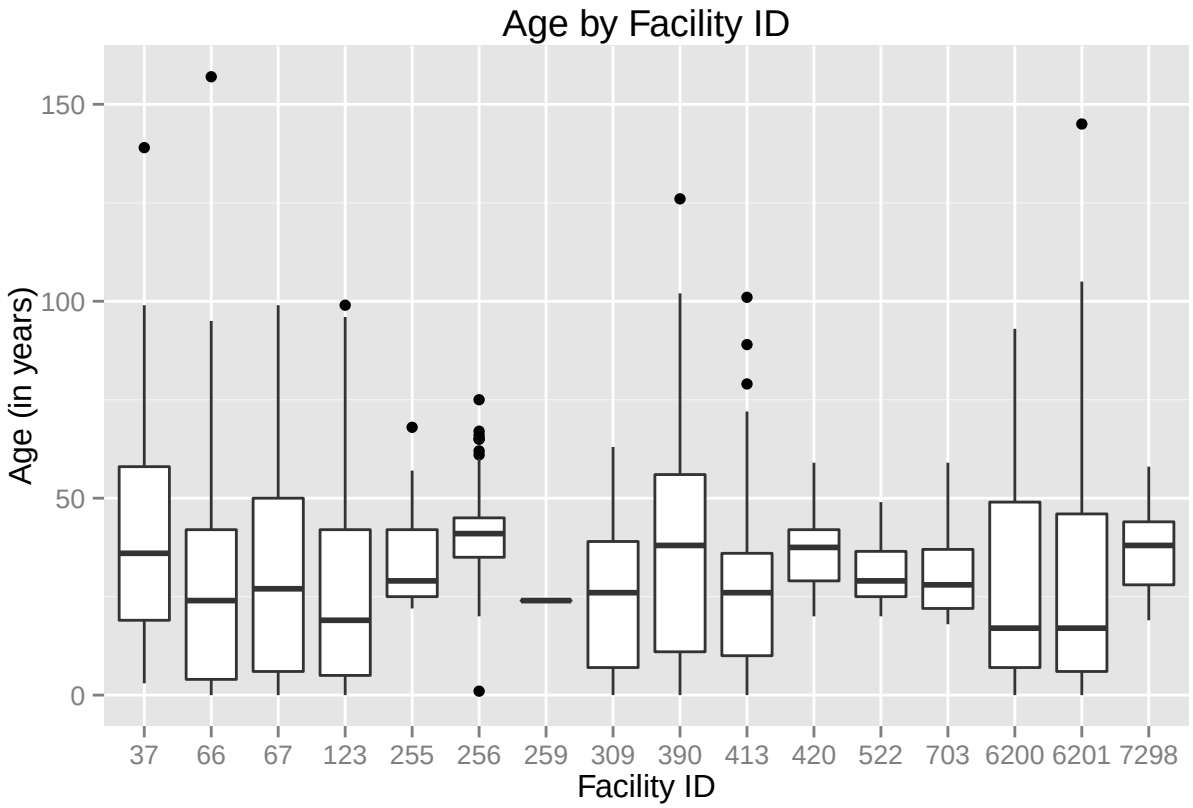



```
ggplot(GI, aes(x=date, y=age)) + geom_point(color='purple')
```



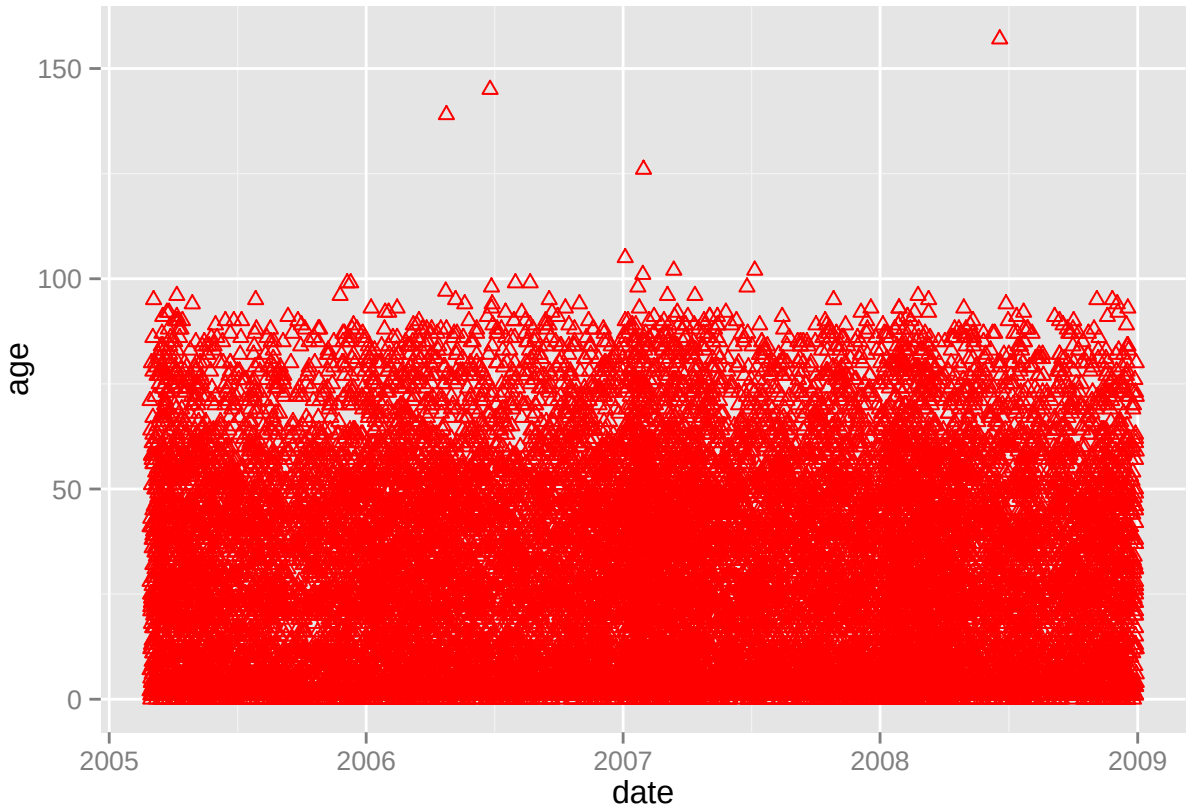
Labels

```
ggplot(GI, aes(x = facility, y = age)) +
  geom_boxplot() +
  labs(x='Facility ID', y='Age (in years)', title='Age by Facility ID')
```



Characters

```
ggplot(GI, aes(x=date, y=age)) + geom_point(shape=2, color='red')
```



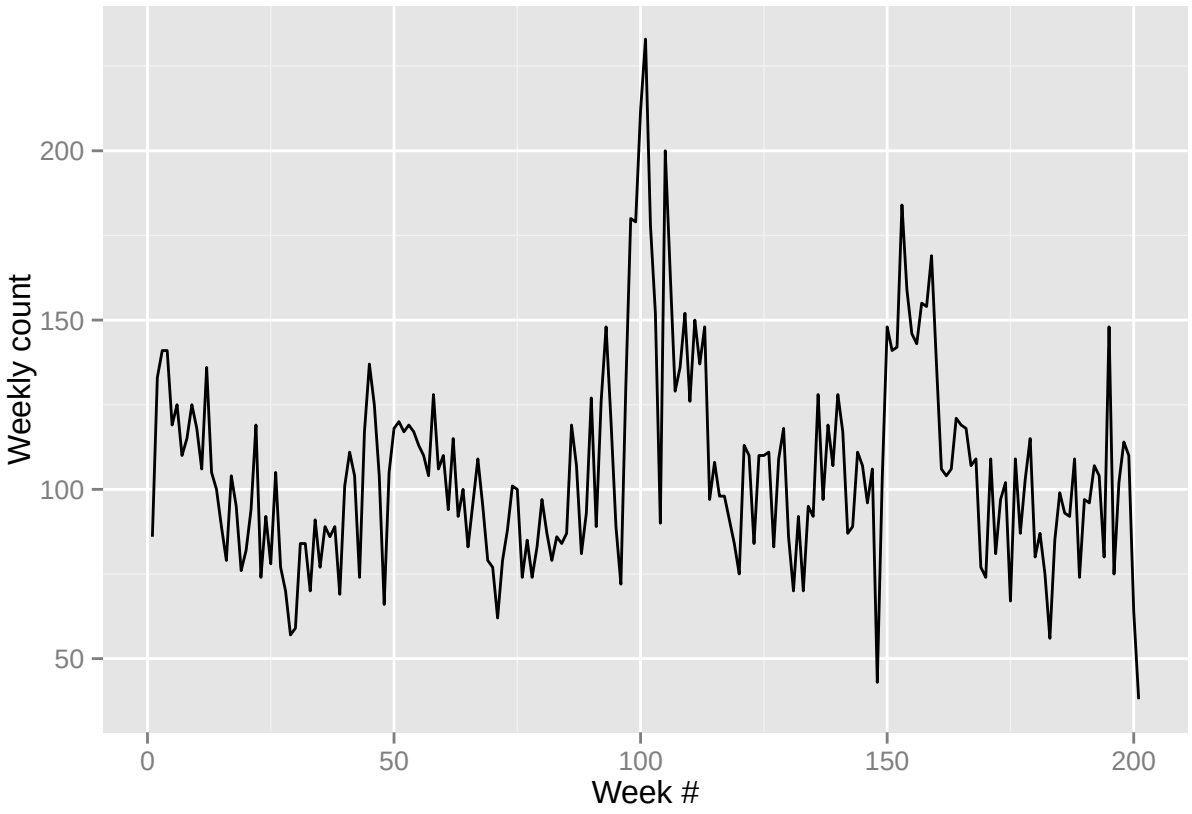
ggplot2 uses the same shape codes as base graphics, see

```
?points
```

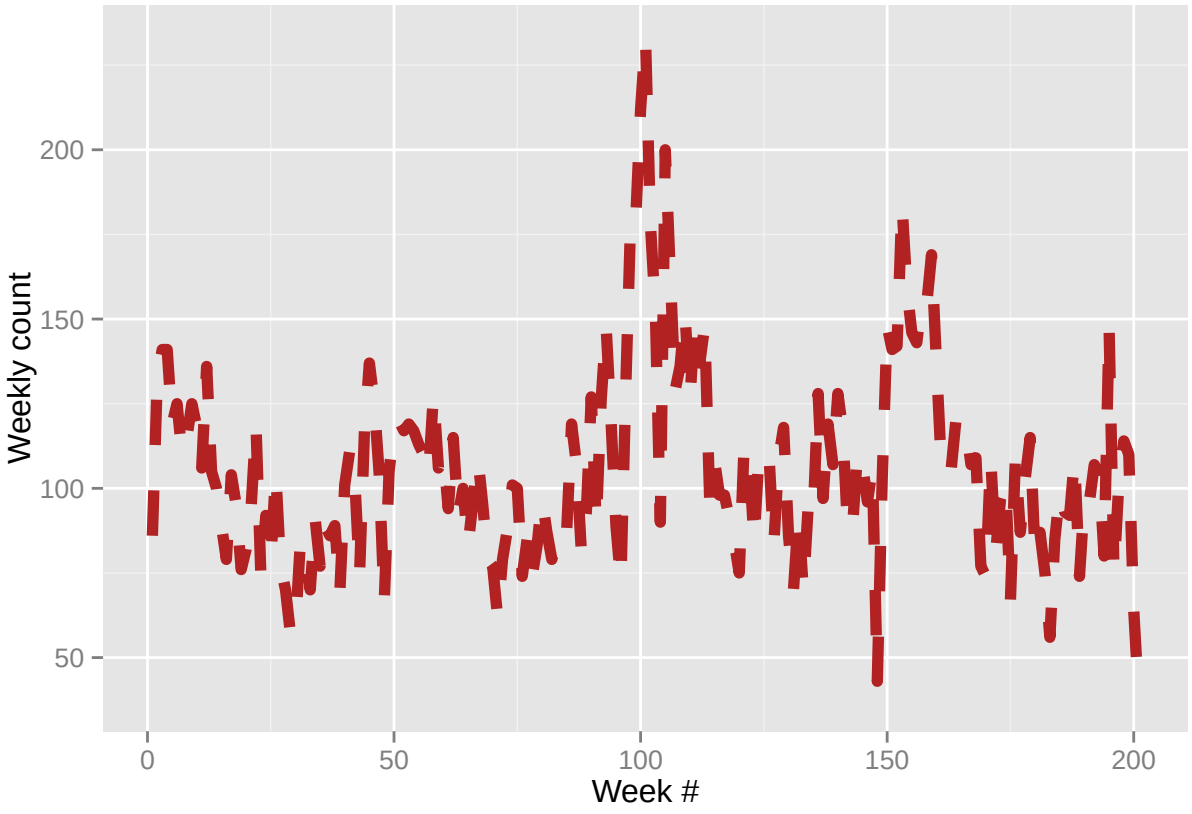
Line types

Here I am also using a trick of setting up part of the plot and assigning it to the object g. Then you can add elements to the plot and if you don't assign it, the plot will be shown.

```
g = ggplot(ddply(GI, .(week), summarize, count=length(id)),
       aes(x=as.numeric(week), y=count)) +
  labs(x='Week #', y='Weekly count')

g + geom_line()
```

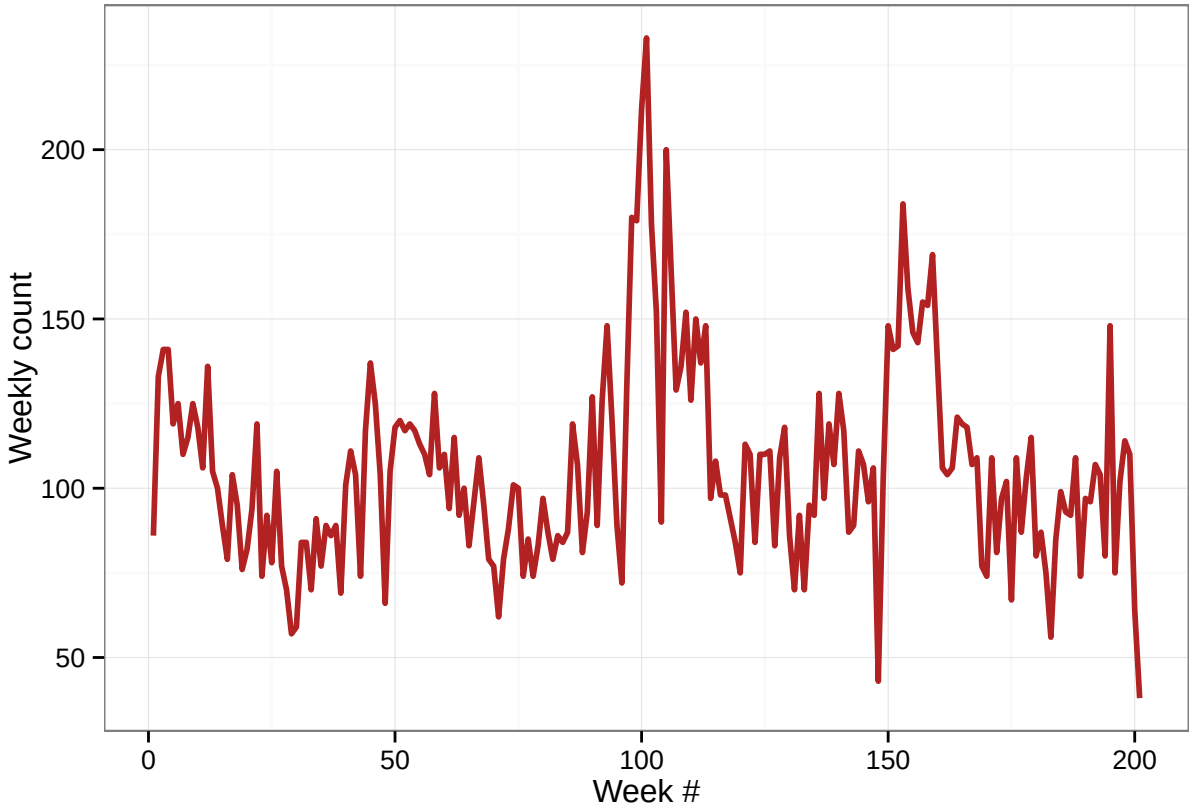



```
g + geom_line(size=2, color='firebrick', linetype=2)
```



Themes

```
g = g+geom_line(size=1, color='firebrick')
g + theme_bw()
```



For other themes, see

```
?theme
?theme_bw
```

Getting help on ggplot2

Although the general R help can still be used, e.g.

```
?ggplot
?geom_point
```

It is much more helpful to google for an answer

```
geom_point
ggplot2 line colors
```

The top hits will all have the code along with what the code produces.

Helpful sites

These sites all provide code. The first two also provide the plots that are produced.

- http://docs.ggplot2.org/current/
- http://www.cookbook-r.com/Graphs/
- http://stackoverflow.com/questions/tagged/ggplot2

Activity

Play around with ggplot2 to see what kind of plots you can make.

Data Visualization I (ggplot2)

Jarad Niemi

2014-12-04

Contents

Close down R, say 'No' to saving the workspace.

Workflow

The typical workflow in R is

1. Start R
2. Choose your working directory
3. Open a script
4. Read in data
5. Check data (interactively)
6. Do analysis

If you are using RStudio, I highly recommend you look into setting up projects.

Choose your working directory

Start R and then choose your working directory.

```
setwd(choose.dir())
```

Start the workshop

Now start the workshop to get your data files and scripts set up. Typically you would choose your working directory to be where you actually have your data and scripts.

```
library(ISDSWorkshop)
workshop(launch_index  = FALSE)
```

Open the script

Open the `advanced_graphics.R` script.

Notice that the top of the script has all of the packages that you will use.

```
##########################################
# Pretend this is the top of the script #
##########################################
library(plyr)
library(ggplot2)
library(gridExtra)
```

```
## Loading required package: grid
```

Read in the data

Typically, you will have a script that reads in the data, converts variables, and then performs statistical analyses.

This time we will use the `mutate()` function from the plyr package. This function allows you to create new columns in a `data.frame`.

```
# Read in csv files
GI = read.csv('GI.csv')
icd9df = read.csv("icd9.csv")

# Mutate data.frame
GI = mutate(GI,
            date      = as.Date(date),
            weekC     = cut(date, breaks="weeks"),
            week      = as.numeric(weekC),
            facility  = as.factor(facility),
            icd9class = factor(cut(icd9,
                           breaks = icd9df$code_cutpoint,
                           labels = icd9df$classification[-nrow(icd9df)],
                           right  = TRUE)),
            ageC      = cut(age,
                           breaks = c(-Inf, 5, 18, 45 ,60, Inf)),
            zip3      = trunc(zipcode/100))
```

Reading other scripts

If the script to read in the data is extensive, it may be better to separate your scripts into two different files: one for reading in the data and converting variables and another (or more) to perform statistical analyses. Then, in the second file, i.e. the one that does an analysis, you can `source()` the one that just reads in the data, e.g.

```
source("read_data.R")
```

Check the data

At this point, I typically check the data to make sure it is what I think it is. I usually do this in interactive mode, i.e. not in a script.

```
dim(GI)
```

```
## [1] 21244    14
```

```
str(GI)
```

```
## 'data.frame':    21244 obs. of  14 variables:
##  $ id             : int  1001301988 1001829757 1001581758 1001950471 1001076304 1001087075 100153694
##  $ date           : Date, format: "2005-02-28" "2005-02-28" ...
##  $ facility       : Factor w/ 16 levels "37","66","67",..: 3 3 4 4 8 2 15 3 2 2 ...
##  $ icd9           : num  787 559 788 788 559 ...
```

```
##  $ age            : int  7 41 2 71 28 43 12 1 25 64 ...
##  $ zipcode        : int  21075 20721 22152 22060 21702 20762 22192 20121 20772 20602 ...
##  $ chief_complaint: Factor w/ 2936 levels " /v"," /V/D",..: 315 2600 1100 15 1741 2238 640 2480 1833
##  $ syndrome       : Factor w/ 1 level "GI": 1 1 1 1 1 1 1 1 1 1 ...
##  $ gender         : Factor w/ 2 levels "Female","Male": 2 1 2 2 1 1 2 2 2 2 ...
##  $ weekC          : Factor w/ 201 levels "2005-02-28","2005-03-07",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ week           : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ icd9class      : Factor w/ 4 levels "infectious and parasitic disease",..: 3 2 3 3 2 3 2 3 3 3 ..
##  $ ageC           : Factor w/ 5 levels "(-Inf,5]","(5,18]",..: 2 3 1 5 3 3 2 1 3 5 ...
##  $ zip3           : num  210 207 221 220 217 207 221 201 207 206 ...
```

```
summary(GI)
```

```
##        id                  date                facility         icd9
##  Min.   :1.001e+09   Min.   :2005-02-28   123    :4408   Min.   :    3.0
##  1st Qu.:1.001e+09   1st Qu.:2006-03-07   67     :4281   1st Qu.:  558.9
##  Median :1.001e+09   Median :2007-02-14   37     :3571   Median :  787.0
##  Mean   :1.001e+09   Mean   :2007-02-05   66     :2950   Mean   : 1043.2
##  3rd Qu.:1.002e+09   3rd Qu.:2008-01-17   6201   :1928   3rd Qu.:  787.3
##  Max.   :1.002e+09   Max.   :2008-12-30   6200   :1423   Max.   :78791.0
##                                           (Other):2683
##       age             zipcode           chief_complaint  syndrome
##  Min.   :  0.00   Min.   :20001   Abd Pain      : 1390   GI:21244
##  1st Qu.:  8.00   1st Qu.:20747   ABD PAIN      : 1074
##  Median : 27.00   Median :21740   Vomiting      :  661
##  Mean   : 29.98   Mean   :21420   VOMITING      :  563
##  3rd Qu.: 47.00   3rd Qu.:22182   ABDOMINAL PAIN:  452
##  Max.   :157.00   Max.   :22556   vomiting      :  423
##                                   (Other)       :16681
##     gender            weekC            week
##  Female:10653   2007-01-29:  233   Min.   :  1.0
##  Male  :10591   2007-01-22:  212   1st Qu.: 54.0
##                 2007-02-26:  200   Median :103.0
##                 2008-01-28:  184   Mean   :101.7
##                 2007-01-08:  180   3rd Qu.:151.0
##                 2007-01-15:  179   Max.   :201.0
##                 (Other)   :20056
##                                        icd9class            ageC
##  infectious and parasitic disease           : 1611   (-Inf,5] :4324
##  diseases of the digestive system           : 7242   (5,18]   :3802
##  symptoms, signs, and ill-defined conditions:12229   (18,45]  :7476
##  injury and poisoning                       :  162   (45,60]  :3133
##                                                      (60, Inf]:2509
##
##
##       zip3
##  Min.   :200.0
##  1st Qu.:207.0
##  Median :217.0
##  Mean   :213.8
##  3rd Qu.:221.0
##  Max.   :225.0
##
```

Activity

Create a new variable in the GI data set called `weekD` that is a `Date` object with each observation having the Monday of the week for that observation. Check that it is actually a date using the `str()` function.

When you have completed the activity, compare your results to the solutions.

Graph workflow

Now to construct a graph, the workflow is

1. construct an appropriate data set and
2. construct the graph.

Construct the data set

Suppose we want to plot the number of weekly GI cases by facility. We need to aggregate the data by week and facility.

```
GI_wf = ddply(GI, .(week, facility), summarize, count = length(id))
```

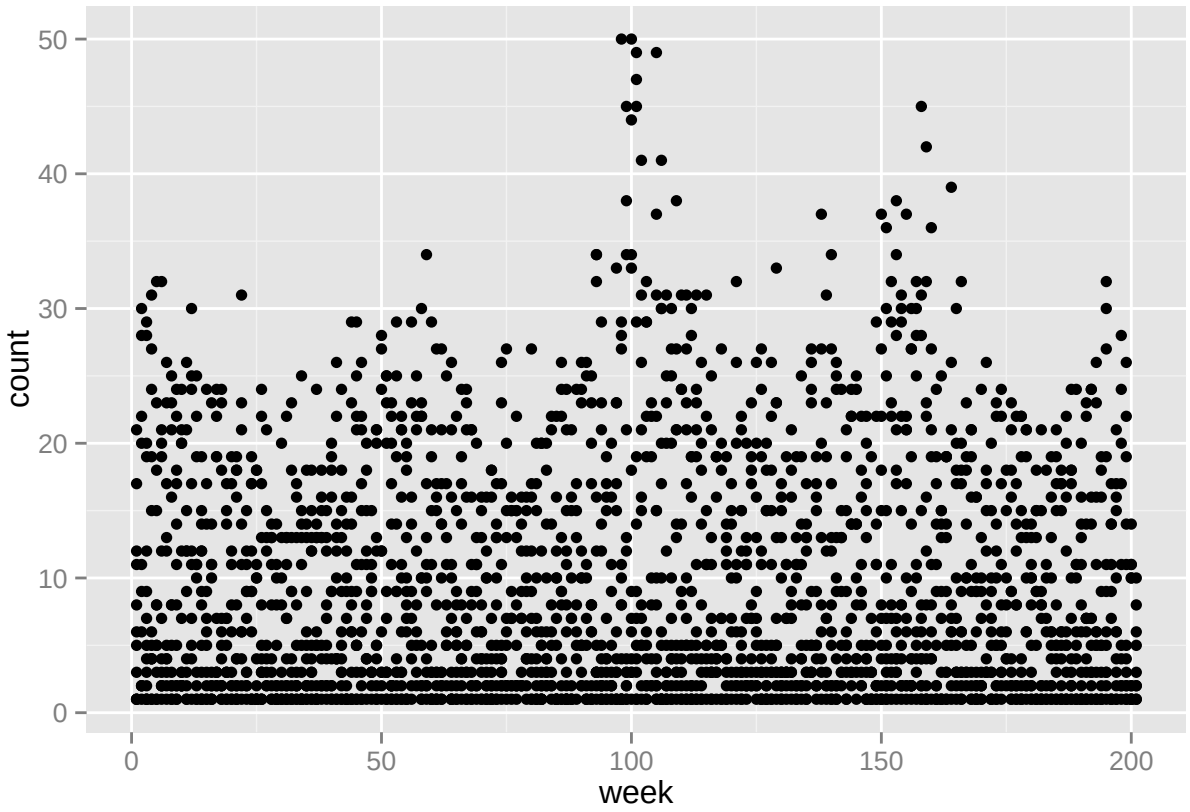
In interactive mode, you should verify that this data set is correct, e.g.

```
nrow(GI_wf) # Should have number of weeks times number of facilities rows
ncol(GI_wf) # Should have 3 columns: week, facility, count
dim(GI_wf)
head(GI_wf)
tail(GI_wf)
summary(GI_wf)
summary(GI_wf$facility)
```

Construct the graph

Now, we would like week on the x-axis and count on the y-axis.

```
ggplot(GI_wf, aes(x=week, y=count)) + geom_point()
```



But, clearly we need to distinguish the facilities.

Try colors and shapes to distinguish facilities

Colors:

```
ggplot(GI_wf, aes(x=week, y=count, color=facility)) + geom_point()
```

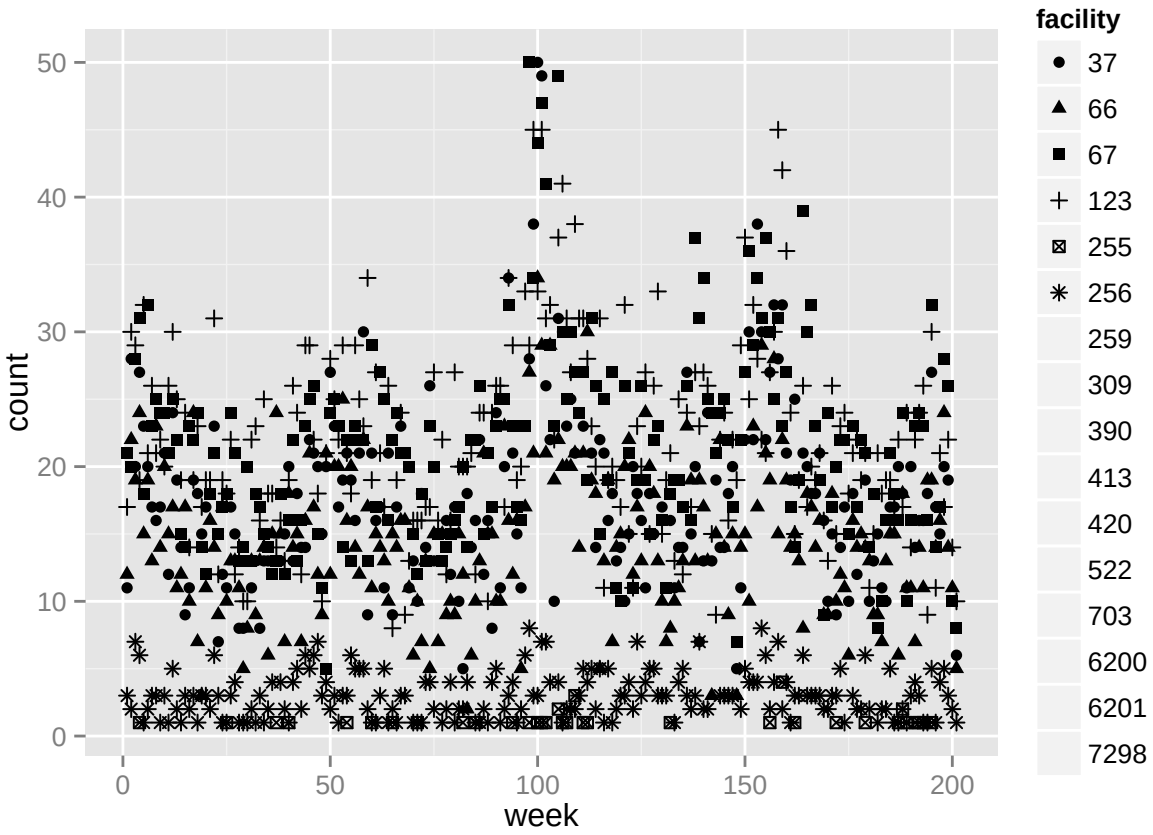
Shapes:

```
ggplot(GI_wf, aes(x=week, y=count, shape=facility)) + geom_point()
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have
## 16. Consider specifying shapes manually. if you must have them.
```

```
## Warning: Removed 1229 rows containing missing values (geom_point).
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have
## 16. Consider specifying shapes manually. if you must have them.
```



ggplot2 only plots 6 different shapes and provides a warning.

Activity - weekly GI counts by age category

Construct a data set and then a plot to look at weekly GI cases by age category.

When you have completed the activity, compare your results to the solutions.

Faceting

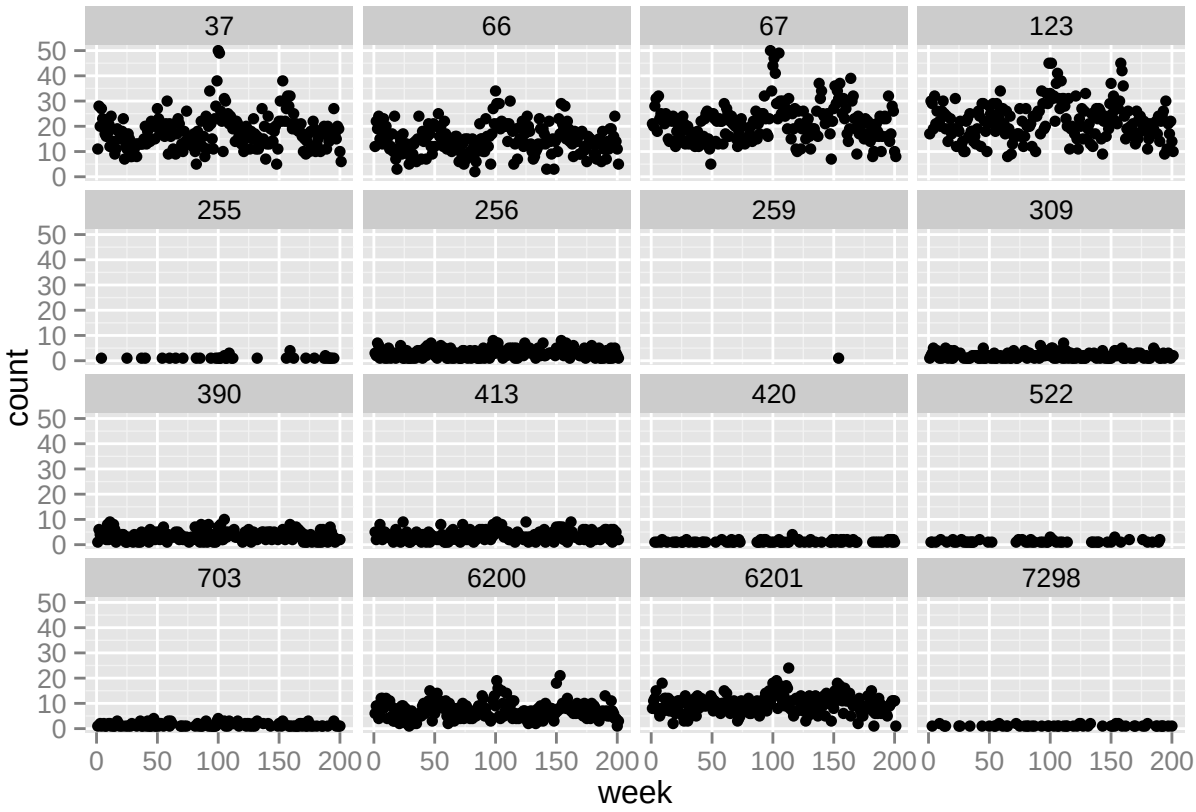
Faceting is a technique to view many small graphs on a single page.

In ggplot, there are two different ways to construct facets: `facet_wrap()` and `facet_grid()`. The former performs the faceting for a single variable and constructs the matrix of plots automatically. The latter performs faceting for two variables putting one vertically and the other horizontally.

Faceting on a single variable

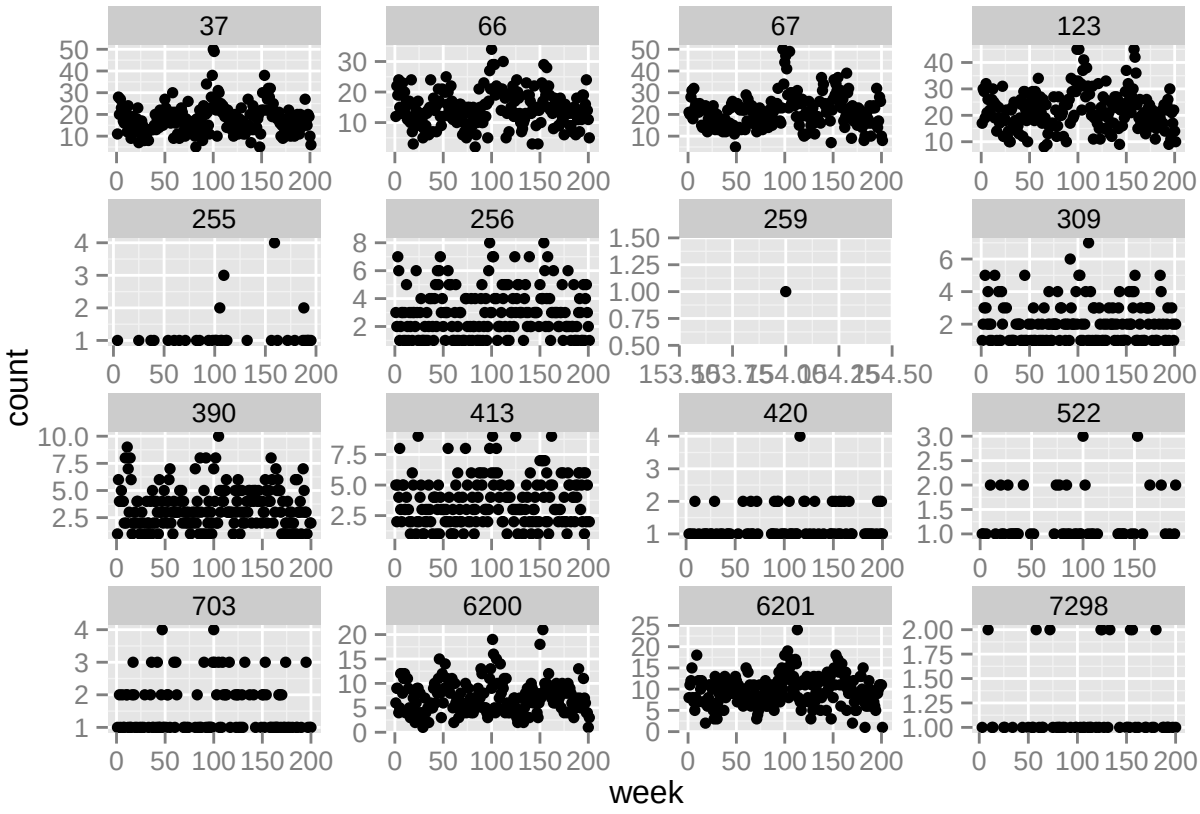
When there are many levels of a single variable, `facet_wrap()` can often allow comparisons across that variable.

```
ggplot(GI_wf, aes(x=week, y=count)) + geom_point() + facet_wrap(~facility)
```



By default, faceting forces all axes to be exactly the same. If you want the axis to automatically scale for each facet use `scales="free"`.

```
ggplot(GI_wf, aes(x=week, y=count)) + geom_point() + facet_wrap(~facility, scales="free")
```



Faceting on two variables

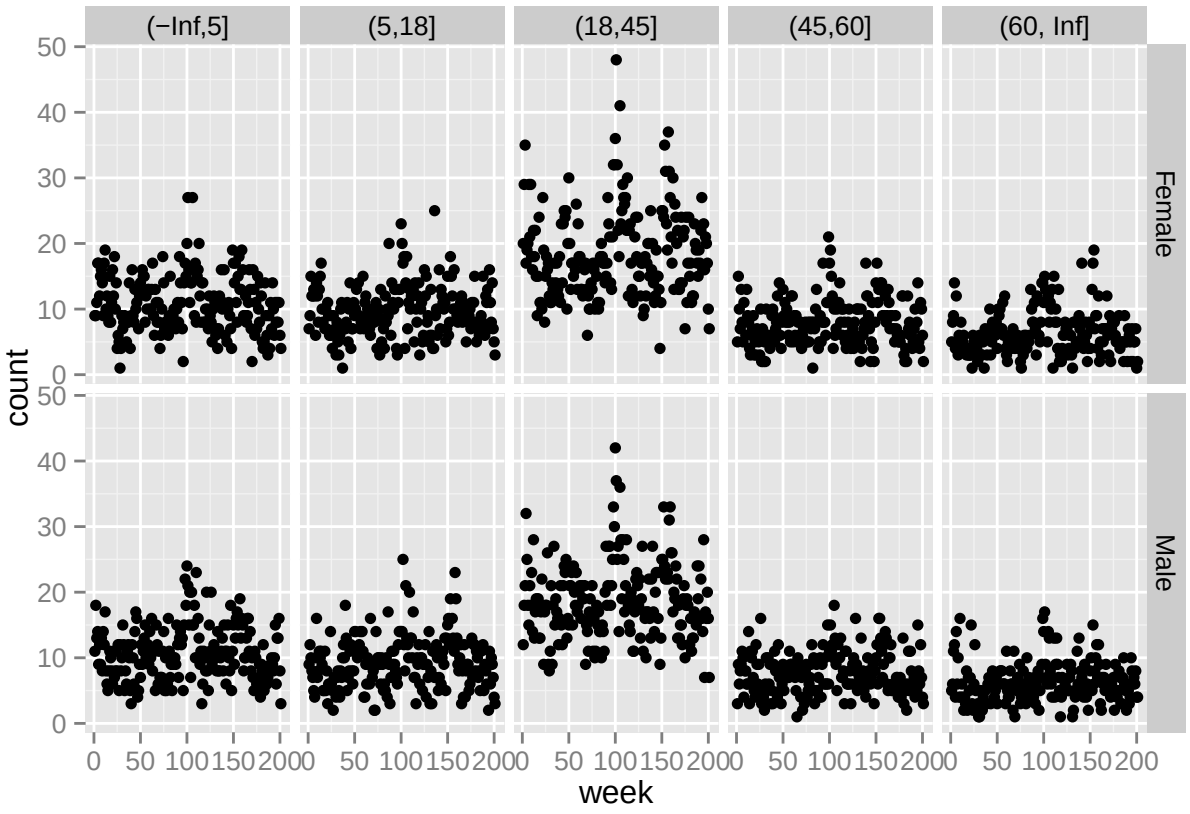
When there are two variables (with not very many levels in each), `facet_grid()` can allow relevant comparisons.

Suppose we are interested in weekly GI counts by gender and age category. First, construct the data set

```
GI_sa = ddply(GI, .(week, gender, ageC), summarize, count = length(id))
```

Next, construct the plot using `facet_grid(row~column)` syntax.

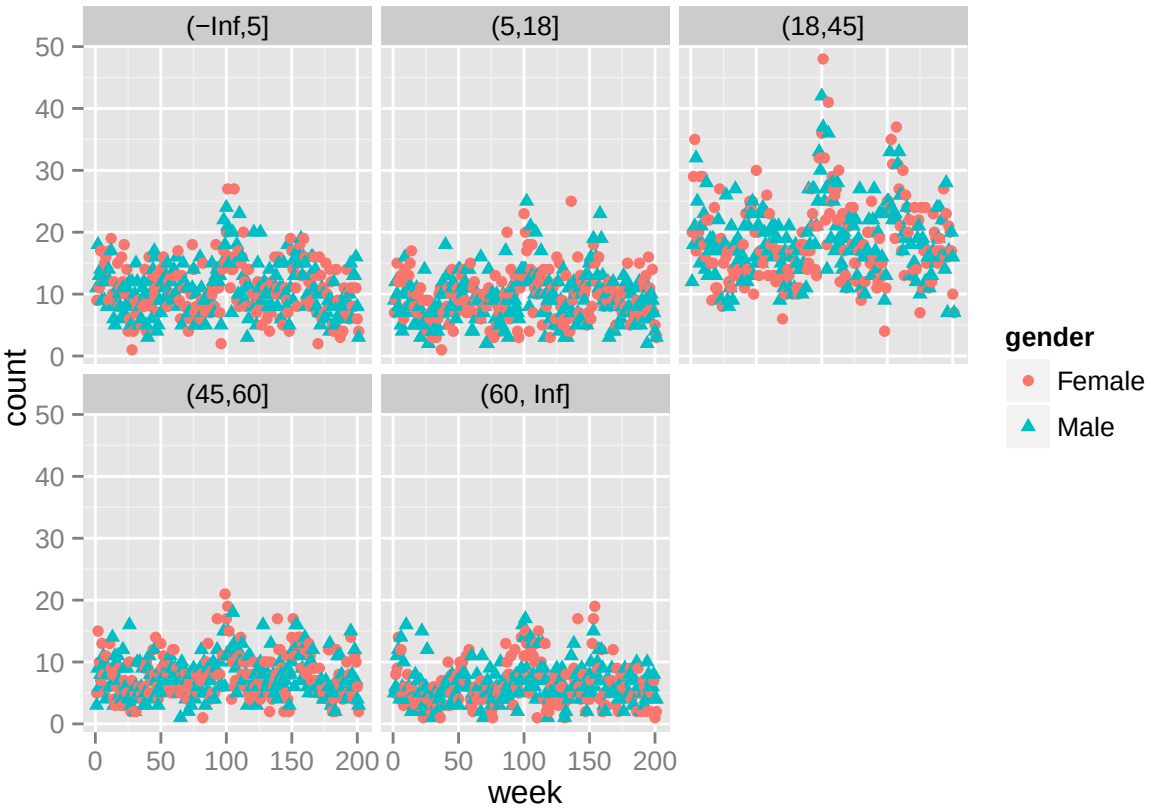
```
ggplot(GI_sa, aes(x=week, y=count)) + geom_point() + facet_grid(gender~ageC)
```



Using faceting and colors/shapes

In addition, we could use faceting together with colors/shapes

```
ggplot(GI_sa, aes(x=week, y=count, shape=gender, color=gender)) + geom_point() + facet_wrap(~ageC)
```



Activity - weekly GI counts by zip3 and ageC

Construct a plot of weekly GI counts by zip3 and ageC.

When you have completed the activity, compare your results to the solutions.

Subsetting

Often our data set is large and we would like to subset the data to focus on a particular group or time frame. You may want to subset by a

- categorical,
- continuous (numeric), or
- `Date` variable.

Subsetting by a categorical variable

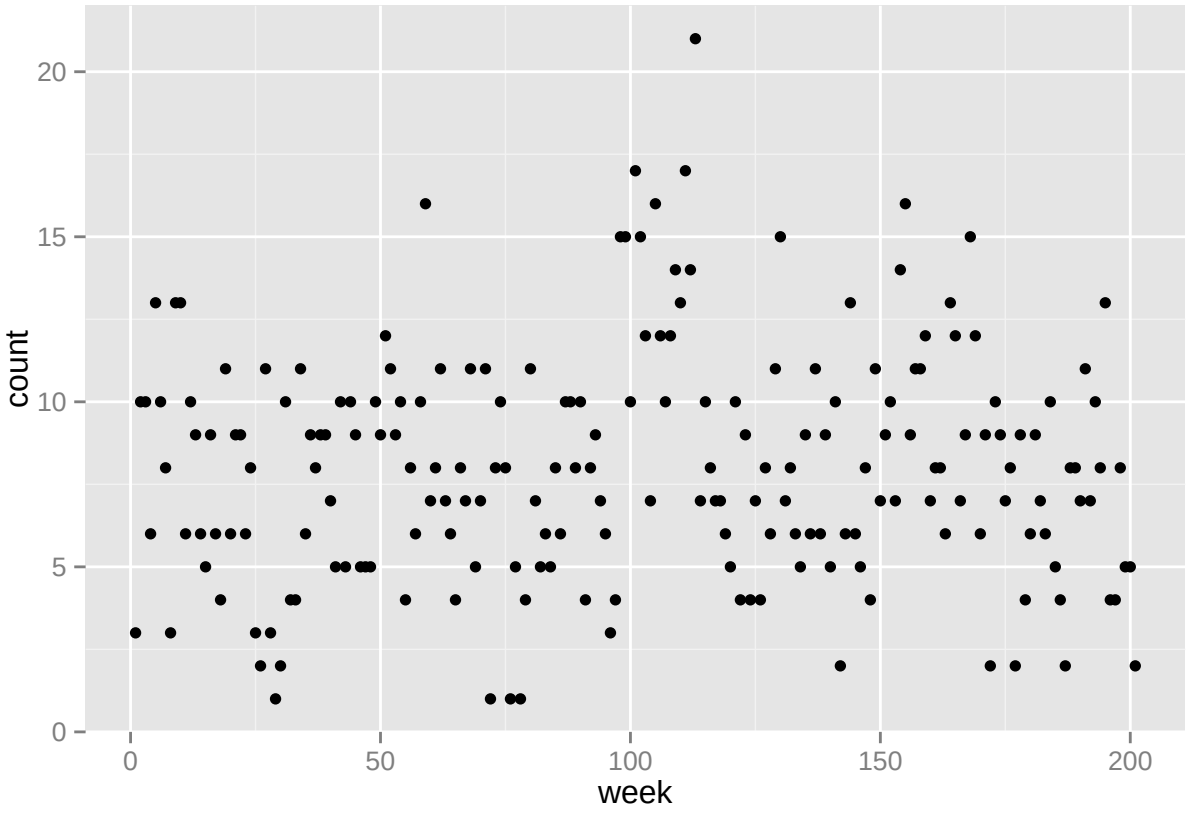
For a categorical variable, you want to subset by

- a single category
- a set of categories
- not in a single category
- not in a set of categories

Subsetting by a single category

Suppose we are only interested in counts specifically for infectious and parasitic diseases (IPD).

```
IPD = subset(GI, icd9class == "infectious and parasitic disease")
IPD_w = ddply(IPD, .(week), summarize, count=length(id))
ggplot(IPD_w, aes(x=week, y=count)) + geom_point()
```



Subsetting by a set of categories

Suppose we are interested in looking at both IPD and ill-defined conditions

```
conditions = levels(GI$icd9class)
conditions
```

```
## [1] "infectious and parasitic disease"
## [2] "diseases of the digestive system"
## [3] "symptoms, signs, and ill-defined conditions"
## [4] "injury and poisoning"
```

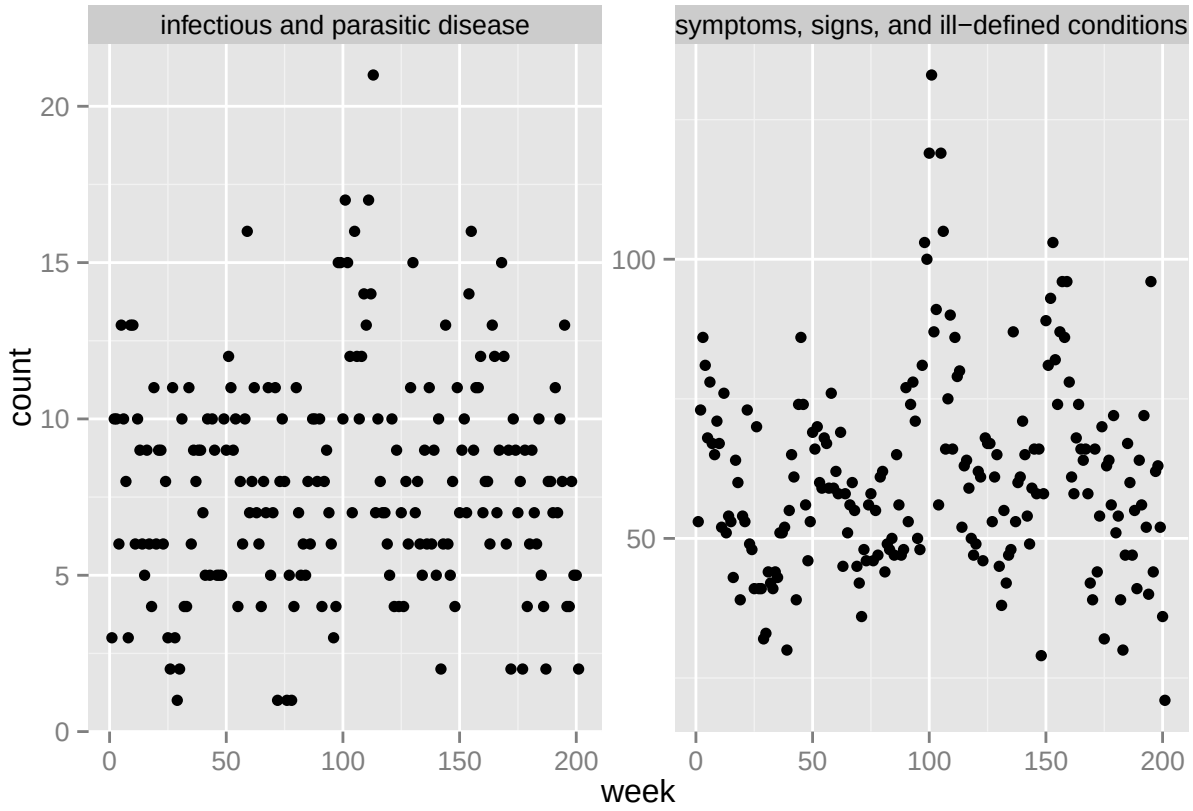
Say we want levels 1 and 3.

```
IPDp = subset(GI, icd9class %in% levels(icd9class)[c(1,3)])
summary(IPDp$icd9class)
```

```
##            infectious and parasitic disease
##                                        1611
##            diseases of the digestive system
##                                           0
## symptoms, signs, and ill-defined conditions
##                                       12229
##                         injury and poisoning
##                                           0
```

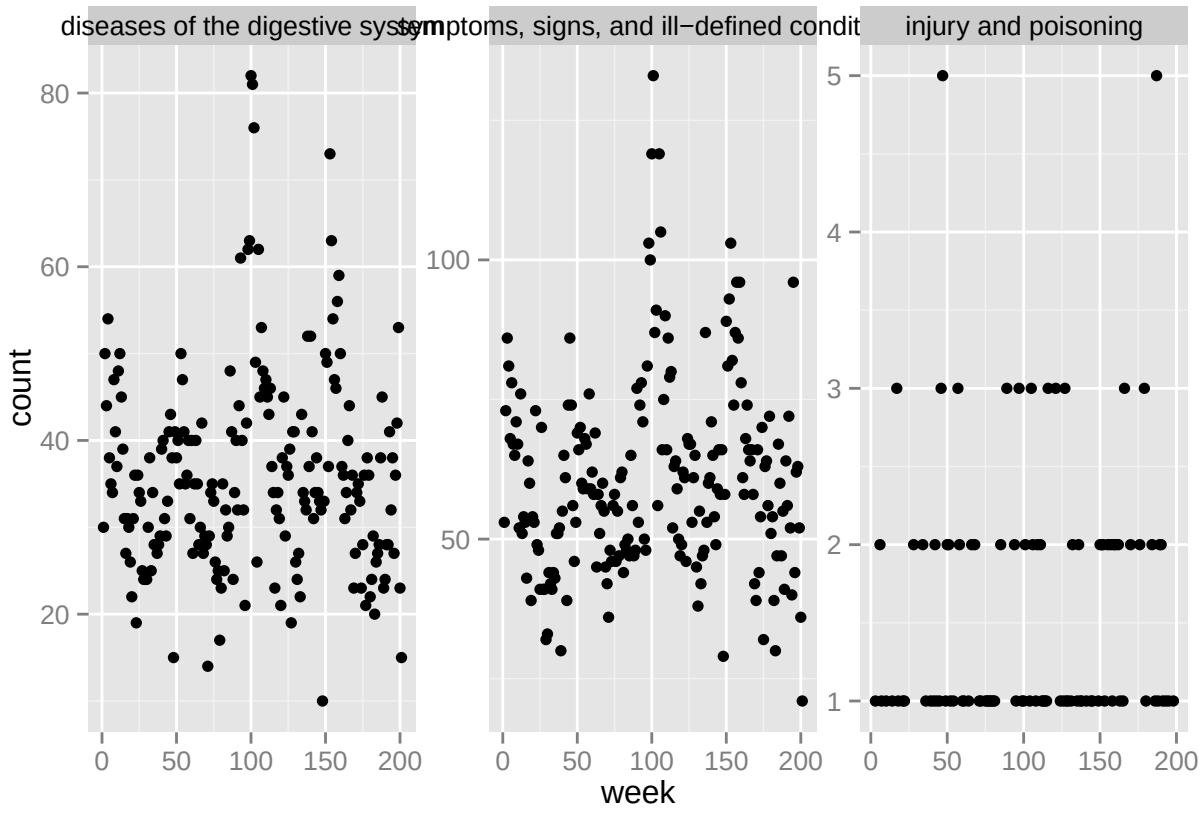
Now construct the graph

```
IPDp_w = ddply(IPDp, .(week, icd9class), summarize, count = length(id))
ggplot(IPDp_w, aes(x=week, y=count)) + geom_point() + facet_wrap(~icd9class, scales="free")
```



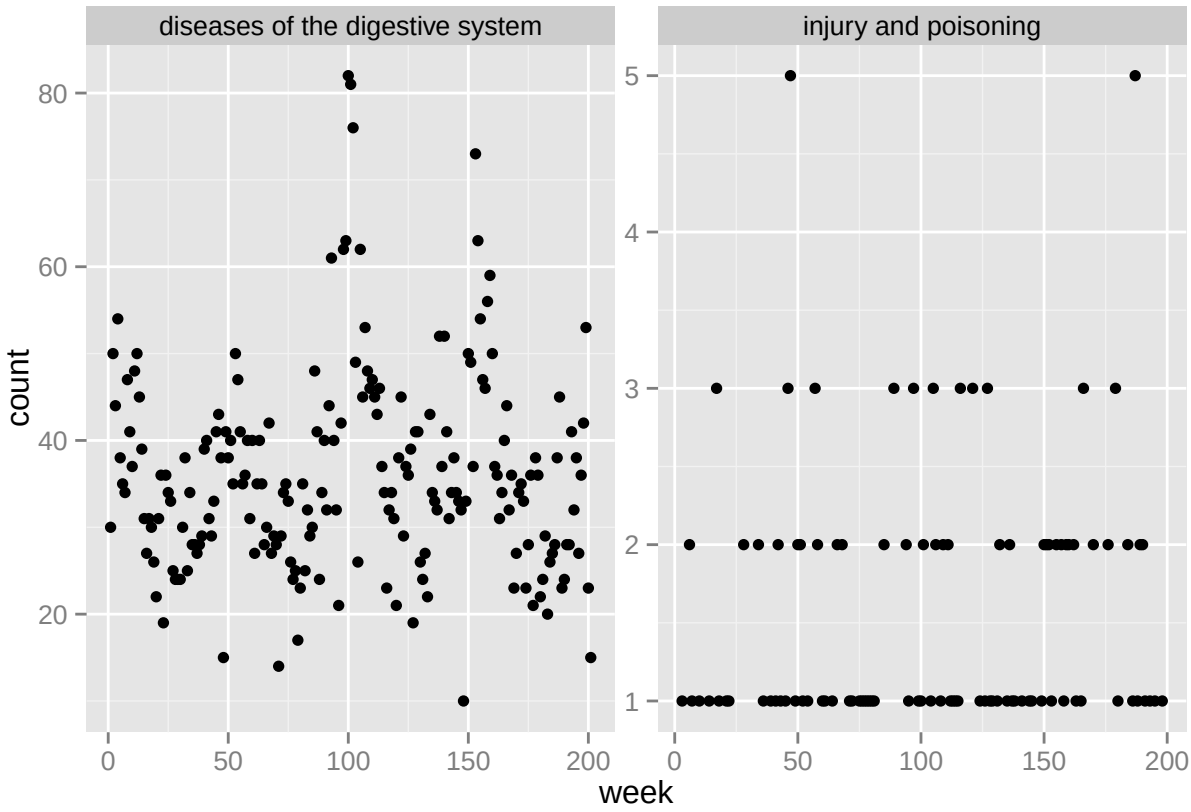
Subsetting by not in a single category

```
nIPD = subset(GI, icd9class != "infectious and parasitic disease")
nIPD_w = ddply(nIPD, .(week, icd9class), summarize, count = length(id))
ggplot(nIPD_w, aes(x=week, y=count)) + geom_point() + facet_wrap(~icd9class, scales="free")
```



Subsetting by not in a set of categories

```
nIPDp = subset(GI, !(icd9class %in% levels(icd9class)[c(1,3)]))
nIPDp_w = ddply(nIPDp, .(week, icd9class), summarize, count = length(id))
ggplot(nIPDp_w, aes(x=week, y=count)) + geom_point() + facet_wrap(~icd9class, scales="free")
```



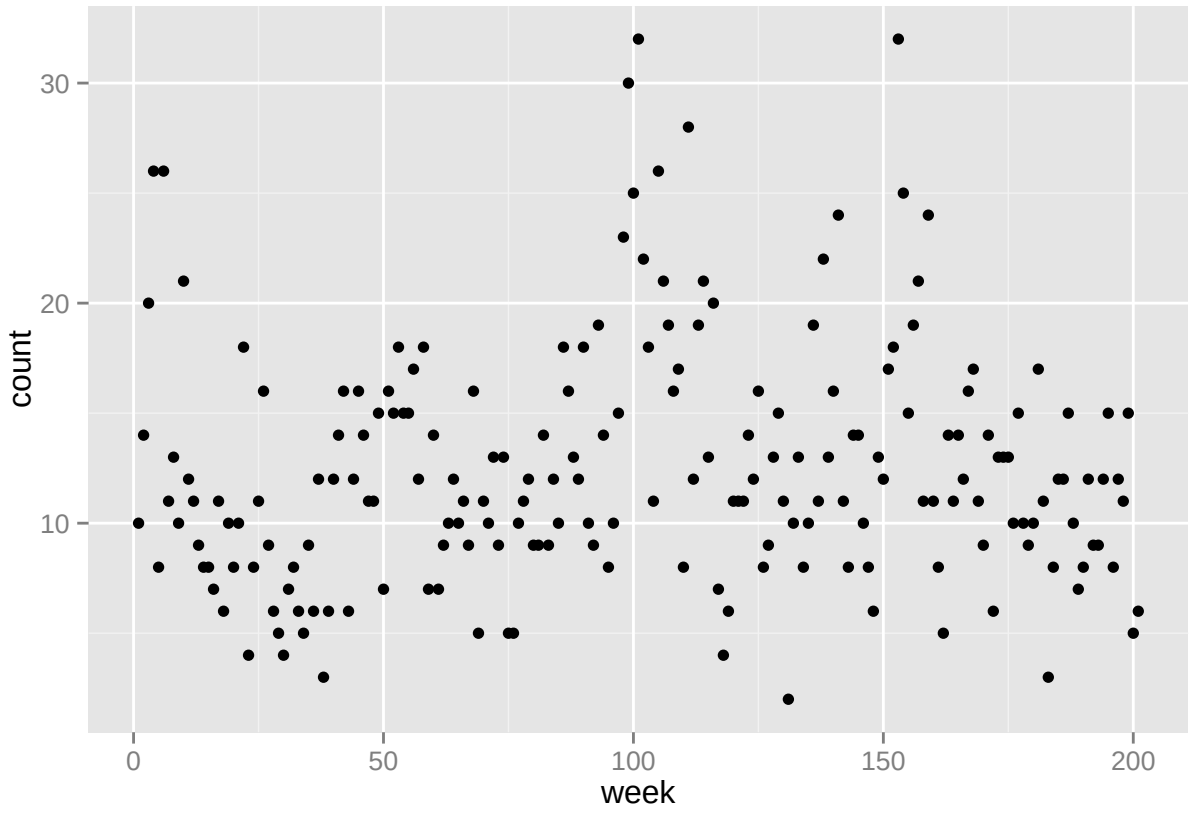
Subsetting by a continuous (numeric) variable

We may want to subset a continuous (numeric) variable by

- Larger than (or equal to) a number
- Small than (or equal to) a number
- In a range of numbers (with our without endpoints)

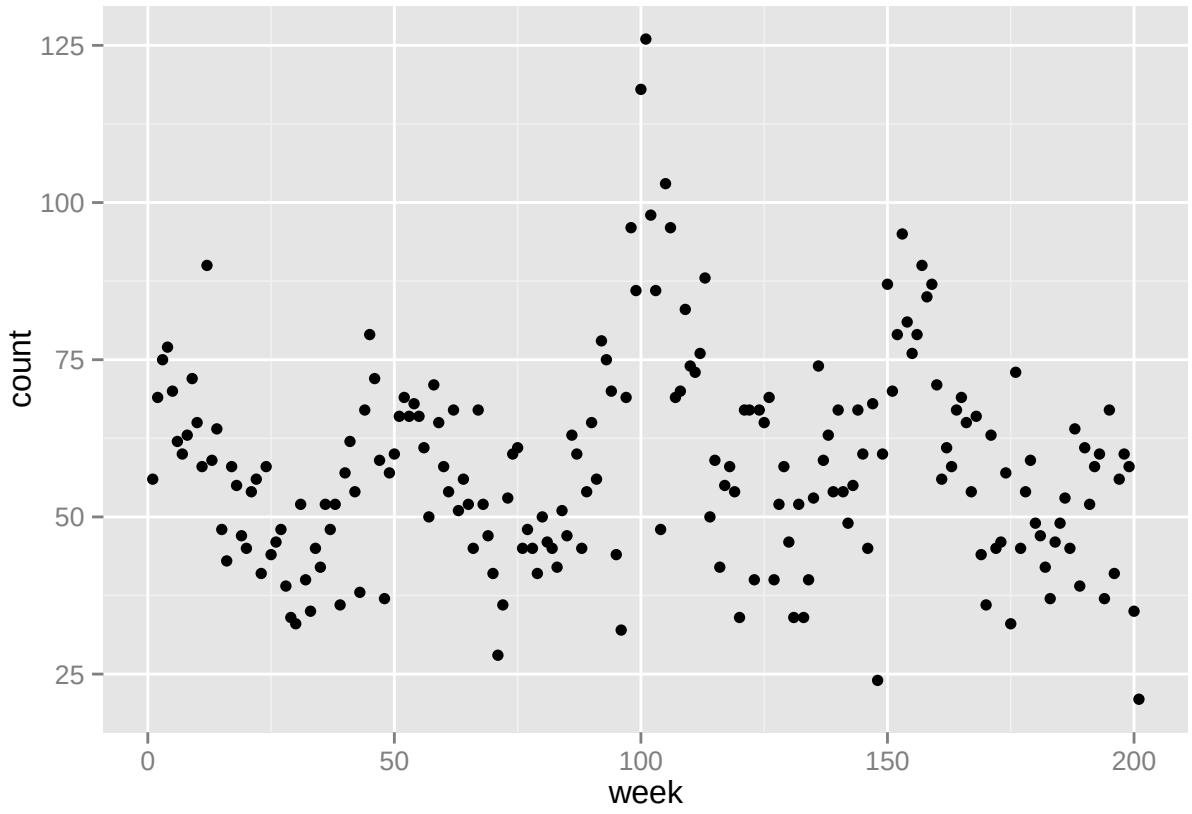
Subsetting larger than a number

```
Age60p = subset(GI, age > 60)
Age60p_w = ddply(Age60p, .(week), summarize, count = length(id))
ggplot(Age60p_w, aes(x=week, y=count)) + geom_point()
```



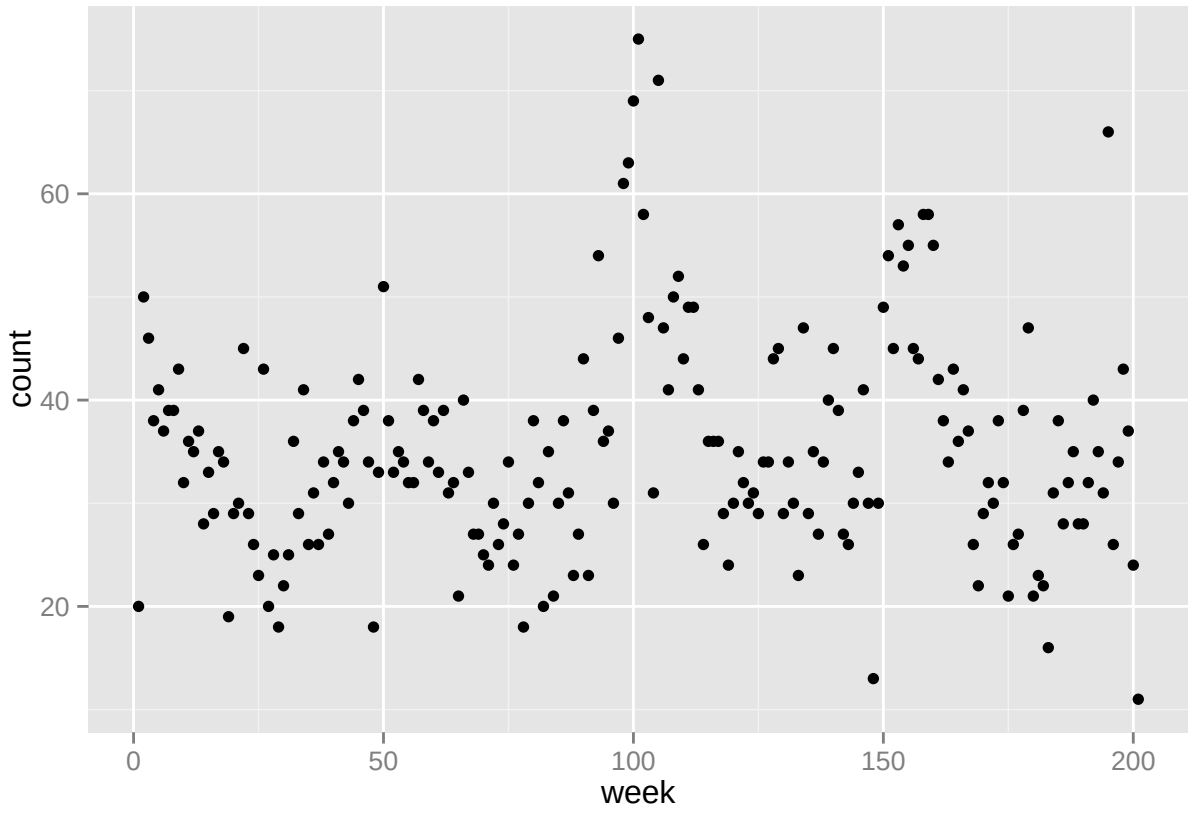
Subsetting less than or equal to a number

```
Age_lte30 = subset(GI, age <= 30)
Age_lte30_w = ddply(Age_lte30, .(week), summarize, count = length(id))
ggplot(Age_lte30_w, aes(x=week, y=count)) + geom_point()
```



Subsetting within a range

```
Age30to60 = subset(GI, age > 30 & age <= 60) # This includes those who are 60, but not those who are 30
Age30to60_w = ddply(Age30to60, .(week), summarize, count = length(id))
ggplot(Age30to60_w, aes(x=week, y=count)) + geom_point()
```



Subsetting a date

Subsetting a date works similar to subsetting a continuous (numeric) variable once you convert a comparison value to a `Date` using the `as.Date()` function.

Let's create a `Date` vector containing the two values we will use for subsetting.

```
two_dates = as.Date(c("2007-01-01", "2007-12-31"))
```

Subsetting dates

```
early = subset(GI, date <  two_dates[1])
late  = subset(GI, date >= two_dates[2])
mid   = subset(GI, date >= two_dates[1] & date < two_dates[2])
```

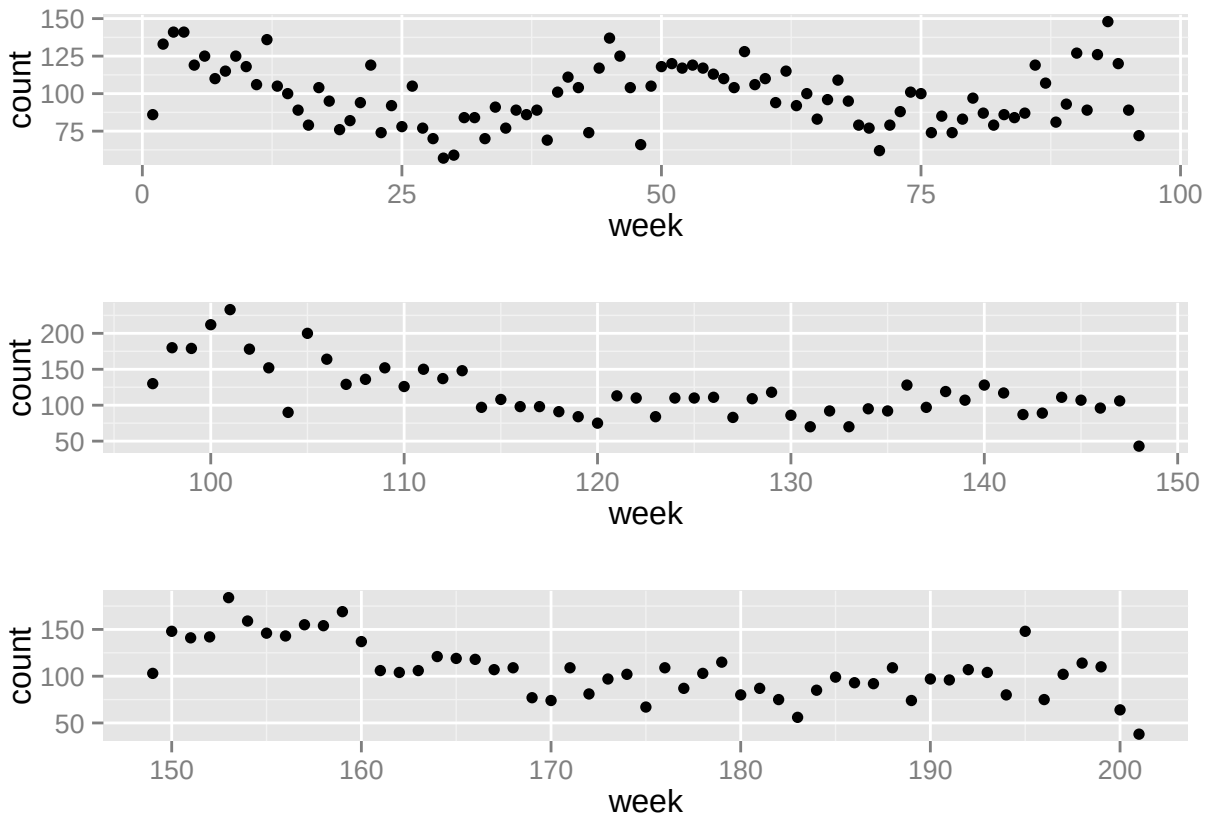
Now make plots

```
early_w = ddply(early, .(week), summarize, count = length(id))
late_w  = ddply(late,  .(week), summarize, count = length(id))
mid_w   = ddply(mid,   .(week), summarize, count = length(id))

g1 = ggplot(early_w, aes(x=week, y=count)) + geom_point()
g2 = ggplot(late_w,  aes(x=week, y=count)) + geom_point()
g3 = ggplot(mid_w,   aes(x=week, y=count)) + geom_point()
```

To arrange multiple plots, use the `grid.arrange` function in the `gridExtra` package.

```
library(gridExtra)
```

```
grid.arrange(g1,g3,g2)
```



Activity - subsetted graphs

Construct a plot for those in zipcode 206xx between Jan 1, 2008 and Dec 31, 2008.

When you have completed the activity, compare your results to the solutions.

Making professional looking graphics

To make the graphics we are producing look professional, we will need to customize the graph according to the final production platform, e.g. website, word document, pdf document, etc. This will be highly specific to your use. Nonetheless, likely you will use the following `ggplot2` functions:

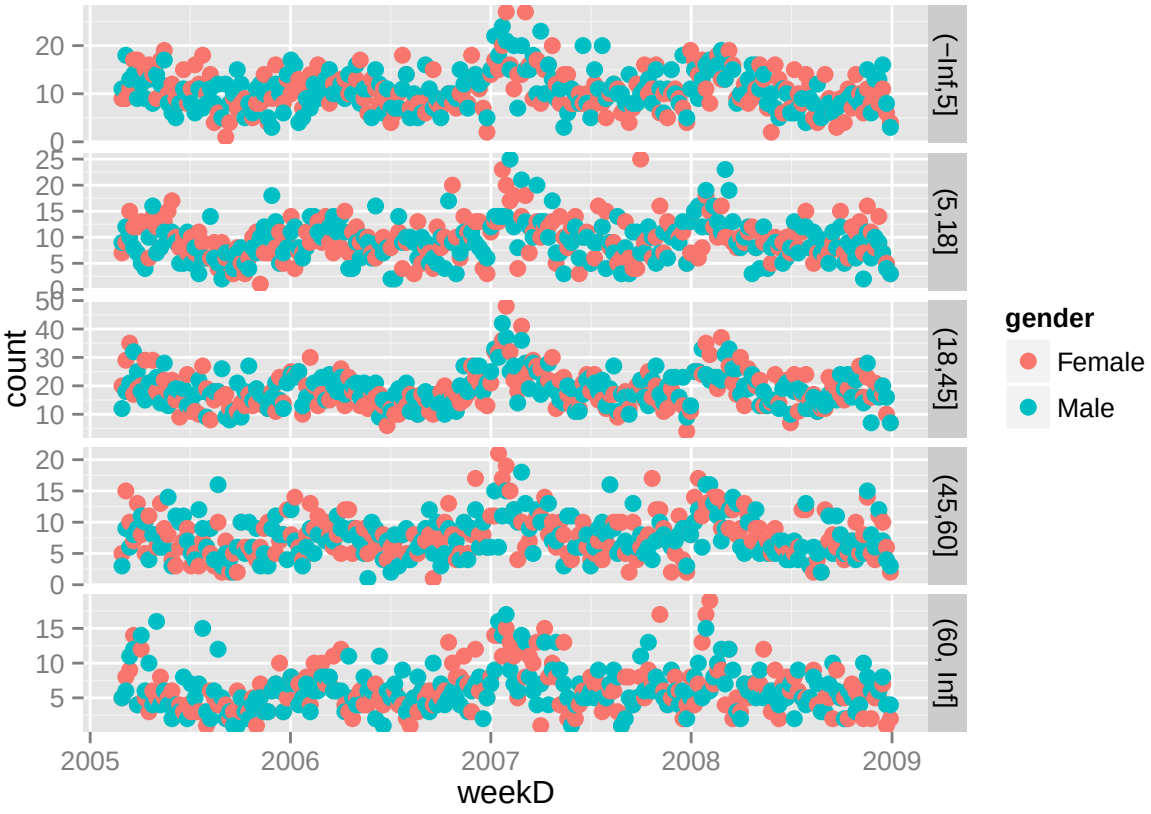
- `scale_color_manual`
- `labs`
- `theme`

But there are many others. One thing you will notice is that we can keep updating the graph by updating the R object that holds the graph.

Base graphic

Let's start with the following graph

```
GI$weekD = as.Date(GI$weekC)
GI_sum = ddply(GI, .(weekD, gender, ageC), summarize, count = length(id))
ggplot(GI_sum, aes(x=weekD, y=count, color=gender)) + geom_point(size=3) + facet_grid(ageC~., scales="fi
```



Change age category labels

The easiest way to change some of the labels on the graph is to change the underlying data. Here we change the levels associated with the `ageC` variable.

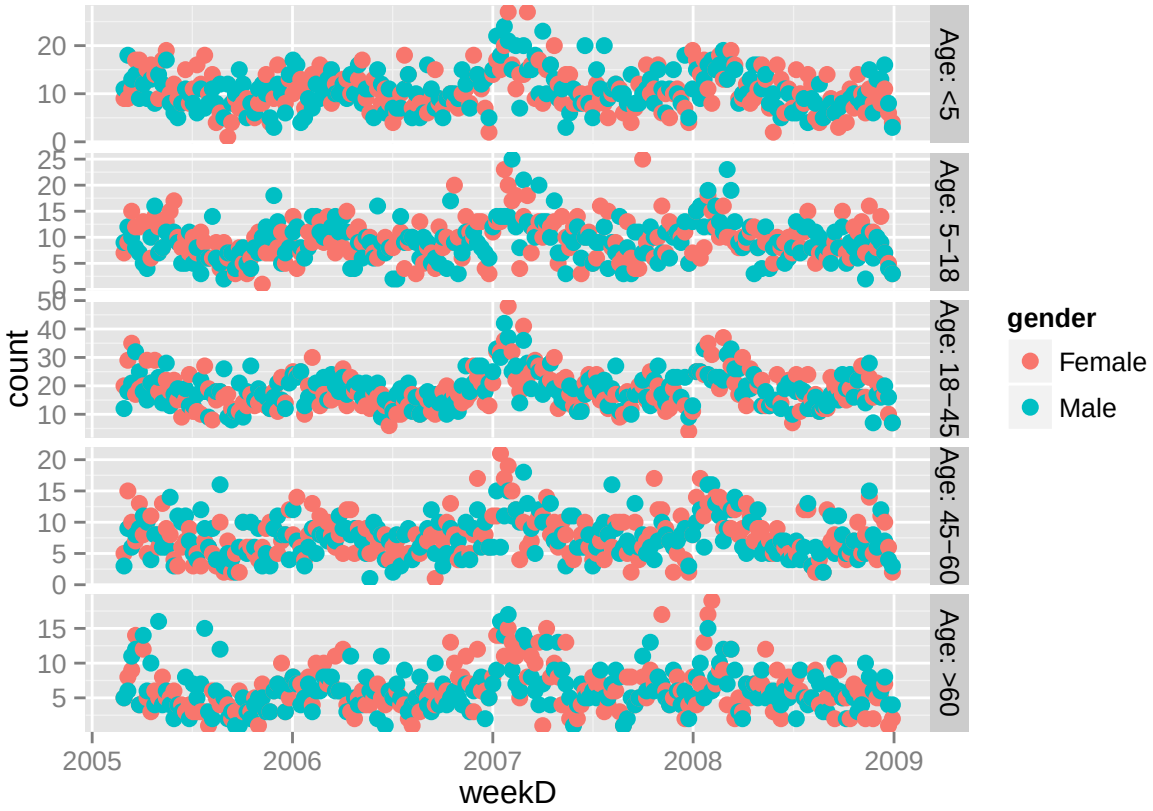
```
levels(GI_sum$ageC)
```

```
## [1] "(-Inf,5]"  "(5,18]"    "(18,45]"   "(45,60]"   "(60, Inf]"
```

```
levels(GI_sum$ageC) = paste("Age:", c("<5","5-18","18-45","45-60",">60"))
table(GI_sum$ageC)
```

```
##
##    Age: <5  Age: 5-18 Age: 18-45 Age: 45-60   Age: >60
##        402        402        402        402        400
```

```
g = ggplot(GI_sum, aes(x=weekD, y=count, color=gender)) + geom_point(size=3) + facet_grid(ageC~., scales
g
```

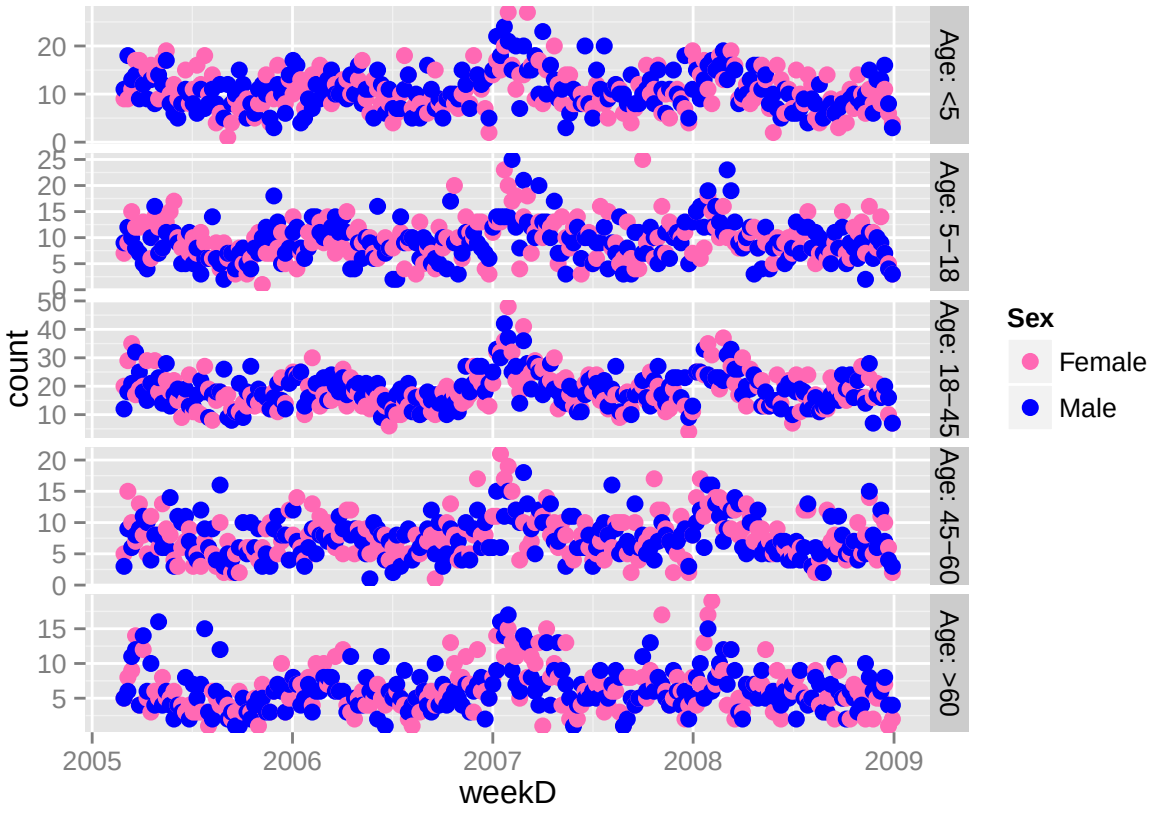


g is the R object that contains the graph

Change colors

Intuitive colors can make a graph easier to read.

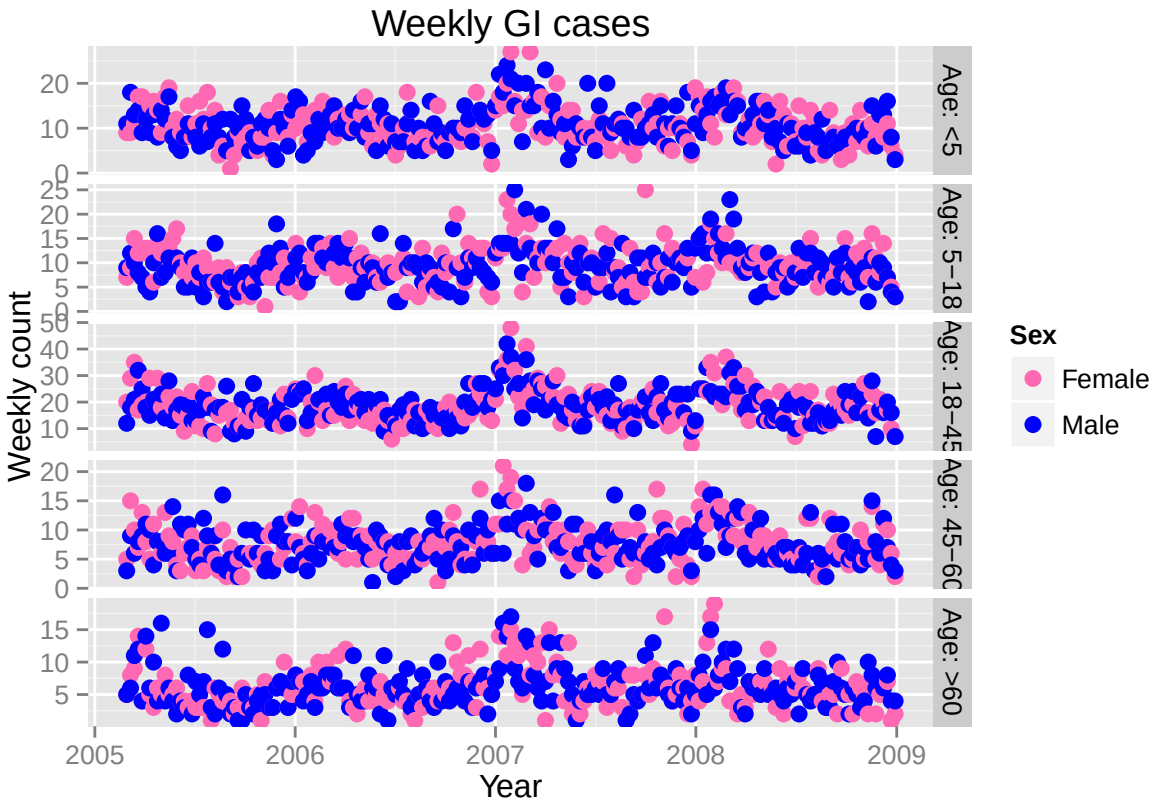
```
g = g + scale_color_manual(values=c("hotpink","blue"), name='Sex')
g
```



Change title and axis labels

Use proper names and capitalization in title and axis labels.

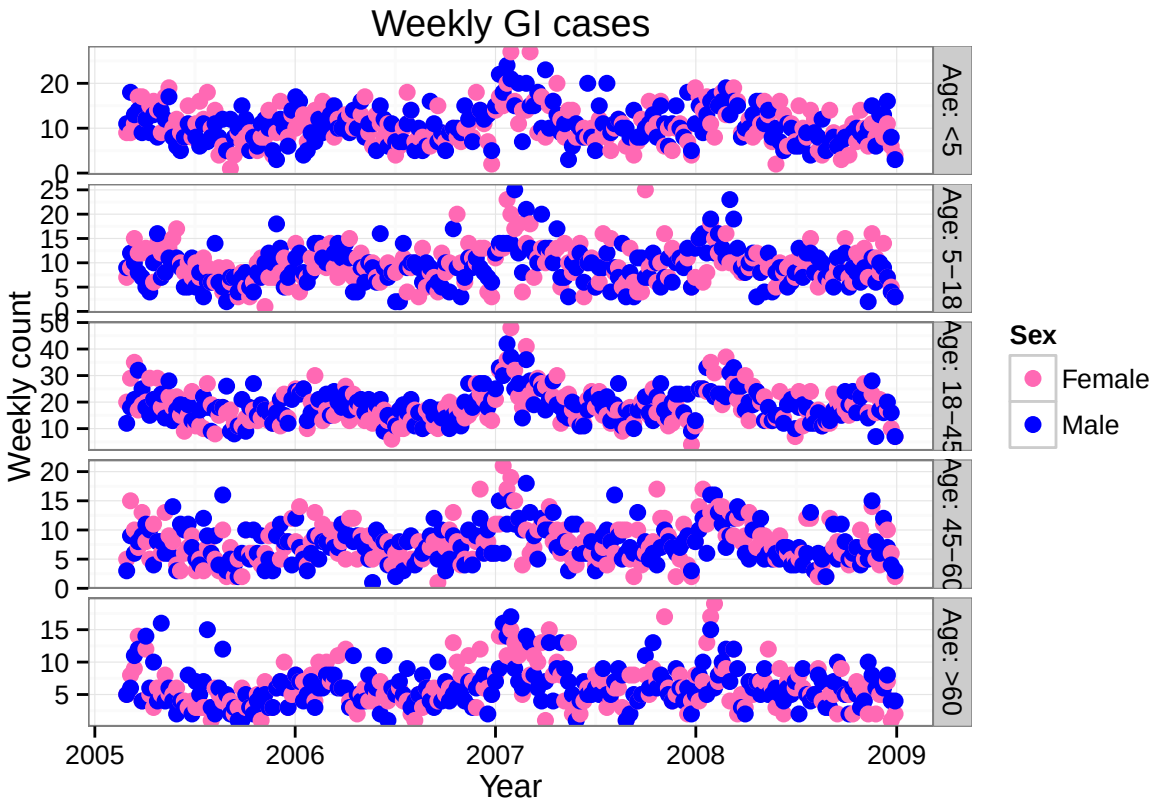
```
g = g + labs(title="Weekly GI cases", x="Year", y="Weekly count")
g
```



Try alternate themes

Try alternate themes:

```
g = g + theme_bw()
g
```



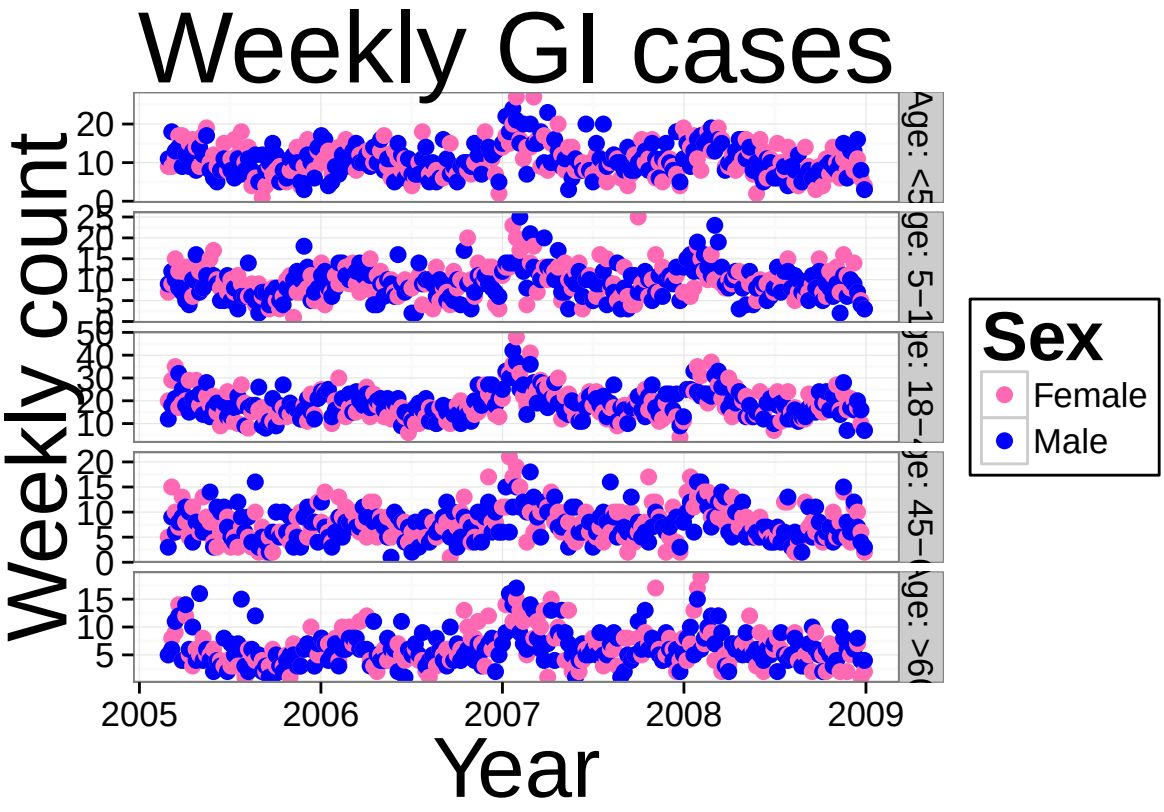
To see a list of possibilities:

```
?theme_bw
```

Adjust font sizes

Depending on the final platform, font sizes may need to be adjusted.

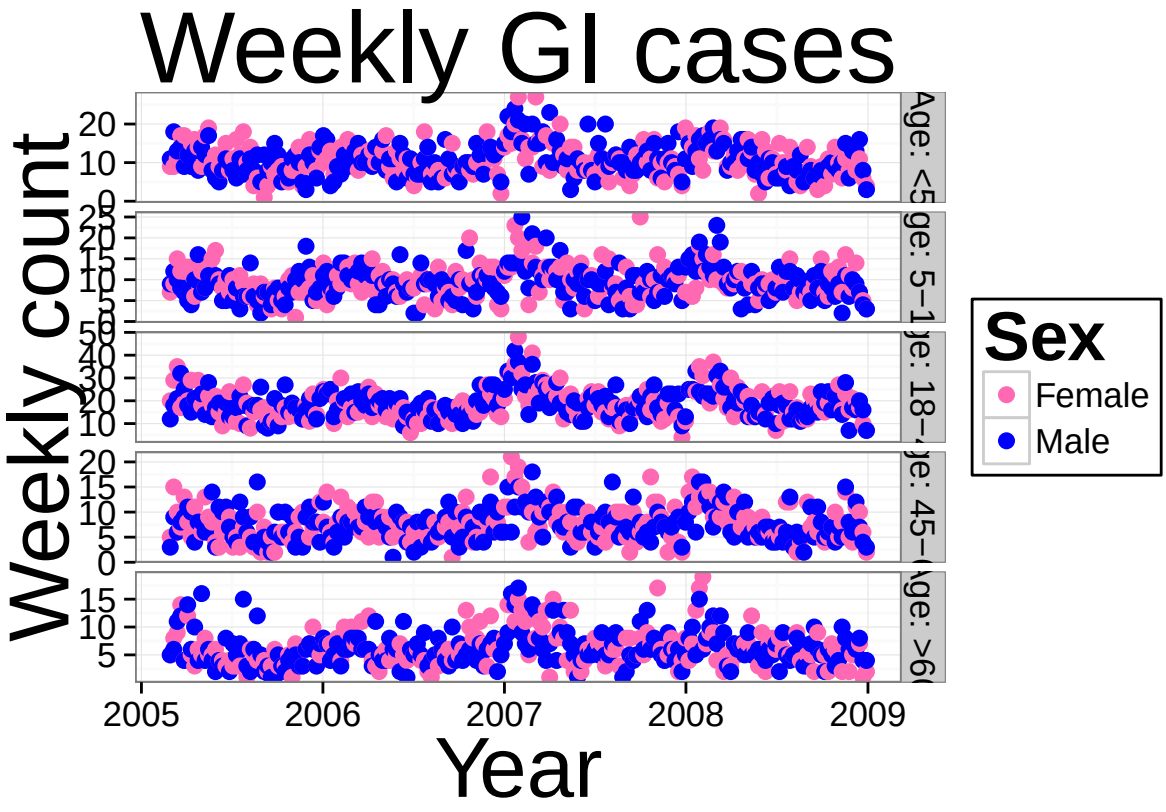
```
g = g + theme(title=element_text(size=rel(2)),
              text = element_text(size=16),
              legend.background = element_rect(fill="white", size=.5, color="black"))
g
```



Exporting graphs

To export the graphs, use the `ggsave()` function which saves the last plot that you displayed. As a default, it uses the size of the current graphics device, but you will likely want to modify this.

```
g
```




```
ggsave("plot.pdf")
```

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

If you look in your current working directory, you will see the plot.pdf file.

```
ggsave("plot.pdf", width=14, height=8)
```

Advanced Biosurveillance

Jarad Niemi

2014-12-04

Contents

```
library(ggplot2)
library(plyr)
library(reshape2)
library(ISDSWorkshop)
workshop(launch_index=FALSE)
```

```
# Read csv files
GI     = read.csv("GI.csv")
icd9df = read.csv("icd9.csv")
```

```
# Mutate data.frame
GI = mutate(GI,
            date      = as.Date(date),
            weekC     = cut(date, breaks="weeks"),
            week      = as.numeric(weekC),
            weekD     = as.Date(weekC),
            facility  = as.factor(facility),
            icd9class = factor(cut(icd9,
                           breaks = icd9df$code_cutpoint,
                           labels = icd9df$classification[-nrow(icd9df)],
                           right  = TRUE)),
            ageC      = cut(age,
                           breaks = c(-Inf, 5, 18, 45 ,60, Inf)),
            zip3      = trunc(zipcode/100))
```

Exporting tables

There are many ways to exporting tables. Here I will cover two basic approachess that are simple and work well.

- cut-and-paste
- create HTML table and cut-and-paste to Word

There are more sophisticated approaches that I will not cover:

- using the rtf package
- writing a whole doc fie in R

Cut-and-paste

You can cut-and-paste directly from R.

```
ga_l = ddply(GI, .(gender, ageC), summarize, count = length(id))
ga_w = dcast(ga_l, gender~ageC) # Long to wide
```

```
## Using count as value column: use value.var to override.
```

```
print(ga_w, row.names=FALSE)
```

```
##  gender (-Inf,5] (5,18] (18,45] (45,60] (60, Inf]
##  Female     2161   1942    3710    1590      1250
##    Male     2163   1860    3766    1543      1259
```

Cut-and-pasting this table is done in ASCII format. This looks good in a **plain text** document, e.g. Notepad, TextEdit, and some email, but will not look good in other formats, e.g. docx.

Create HTML table

```
library(xtable)
tab = xtable(ga_w,
             caption = "Total GI cases by Sex and Age Category",
             label   = "myHTMLanchor",
             align   = "ll|rrrrr") # rownames gets a column
```

Save the table to a file

```
print(tab, file="table.html", type="html", include.rownames=FALSE)
```

Output for this HTML table

The HTML code looks like

```
print(tab, type="html", include.rownames=FALSE)
```

```
## <!-- html table generated in R 3.1.1 by xtable 1.7-4 package -->
## <!-- Thu Dec  4 13:03:00 2014 -->
## <table border=1>
## <caption align="bottom"> Total GI cases by Sex and Age Category </caption>
## <tr> <th> gender </th> <th> (-Inf,5] </th> <th> (5,18] </th> <th> (18,45] </th> <th> (45,60] </th> <t
##   <tr> <td> Female </td> <td align="right"> 2161 </td> <td align="right"> 1942 </td> <td align="right
##   <tr> <td> Male </td> <td align="right"> 2163 </td> <td align="right"> 1860 </td> <td align="right"
##    <a name=myHTMLanchor></a>
## </table>
```

and the resulting table looks like

```
print(tab, type="html", include.rownames=FALSE)
```

Total GI cases by Sex and Age Category

gender

(-Inf,5]

(5,18]

(18,45]

(45,60]

(60, Inf]

Female

2161

1942

3710

1590

1250

Male

2163

1860

3766

1543

1259

Copy-and-paste table to Word

Now you can

1. Open the file (`table.html`)
2. Copy-and-paste this table to Word.

Activity - exporting tables

Create a Word table for the number of cases by facility and age category.

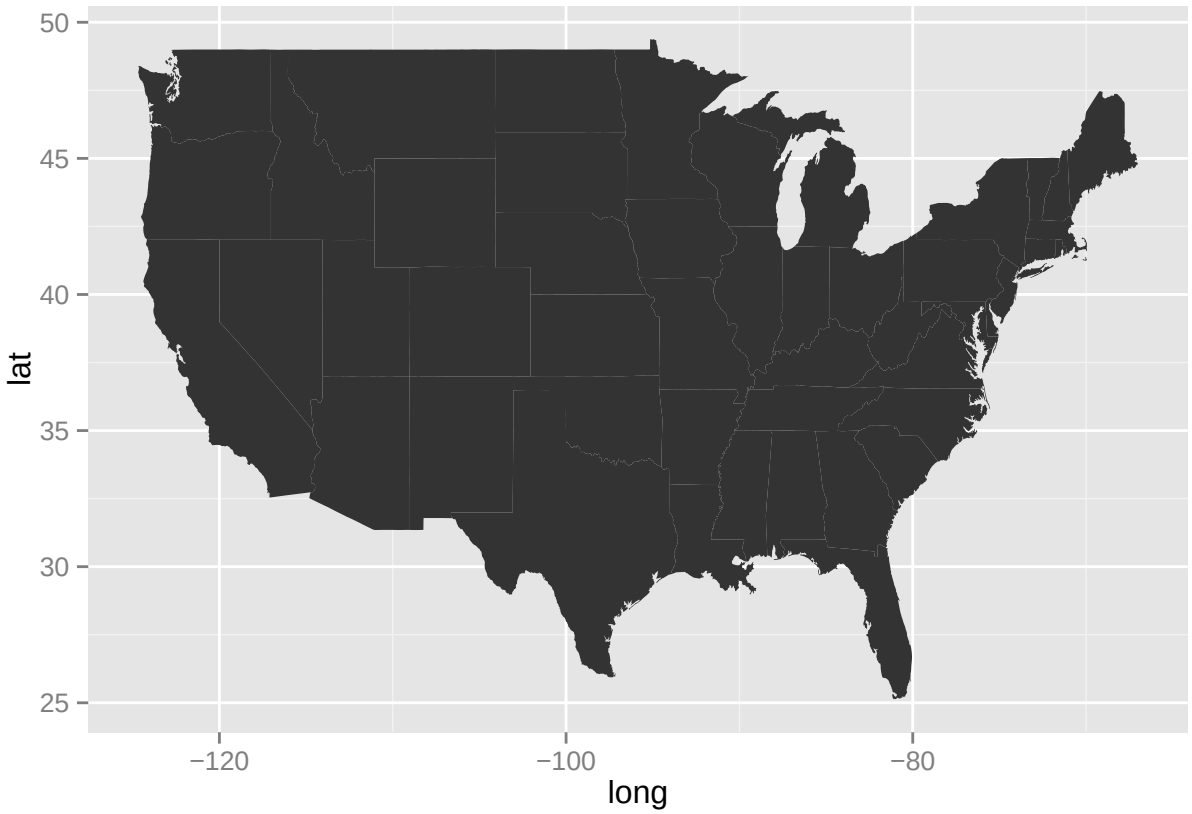
When you have completed the activity, compare your results to the solutions.

Maps

The packages `ggplot2` and `maps` can be used together to make maps.

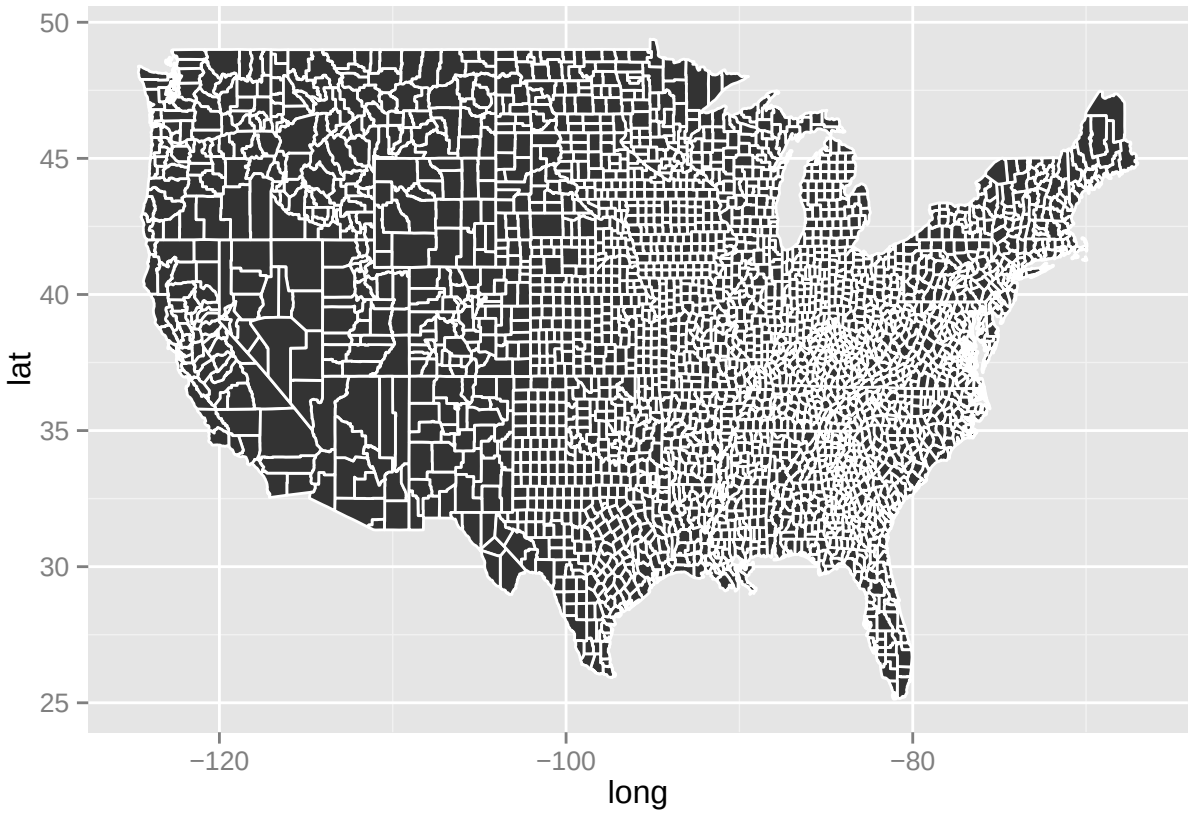
Map of the continental US

```
library(maps)
states = map_data("state")
ggplot(states, aes(x=long, y=lat, group=group)) + geom_polygon()
```



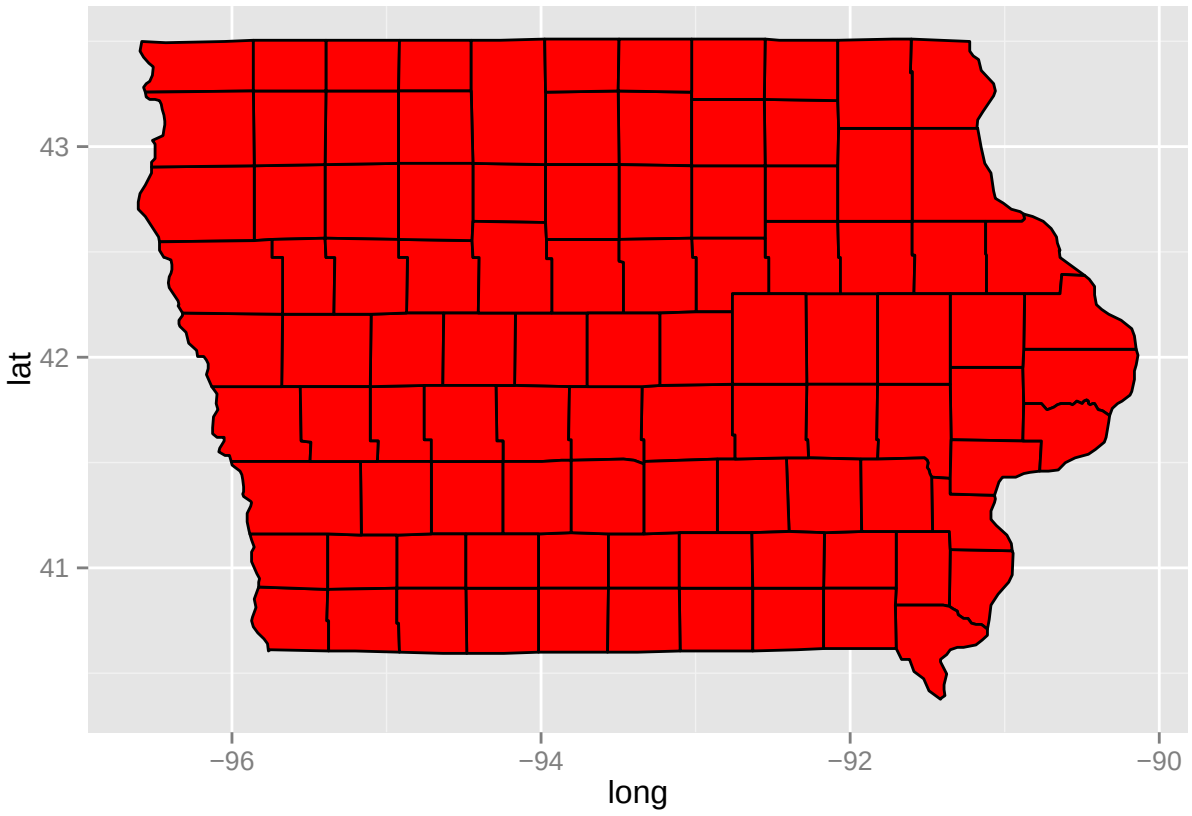
Map of the counties in the continental US

```
counties = map_data("county")
ggplot(counties, aes(x=long, y=lat, group=group)) + geom_polygon(color="white")
```



Map of the counties in Iowa

```
ggplot(subset(counties, region=="iowa"),
       aes(x=long, y=lat, group=group)) + geom_polygon(fill="red", color="black")
```



Construct an appropriate data set

To make an informative map, we need to add data.

```
fluTrends = read.csv("fluTrends.csv", check.names=FALSE)
```

For simplicity, only keep the most recent 12 weeks on states.

```
nr = nrow(fluTrends)
flu_w = fluTrends[(nr-11):nr, c(1,3:53)]
dim(flu_w)
```

```
## [1] 12 52
```

Reshape to long format

```
flu_l = melt(flu_w, id.var='Date',
             variable.name='region', # to match map_data
             value.name='index')
```

Merge fluTrends data with map_data

The region names in `map_data` are lower case, so use `tolower()` to convert all the region names in flu_l to lowercase. Then the `map_data` and fluTrends data are merged using `merge()`.

```
head(unique(states$region))
```

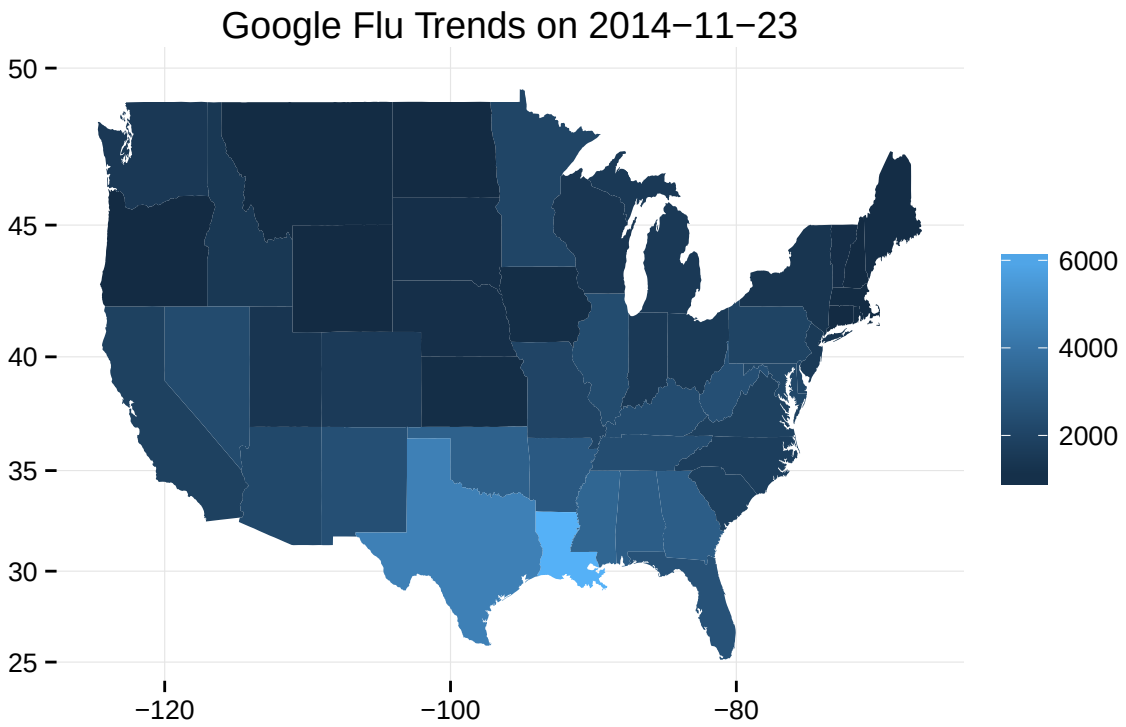
```
## [1] "alabama"     "arizona"     "arkansas"    "california"  "colorado"
## [6] "connecticut"
```

```
flu_l$region = tolower(flu_l$region)
states_merged = merge(states, flu_l, sort=FALSE, by='region')
```

Make the plots

Most recent Google Flu Trend data.

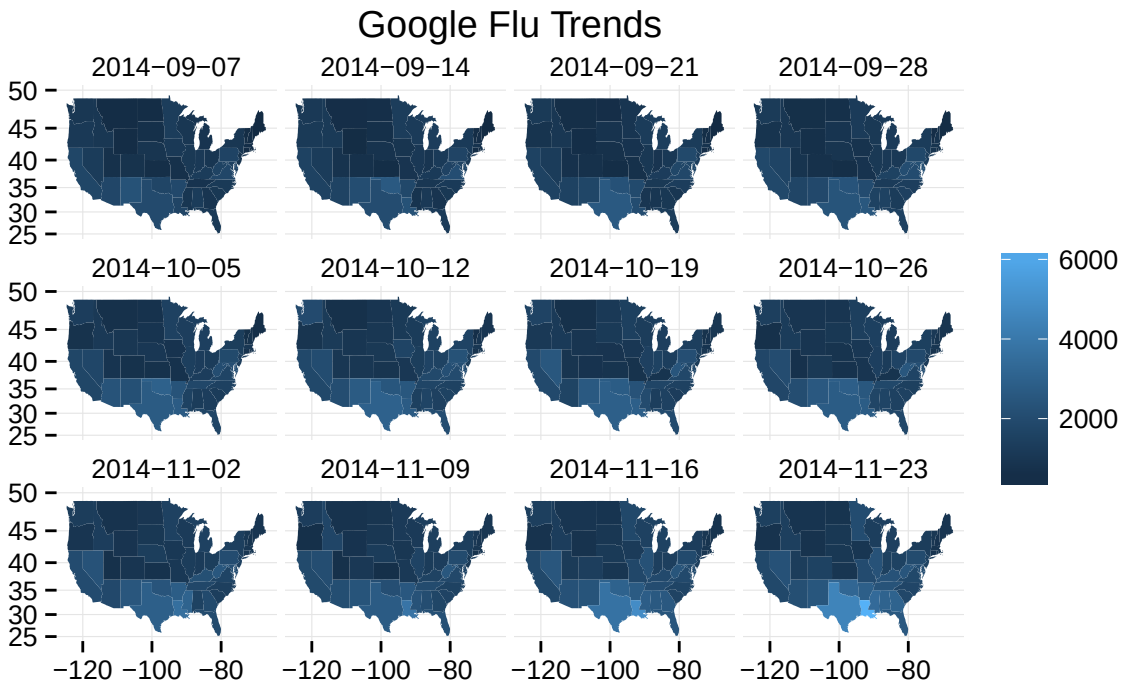
```
states_merged$Date = as.Date(states_merged$Date)
mx_date = max(states_merged$Date)
ggplot(subset(states_merged, Date == mx_date),
       aes(x=long, y=lat, group=group, fill=index)) +
  geom_polygon() +
  labs(title=paste('Google Flu Trends on', mx_date), x='', y='') +
  theme_minimal() +
  theme(legend.title = element_blank()) +
  coord_map("cylindrical")
```



Last 12 weeks.

```
ggplot(states_merged,
       aes(x=long, y=lat, group=group, fill=index)) +
  geom_polygon() +
  labs(title='Google Flu Trends', x='', y='') +
```

```
  theme_minimal() +
  theme(legend.title = element_blank()) +
  facet_wrap(~Date) +
  coord_map("cylindrical")
```



Activity

Modify the code to determine what elements of the map are affected.

Advanced: Download the data directly from http://www.google.org/flutrends/us/data.txt using 'read.csv' and then construct a map of the most recent Google Flu Trends data.

When you have completed the activity, compare your results to the solutions.

Packages

A package provides additional functionality besides what base installation of R provides. You have already used a number of additional packages:

- ggplot2
- ISDSWorkshop
- maps
- plyr
- reshape2
- xtable

The Comprehensive R Archive Network (CRAN) has over 6,000 packages. These packages can be installed using the `install.packages()` function, e.g.

```
install.packages("ggplot2")
```

For many reasons, packages may not be on CRAN but may be available from other sources:

- Github
- Bioconductor
- Source file (tar.gz)

Github

To install a package from github, you can use the `install_github()` function from the `devtools` package. For example,

```
# install.packages("devtools")
library(devtools)
install_github('nandadorea/vetsyn')
```

This will not install any vignettes which is why I didn't use this method for this workshop.

Bioconductor

Bioconductor provides tools for high-throughput genomic data. There are over 900 packages available from bioconductor. To install bioconductor, use

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

Then to install a package from bioconductor use

```
biocLite("edgeR")
```

The bioconductor repository may be useful in the future for public health biosurveillance.

Source

All packages can be installed from their source. If a package is not available in a repository, then this is the only way to install the package. To install from source download the source file (.tar.gz) and then use the `install.packages()` function with arguments `repos=NULl` and `type="source"`. For example, to install the ISDSWorkshop package you used

```
install.packages("ISDSWorkshop_0.1.tar.gz",
                 repos = NULL,
                 type  = "source")
```

Packages for surveillance

Some packages for performing surveillance are

- accrued
- Epi
- SpatialEpi
- surveillance
- vetsyn

SpatialEpi - Kulldorff example

The `SpatialEpi` package implements the Kulldorff cluster detection method in the `kulldorff()` function.

Setting up the analysis (taken directly from the example)

```
library(SpatialEpi)
```

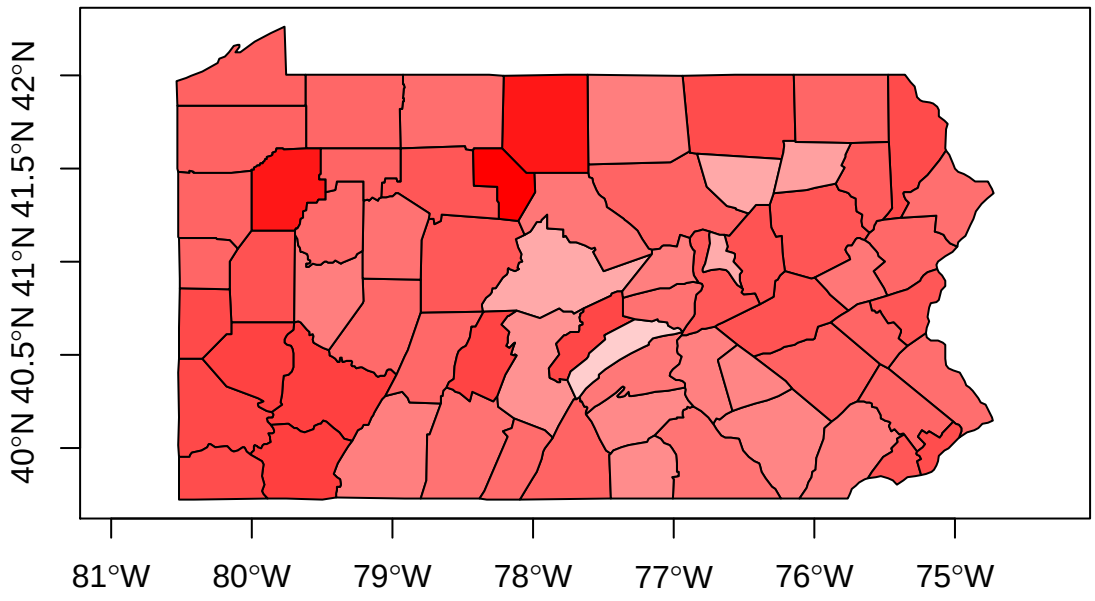
```
## Loading required package: sp
```

```
## Load Pennsylvania Lung Cancer Data
data(pennLC)
data <- pennLC$data

## Process geographical information and convert to grid
geo <- pennLC$geo[,2:3]
geo <- latlong2grid(geo)

## Get aggregated counts of population and cases for each county
population <- tapply(data$population, data$county, sum)
cases      <- tapply(data$cases,      data$county, sum)

## Based on the 16 strata levels, compute expected numbers of disease
n.strata       <- 16
expected.cases <- expected(data$population, data$cases, n.strata)

## Set Parameters
pop.upper.bound <- 0.5
n.simulations   <- 999
alpha.level     <- 0.05
plot            <- TRUE
```

SpatialEpi - Kulldorff example

Plot the data

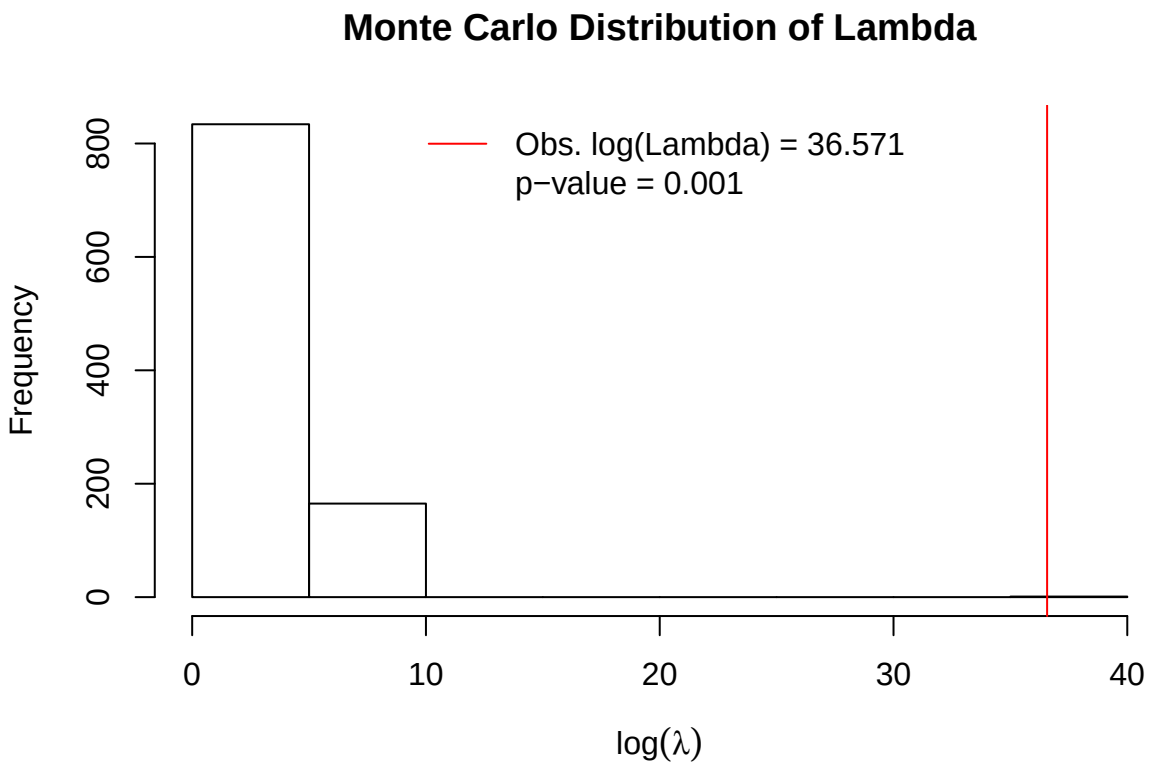
```
clr = cases/population
clr = 1-clr/max(clr) # make sure range is (0,1)
plot(pennLC$spatial.polygon, axes=TRUE, col=rgb(1, clr, clr))
```



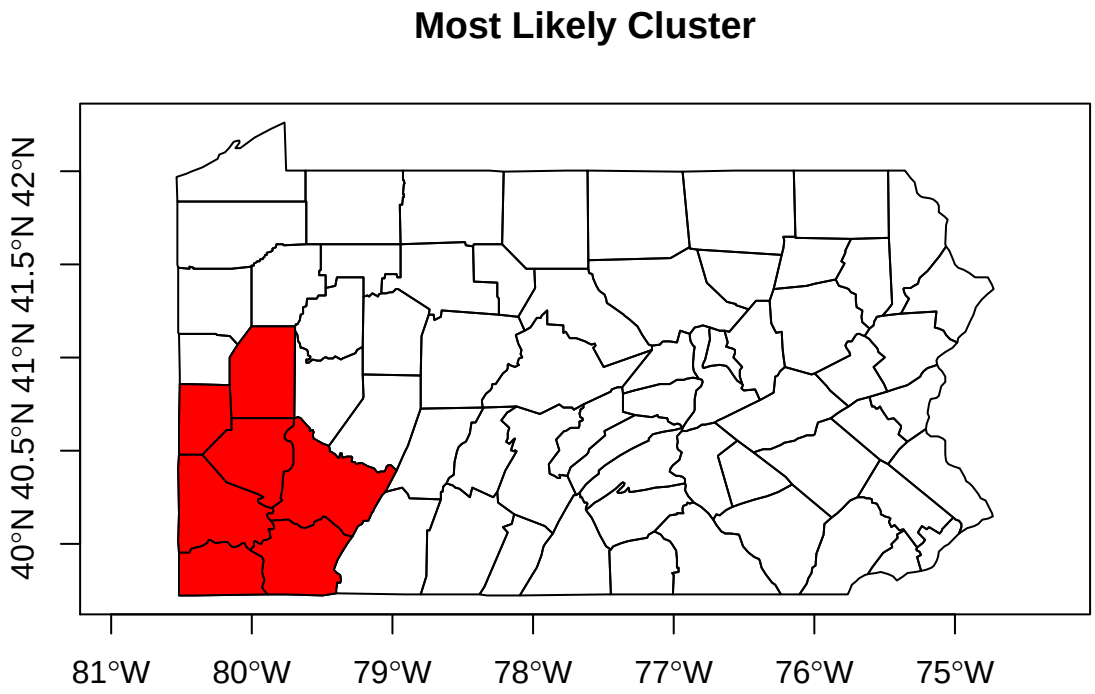
SpatialEpi - Kulldorff example

Run the analysis and plot the results (directly from the example)

```
## Kulldorff using Binomial likelihoods
binomial <- kulldorff(geo, cases, population, NULL, pop.upper.bound, n.simulations,
  alpha.level, plot)
```



```
cluster <- binomial$most.likely.cluster$location.IDs.included

## plot
plot(pennLC$spatial.polygon,axes=TRUE)
plot(pennLC$spatial.polygon[cluster],add=TRUE,col="red")
title("Most Likely Cluster")
```



Activity - Install the `surveillance` package

If you are connected to the internet, try installing the `surveillance` package. If you are successful, look at its help file.

When you have completed the activity, compare your results to the solutions.

Functions

Packages are typically a collection of functions. But you can write your own. For example,

```
add = function(a,b) {
  return(a+b)
}
add(1,2)
```

```
## [1] 3
```

or

```
add_vector = function(v) {
  sum = 0
  for (i in 1:length(v)) sum = sum + v[i]
  return(sum)
}
```

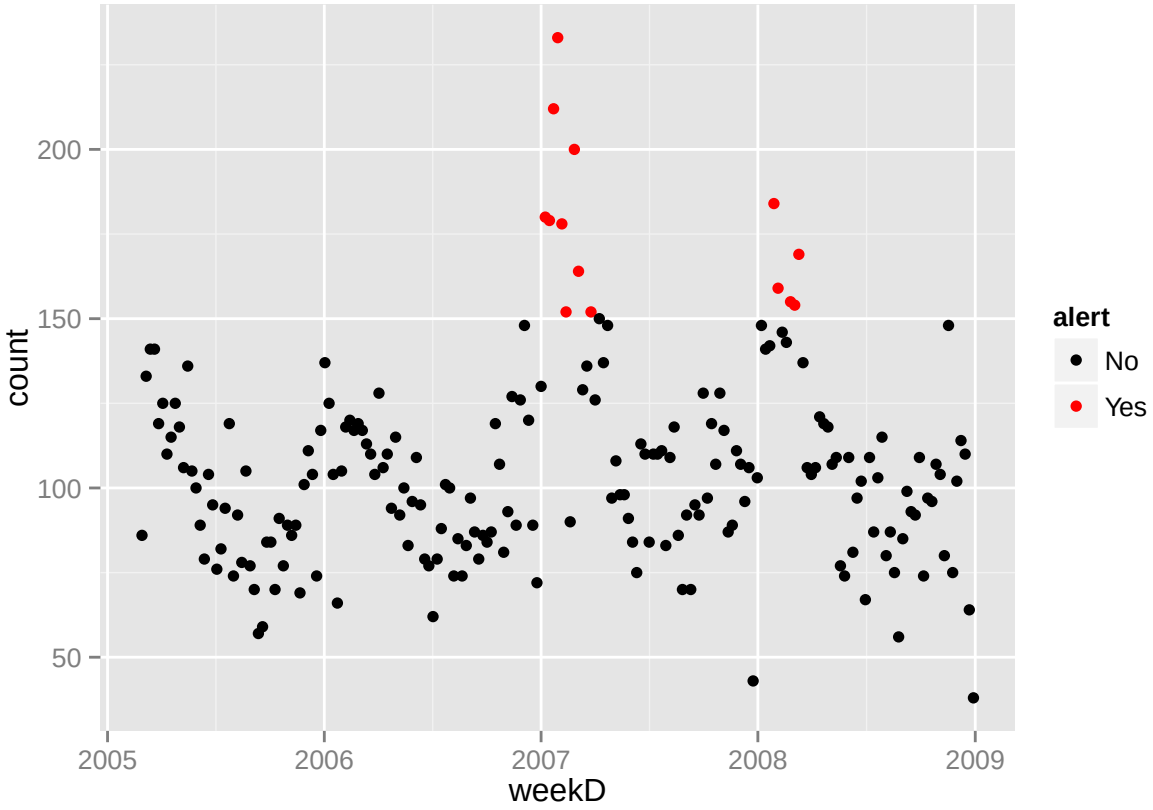
A simple outbreak detection function

Here is a simple outbreak detection function

```
alert = function(y,threshold=100) {
  # y is the time series
  # an alert is issue for any y above the threshold (default is 100)
  factor(ifelse(y>threshold, "Yes", "No"))
}
```

Run this on our weekly GI data.

```
GI_w = ddply(GI, .(weekD), summarize, count = length(id))
GI_w$alert = alert(y = GI_w$count, threshold = 150)
ggplot(GI_w, aes(x=weekD, y=count, color=alert)) +
  geom_point() +
  scale_color_manual(values=c("black","red"))
```



R in batch

For routine analysis, it can be helpful to run R in batch mode rather than interactively. To do this, create a script and save the script with a .R extension, e.g. `script.R`:

```
# Read in the data perhaps as a csv file

# Create some table and save them to html files

# Create some figures and save them to jpegs

# Run some outbreak detection algorithms and produce figures
```

Now, from the command line run

```
R CMD BATCH script.R
```

Now each week you can update the csv file and then just run the script which will create all new tables and figures.

Shiny apps

R can be used to create HTML applications through the `shiny` package. The code is written entirely in R, although you can use cascading style sheets (CSS) if you want to update how it looks. Here is an example.