Lecture 3

Lists and Data Cleaning

Andrew Jaffe, John Muschelli
Dynamic Duo

More resources

- UCLA Institute for Digital Research and Education: http://www.ats.ucla.edu/stat/r/
- R reference card: http://cran.r-project.org/doc/contrib/Short-refcard.pdf
- Undergrad Guide to R: https://sites.google.com/site/undergraduateguidetor/
- Quick R: http://statmethods.net/

Extra Credit

Completing all 7 levels of the "Try R" course on Code School will replace your lowest homework score with a 100%

http://www.codeschool.com/courses/try-r

Just save a screenshot of this page with the challenges completed:

http://tryr.codeschool.com/levels/7/challenges/1

Quiz!

"Open Book" quiz, you have 10 minutes.

We will go over the answers after everyone turns it in

Review of Days 1 and 2

- Reading data into R {read.table()}
- Subsetting vectors {[ind]} and data frames {[row,col]}
- Creating logical tests for variables in your dataset
- Creating new variables
  - Binary
  - Categorical
  - Transforming, e.g. log(), exp(), sqrt()
- Summarizing variables
  - Basic statistics, e.g. mean(), sum(), sd()
  - One variable by levels of another variable: tapply()
  - Basic exploratory plots

You should feel comfortable doing most of the above

Data

- We will be using multiple data sets in this lecture:
  - Salary, Monument, Circulator, and Restaurant from OpenBaltimore: https://data.baltimorecity.gov/browse?limitTo=datasets
  - Gap Minder - very interesting way of viewing longitudinal data
    - Data is here - http://www.gapminder.org/data/
  - http://spreadsheets.google.com/pub?key=rMsQHawTObBb6_U2ESjKXYw&output=xls
  - Also located at http://biostat.jhsph.edu/~ajaffe/indicator_estimatedincidencealltbper100000.xlsx
- Let us know if you have data that is much more complicated

Lists

- One other data type that is the most generic are lists.
- Can be created using list()
- Can hold vectors, strings, matrices, models, list of other list, lists upon lists!
- Can reference data using $ (if the elements are named), or using [], or [[]]

```
> mylist <- list(letters = c("A", "b", "c"), numbers = 1:3, matrix(1:25, ncol = 5))
```

List Structure

```
> head(mylist)
```

```
$letters
[1] "A" "b" "c"

$numbers
[1] 1 2 3

[[3]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

List referencing

```
> mylist[1]  # returns a list
```

```
$letters
[1] "A" "b" "c"
```

```
> mylist["letters"]  # returns a list
```

```
$letters
[1] "A" "b" "c"
```

List referencing

```
> mylist[[1]]  # returns the vector 'letters'
```

```
[1] "A" "b" "c"
```

```
> mylist$letters  # returns vector
```

```
[1] "A" "b" "c"
```

```
> mylist[["letters"]]  # returns the vector 'letters'
```

```
[1] "A" "b" "c"
```

List referencing

You can also select multiple lists with the single brackets.

```
> mylist[1:2]  # returns a list
```

```
$letters
[1] "A" "b" "c"

$numbers
[1] 1 2 3
```

List referencing

You can also select down several levels of a list at once

```
> mylist$letters[1]
```

```
[1] "A"
```

```
> mylist[[2]][1]
```

```
[1] 1
```

```
> mylist[[3]][1:2, 1:2]
```

```
     [,1] [,2]
[1,]    1    6
[2,]    2    7
```

Data Cleaning

In general, data cleaning is a process of investigating your data for inaccuracies, or recoding it in a way that makes it more manageable.

MOST IMPORTANT RULE - LOOK AT YOUR DATA!

Again - table, summarize, is.na, any, all are useful.

Data Cleaning

```
> table(c(0, 1, 2, 3, NA, 3, 3, 2, 2, 3), useNA = "ifany")
```

```

   0    1    2    3 <NA> 
   1    1    3    4    1 
```

```
> table(c(0, 1, 2, 3, 2, 3, 3, 2, 2, 3), useNA = "always")
```

```

   0    1    2    3 <NA> 
   1    1    4    4    0 
```

```
> tab <- table(c(0, 1, 2, 3, 2, 3, 3, 2, 2, 3), c(0, 1, 2, 3, 2, 3, 3, 4, 4, 3),
+     useNA = "always")
> margin.table(tab, 2)
```

```

   0    1    2    3    4 <NA> 
   1    1    2    4    2    0 
```

```
> prop.table(tab, 2)  # tab x y, col in stata (1 for row), neither for cell
```

```

         0   1   2   3   4 <NA>
```

Data Cleaning

- any - checks if there are any TRUES
- all - checks if ALL are true

```
> any(is.na(Sal$Name))
```

```
[1] FALSE
```

```
> # remove leading $ off money amount
> sals <- as.numeric(gsub(pattern = "$", replacement = "", Sal$AnnualSalary, ,
+     fixed = TRUE))
> quantile(sals)
```

```
    0%    25%    50%    75%   100%
   377  31609  43614  59916 238772
```

Cross Tabs

- xtabs allows you to look at multiple levels

```
> warpbreaks$replicate <- rep(1:9, len = nrow(warpbreaks))
> print(xt <- xtabs(breaks ~ wool + tension + replicate, data = warpbreaks))
```

```
, , replicate = 1

    tension
wool  L  M  H
   A 26 18 36
   B 27 42 20

, , replicate = 2

    tension
wool  L  M  H
   A 30 21 21
   B 14 26 21

, , replicate = 3

    tension
wool  L  M  H
   A 54 29 24
   B 29 19 24

, , replicate = 4

    tension
wool  L  M  H
   A 25 17 18
```

Flat Contingency Tables: ftable()

```
> ftable(xt)
```

```
             replicate  1  2  3  4  5  6  7  8  9
wool tension
A    L                 26 30 54 25 70 52 51 26 67
     M                 18 21 29 17 12 18 35 30 36
     H                 36 21 24 18 10 43 28 15 26
B    L                 27 14 29 19 29 31 41 20 44
     M                 42 26 19 16 39 28 21 39 29
     H                 20 21 24 17 13 15 15 16 28
```

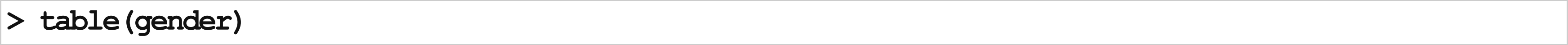
Example of Cleaning:

For example, let's say gender was coded as Male, M, m, Female, F, f. Using Excel to find all of these would be a matter of filtering and changing all by hand or using if statements.

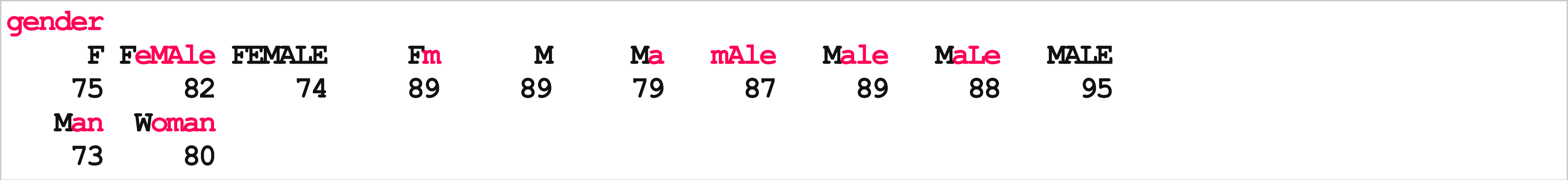
In R, you can simply do something like:

```
data$gender[data$gender %in% c("Male", "M", "m")] <- "Male"
```

Sometimes though, it's not so simple. That's where functions that find patterns come in very useful.



```
> table(gender)
```



```
gender
     F FeMAle FEMALE     Fm      M     Ma   mAle   Male   MaLe   MALE
    75     82     74     89     89     79     87     89     88     95
   Man  Woman
    73     80
```

Find/Replace and Regular Expressions

- R can do much more than find exact matches for a whole string
- Like Perl and other languages, it can use regular expressions.
- What are regular expressions?
- Ways to search for specific strings
- Can be very complicated or simple
- Highly Useful

'Find' functions

grep: grep, grepl, regexpr and gregexpr search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.

grep(pattern, x, fixed=FALSE), where:

- pattern = character string containing a regular expression to be matched in the given character vector.
- x = a character vector where matches are sought, or an object which can be coerced by as.character to a character vector.
- If fixed=TRUE, it will do exact matching for the phrase anywhere in the vector (regular find)

```
> grep("Rawlings", Sal$Name)  # These are the indices/elements where the pattern match occurs
```

```
[1] 10554 10555 10556
```

grep() returns something similar to which() on a logical statement

grep() as indices

```
> head(grep("Rawlings", Sal$Name))
```

```
[1] 10554 10555 10556
```

```
> head(grepl("Rawlings", Sal$Name))
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
> head(Rawlings <- Sal[grepl("Rawlings", Sal$Name), c("Name", "JobTitle")], 2)
```

```
                   Name             JobTitle
10554 Rawlings,Kellye A EMERGENCY DISPATCHER
10555  Rawlings,Paula M       COMMUNITY AIDE
```

grepl() returns something analagous to logical tests we covered yesterday.

Grep Options

```
> head(grep("Tajhgh", Sal$Name, value = TRUE))
```

```
[1] "Reynold,Tajhgh J"
```

```
> grep("Jaffe", Sal$Name)
```

```
integer(0)
```

```
> length(grep("Jaffe", Sal$Name))
```

```
[1] 0
```

A bit on Regular Expressions

- http://www.regular-expressions.info/reference.html
- They can use to match a large number of strings in one statement
- . matches any single character
- * means repeat as many (even if 0) more times the last character
- ? makes the last thing optional

Using Regular Expressions

- We will look for any instance that starts with:
    - Payne at the beginning,
    - Leonard and then an S
    - Spence then a capital C

```
> grep("Payne.*", x = Sal$Name, value = TRUE)
```

```
 [1] "Payne,Alexandra"          "Payne-Cooke,Shelley F"
 [3] "Payne,Denise I"           "Payne El,Jackie"
 [5] "Payne,James R"            "Payne,Jasman T"
 [7] "Payne Johnson,Nickole A" "Payne,Joseph"
 [9] "Payne,Karen V"            "Payne,Leonard S"
[11] "Payne,Marvin C"           "Payne,Mary A"
[13] "Payne,Micah W"            "Payne,Michael N"
[15] "Payne,Walter"             "Ray Payne,Marion J"
```

```
> grep("Leonard.?S", x = Sal$Name, value = TRUE)[1:5]
```

```
[1] "Payne,Leonard S"      "Szumlanski,Leonard S" NA
[4] NA                     NA
```

```
> grep("Spence.*C.*", x = Sal$Name, value = TRUE)
```

Replace

So we must change the annual pay into a numeric:

```
> head(as.numeric(Sal$AnnualSalary), 4)
```

```
[1] NA NA NA NA
```

R didn't like the $ so it thought turned them all to NA.

sub and gsub now do the replacing part.

Replacing and subbing

Now we can replace the $ with nothing (used fixed=TRUE because $ means something in regular expressions):

```
> Sal$AnnualSalary <- as.numeric(gsub(pattern = "$", replacement = "", Sal$AnnualSalary,
+     fixed = TRUE))
> Sal <- Sal[order(-Sal$AnnualSalary), ]  # use negative to sort descending
> head(Sal[, c("Name", "AnnualSalary", "JobTitle")])
```

```
                           Name AnnualSalary              JobTitle
881           Bernstein,Gregg L       238772      STATE'S ATTORNEY
734   Bealefeld III,Frederick H       193800   EXECUTIVE LEVEL III
4561         Gallagher,Edward J       181472   EXECUTIVE LEVEL III
589               Barbot,Oxiris       170000   EXECUTIVE LEVEL III
13920         Williams Jr,Henry       166400 CONTRACT SERV SPEC II
4384                Foxx,Alfred       160000 DIRECTOR PUBLIC WORKS
```

Useful String Functions

Useful String functions

- toupper(), tolower() - uppercase or lowercase your data:
- str_trim() (in the stringr package) - will trim whitespace
- nchar - get the number of characters in a string
- substr(x, start, stop) - substrings from position start to position stop
- strsplit(x, split) - splits strings up - returns list!
- paste() - paste strings together - look at ?paste

Paste

Paste can be very useful for joining vectors together:

```
> paste("Visit", 1:5, sep = "_")
```

```
[1] "Visit_1" "Visit_2" "Visit_3" "Visit_4" "Visit_5"
```

```
> paste("Visit", 1:5, sep = "_", collapse = " ")
```

```
[1] "Visit_1 Visit_2 Visit_3 Visit_4 Visit_5"
```

```
> paste("To", "is going be the ", "we go to the store!", sep = "day ")
```

```
[1] "Today is going be the day we go to the store!"
```

Writing your own functions

This is a brief introduction - we will cover more on Friday. The syntax is:

```
functionName = function(inputs) {
function body
return(value)
}
```

Then you would run the 4 lines of the code, which adds it to your workspace.

Writing your own functions

Here we will write a function that returns the second element of a vector:

```
> return2 = function(x) {
+     return(x[2])
+ }
> return2(c(1, 4, 5, 76))
```

```
[1] 4
```

Writing your own functions

Note that your function will automatically return the last line of code run:

```
> return2a = function(x) {
+     x[2]
+ }
> return2a(c(1, 4, 5, 76))
```

```
[1] 4
```

And if your function is really one line or evaluation, like here, you do not need the curly brackets, and you can put everything on one line:

```
> return2b = function(x) x[2]
> return2b(c(1, 4, 5, 76))
```

```
[1] 4
```

Strsplit

```
> x <- c("I really", "like writing", "R code")
> ss <- strsplit(x, split = " ")
> ss[[2]]
```

```
[1] "like"    "writing"
```

```
> sapply(ss, return2b)  # use your own function
```

```
[1] "really"  "writing" "code"
```

```
> sapply(ss, function(x) x[2])  # on the fly
```

```
[1] "really"  "writing" "code"
```

General comments on apply()

Apply functions are like 'for' loops. They 'go over' each element and perform a function on that element

Here, each element of the list 'ss' temporarily takes the value of 'x', and then evaluated.

```
> x = ss[[1]]
> x[2]
```

```
[1] "really"
```

```
> x = ss[[2]]
> x[2]
```

```
[1] "writing"
```

Data Merging/Append

- Merging - joining data sets together - usually on key variables, usually id
- merge is the most common way to do this with data sets
- rbind/cbind - row/column bind, respectively
    - rbind is the equivalent of "appending" in Stata or "setting" in SAS
    - cbind allows you to add columns in addition to the previous ways
- reshape2 package also has a lot of information about different ways to reshape data (wide to long, etc) - but has a different (and sometimes more intuitive syntax)
- t() is a function that will transpose the data

Merging

```
> base <- data.frame(id = 1:10, Age = rnorm(10, mean = 65, sd = 5))
> visits <- data.frame(id = rep(1:8, 3), visit = rep(1:3, 8), Outcome = rnorm(2 *
+     3, mean = 4, sd = 2))
> merged.data <- merge(base, visits, by = "id")
> table(merged.data$id)
```

```

1 2 3 4 5 6 7 8
3 3 3 3 3 3 3 3
```

```
> all.data <- merge(base, visits, by = "id", all = TRUE)
> table(all.data$id)
```

```

 1  2  3  4  5  6  7  8  9 10
 3  3  3  3  3  3  3  3  1  1
```

Problems with partial merges?

```
> all.data[all.data$id %in% c(9, 10), ]
```

```
   id   Age visit Outcome
25  9 58.47    NA      NA
26 10 73.50    NA      NA
```

Anything not merged is considered missing. No "Merge" variable is generated, but you can.

```
> base$base <- 1
> visits$visits <- 1
> all.data <- merge(base, visits, by = "id", all = TRUE)
> all.data[is.na(all.data$visits), ]
```

```
   id   Age base visit Outcome visits
25  9 58.47    1    NA      NA     NA
26 10 73.50    1    NA      NA     NA
```

Table data frames and merging

You can make summaries in Table then merge them

```
> tab <- table(Agency = Sal$Agency, useNA = "ifany")
> head(tab <- as.data.frame(tab, responseName = "N_Employees", stringsAsFactors = FALSE),
+     2)
```

```
         Agency N_Employees
1 Circuit Court         154
2  City Council          88
```

```
> Sal <- merge(Sal, tab, by = "Agency")
> head(Sal[, c("Name", "Agency", "N_Employees")], 2)
```

```
                 Name        Agency N_Employees
1 Elliott,Antoinella A Circuit Court         154
2      Hennigan,Mary L Circuit Court         154
```

Bind and t()

```
> head(all.data, 2)
```

```
  id   Age base visit Outcome visits
1  1 56.78    1     1   2.995      1
2  1 56.78    1     3   2.690      1
```

```
> head(t(all.data)[, 1:2])  # data is transposed
```

```
          [,1]  [,2]
id       1.000  1.00
Age     56.777 56.78
base     1.000  1.00
visit    1.000  3.00
Outcome  2.995  2.69
visits   1.000  1.00
```

```
> head(cbind(all.data, c("hey", "ho")))  #it will repeat to fill in the column
```

```
  id   Age base visit Outcome visits c("hey", "ho")
1  1 56.78    1     1  2.9950      1            hey
2  1 56.78    1     3  2.6904      1             ho
3  1 56.78    1     2  0.5518      1            hey
4  2 60.90    1     2  3.3902      1             ho
5  2 60.90    1     1  4.5470      1            hey
6  2 60.90    1     3  3.0208      1             ho
```

Side note about Binding

- R will wrap around elements to fill a column

```
> cbind(c(0, 1, 2), c(3, 4))
```

```
Warning: number of rows of result is not a multiple of vector length (arg
2)
```

```
     [,1] [,2]
[1,]    0    3
[2,]    1    4
[3,]    2    3
```

Side note about Binding

```
> cbind(c(0, 1, 2), c(3, 4, 5))
```

```
     [,1] [,2]
[1,]    0    3
[2,]    1    4
[3,]    2    5
```

```
> cbind(c(1:10), c(1:5))[3:7, ]
```

```
     [,1] [,2]
[1,]    3    3
[2,]    4    4
[3,]    5    5
[4,]    6    1
[5,]    7    2
```

Packages

Packages are add-ons that are commonly written by users comprised of functions, data, and vignettes

- Use library() or require() to load the package into memory so you can use its functions
- Install packages using install.packages("PackageName")
- Use help(package="PackageName") to see what contents the package has
- http://cran.r-project.org/web/packages/available_packages_by_name.html
- foreign package - read data from Stata/SPSS/SAS
- sas7bdat - read SAS data
- xlsx - reads in XLS files
- geepack - good for GEE analysis
- lme4 - linear/generalized linear mixed models
- survey - Survey data analysis (http://faculty.washington.edu/tlumley/survey/)

Data Reshaping

Disclaimer: the reshape command in R is not remarkably intuitive.

- Wide - multiple measurements are variables / columns so that the data gets wider with more measurements
- Long - multiple measurements are rows so data gets longer with more measurements
- One example would be many ids with multiple visits

Example of Long/Wide

```
> head(wide)
```

```
  id visit1 visit2 visit3
1  1   Good   Good    Bad
```

```
> head(long)
```

```
  id visit Outcome
1  1     1    Good
2  1     2    Good
3  1     3     Bad
```

Data Reshaping

- Good resource: http://www.ats.ucla.edu/stat/r/faq/reshape.htm

```
> times <- c("purple", "green", "orange", "banner")
> v.names <- c("Boardings", "Alightings", "Average")
> print(varying <- c(sapply(times, paste, sep = "", v.names)))
```

```
 [1] "purpleBoardings"  "purpleAlightings" "purpleAverage"
 [4] "greenBoardings"   "greenAlightings"  "greenAverage"
 [7] "orangeBoardings"  "orangeAlightings" "orangeAverage"
[10] "bannerBoardings"  "bannerAlightings" "bannerAverage"
```

Data Reshaping

```
> circ$date <- as.Date(circ$date, "%m/%d/%Y")  # creating a date for sorting
> ## important - varying, times, and v.names need to be in a correct order
> long <- reshape(data = circ, direction = "long", varying = varying, times = times,
+     v.names = v.names, timevar = "line", idvar = c("date"))
> rownames(long) <- NULL  # taking out row names
> long <- long[order(long$date), ]
> head(long)
```

```
         day       date daily   line Boardings Alightings Average
1     Monday 2010-01-11   952 purple        NA         NA      NA
1026  Monday 2010-01-11   952  green        NA         NA      NA
2051  Monday 2010-01-11   952 orange      1027        952     877
3076  Monday 2010-01-11   952 banner        NA         NA      NA
2    Tuesday 2010-01-12   796 purple        NA         NA      NA
1027 Tuesday 2010-01-12   796  green        NA         NA      NA
```

Data Reshaping

```
> dim(long)
```

```
[1] 4100    7
```

```
> long <- long[!is.na(long$Boardings) & !is.na(long$Alightings) & !is.na(long$Average),
+     ]
> dim(long)
```

```
[1] 2290    7
```

Data Reshaping

```
> head(long)
```

```
           day       date daily   line Boardings Alightings Average
2051    Monday 2010-01-11   952 orange      1027        952     877
2052   Tuesday 2010-01-12   796 orange       815        796     777
2053 Wednesday 2010-01-13  1212 orange      1220       1212    1203
2054  Thursday 2010-01-14  1214 orange      1233       1214    1194
2055    Friday 2010-01-15  1644 orange      1643       1644    1645
2056  Saturday 2010-01-16  1490 orange      1524       1490    1457
```

Data Reshaping

- If you've reshaped a data set - to get it back, just reshape it again

```
> head(reshape(long, direction = "wide"), 2)
```

```
         day       date daily purpleAlightings purpleAverage
2051  Monday 2010-01-11   952             1027           952
2052 Tuesday 2010-01-12   796              815           796
     purpleBoardings greenAlightings greenAverage greenBoardings
2051             877              NA           NA             NA
2052             777              NA           NA             NA
     orangeAlightings orangeAverage orangeBoardings bannerAlightings
2051               NA            NA              NA               NA
2052               NA            NA              NA               NA
     bannerAverage bannerBoardings
2051            NA              NA
2052            NA              NA
```

Data Reshaping - A Better Example

```
> library(xlsx, verbose = FALSE)
> TB <- read.xlsx(file = "~/Dropbox/WinterRClass/Datasets/indicator_estimatedincidencealltbper100000.xls
+     sheetName = "Data")
> head(TB, 1)
```

```
  TB.incidence..all.forms..per.population.per.year. X1990 X1991
1                                               Afghanistan   168   168
  X1992 X1993 X1994 X1995 X1996 X1997 X1998 X1999 X2000 X2001 X2002 X2003
1   168   168   168   168   168   168   168   168   168   168   168   168
  X2004 X2005 X2006 X2007 NA.
1   168   168   168   168  NA
```

```
> TB$NA. <- NULL
> head(TB, 1)
```

```
  TB.incidence..all.forms..per.population.per.year. X1990 X1991
1                                               Afghanistan   168   168
  X1992 X1993 X1994 X1995 X1996 X1997 X1998 X1999 X2000 X2001 X2002 X2003
1   168   168   168   168   168   168   168   168   168   168   168   168
  X2004 X2005 X2006 X2007
1   168   168   168   168
```

Data Reshaping - A Better Example

```
> colnames(TB) <- c("Country", paste("Year", 1990:2007, sep = "."))
> head(TB, 1)
```

```
      Country Year.1990 Year.1991 Year.1992 Year.1993 Year.1994 Year.1995
1 Afghanistan       168       168       168       168       168       168
  Year.1996 Year.1997 Year.1998 Year.1999 Year.2000 Year.2001 Year.2002
1       168       168       168       168       168       168       168
  Year.2003 Year.2004 Year.2005 Year.2006 Year.2007
1       168       168       168       168       168
```

Data Reshaping - More is better!

```
> TB.long <- reshape(TB, idvar = "Country", v.names = "Cases", times = 1990:2007,
+     direction = "long", timevar = "Year", varying = paste("Year", 1990:2007,
+         sep = "."))
>
> head(TB.long, 4)
```

```
                           Country Year Cases
Afghanistan.1990       Afghanistan 1990   168
Albania.1990               Albania 1990    25
Algeria.1990               Algeria 1990    38
American Samoa.1990 American Samoa 1990    21
```

```
> rownames(TB.long) <- NULL
> head(TB.long, 4)
```

```
         Country Year Cases
1    Afghanistan 1990   168
2        Albania 1990    25
3        Algeria 1990    38
4 American Samoa 1990    21
```

Data Reshaping - A common "bug?"

```
> TB.long2 <- reshape(TB, idvar = "Country", direction = "long", timevar = "Year",
+     varying = paste("Year", 1990:2007, sep = "."))
> head(TB.long2, 3)  ### what happened?
```

```
                     Country Year
Afghanistan.1990 Afghanistan  168
Albania.1990         Albania   25
Algeria.1990         Algeria   38
```

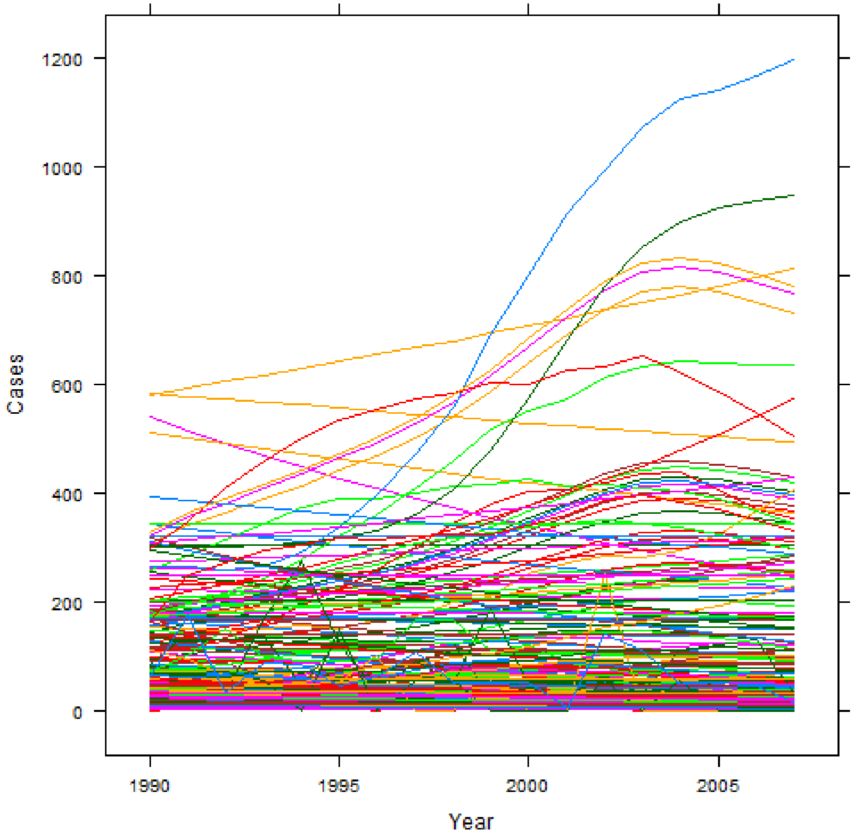
```
> TB.long2 <- reshape(TB, idvar = "Country", direction = "long", timevar = "Blah",
+     varying = paste("Year", 1990:2007, sep = "."))
> head(TB.long2, 3)  ## Timevar can't be the stub of the original variable
```

```
                     Country Blah Year
Afghanistan.1990 Afghanistan 1990  168
Albania.1990         Albania 1990   25
Algeria.1990         Algeria 1990   38
```

Reshaped - let's plot some Spaghetti

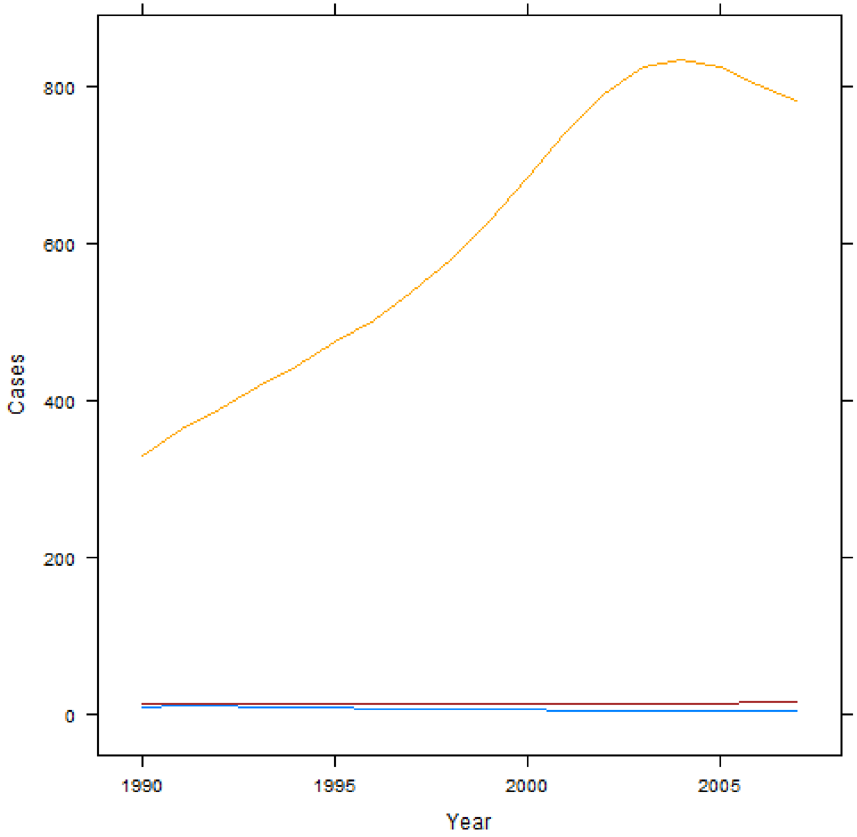
- Spaghetti or "line" plots are relatively easy using the lattice package in R

```
> library(lattice)
> xyplot(Cases ~ Year, groups = Country, data = TB.long, type = "l")
```



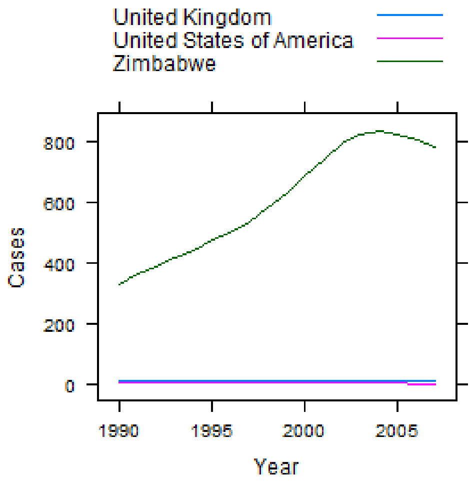
More Spaghetti

```
> ## Only keep a few countries
> xyplot(Cases ~ Year, groups = Country, data = TB.long, subset = Country %in%
+     c("United States of America", "United Kingdom", "Zimbabwe"), type = "l")
```



More Spaghetti

```
> ## plot things 'by' Country xyplot(Cases ~ Year | Country, data=TB.long,
> ## subset=Country %in% c('United States of America', 'United Kingdom',
> ## 'Zimbabwe'), type='l')
> TBC <- TB.long[TB.long$Country %in% c("United States of America", "United Kingdom",
+     "Zimbabwe"), ]
> TBC$Country <- factor(TBC$Country)
> xyplot(Cases ~ Year, groups = Country, data = TBC, type = "l", key = simpleKey(levels(TBC$Country),
+     lines = TRUE, points = FALSE))
```



Reshaping Wide

```
> head(Indometh, 2)
```

```
  Subject time conc
1       1 0.25 1.50
2       1 0.50 0.94
```

```
> wide <- reshape(Indometh, v.names = "conc", idvar = "Subject", timevar = "time",
+     direction = "wide")
> head(Indometh, 2)
```

```
  Subject time conc
1       1 0.25 1.50
2       1 0.50 0.94
```

Lab

Salaries data:

1. Make an object called health.sal using the salaries data set, with only agencies of those with "fire" (or any forms), if any, in the name
2. Make a data set called trans which contains only agencies that contain "TRANS".
3. What is/are the profession(s) of people who have "abra" in their name for Baltimore's Salaries?

Restaurants data:

1. Reshape the restaurants data set to wide, on council district. You may need to create an id variable by the code: `rest$id <- 1:nrow(rest)`

Monuments data:

1. How many monuments contain the phrase "Monument" in them?