Agenda

1. Data Wrangling

```
require(mosaic)
require(Stat2Data)
```

Data Wrangling Data wrangling is a somewhat difficult to define, but highly valuable skill. As you have probably already seen, real-world data analysis projects often involve a significant amount of work just to get the data into a form suitable for analysis. The greater your capacity for accomplishing these tasks, the more quickly and more efficiently you can whip your data into shape, leaving more time for careful analysis.

People are starting to pay more attention to these skills, and are starting to think about data wrangling as a field of expertise unto itself. Chief among these people is Hadley Wickham of Rice University and RStudio, author of the `dplyr` and `tidyr` packages for R. RStudio has also recently released a Data Wrangling Cheatsheet illustrating many common operations.

Consider the following data manipulation *verbs*:

1. `mutate`: add or change columns
2. `select`: take a subset of the columns
3. `filter`: take a subset of the rows
4. `arrange`: sort the rows
5. `summarise` (with `group_by`): aggregate the rows

These five relatively simple ideas can be combined to enable powerful analyses. The use of the chaining operator (`%>%`) makes this particularly elegant.

```
require(nycflights13)
```

1. Count the total number of flights in the `flights` table.
2. Find all the airports that are destinations. Count how many flights went into and out of each airport?
3. Over what time span does this data cover?
4. Find all the flights originating from LaGuardia airport on April 8th, 2013. How many were there? Show only carriers and flight numbers, and sort them by arrival delay.
5. What was the average delay? By airline? By airport? If you had to fly from LaGuardia, JFK, or Newark, which airport would you choose?

Linking Two Tables Two tables can be joined together using a *key* via the `inner_join` function.

1. Are flights that arrive at mile-high altitude (at least 5,280 feet) more or less likely to be on time?

Reshaping One popular source of data is Gapminder [], the brainchild of Hans Rosling. Gapminder contains data about countries over time for a variety of different variables – including the prevalence of HIV among adults aged 15 to 49 – and other health and economic indicators. These data are stored in Google Spreadsheets, or one can download them as Microsoft Excel workbooks. The typical presentation of a small subset of such data is shown below.

```
hiv.key <- "pyj6tScZqmEfbZyl0qjbiRQ"
hiv <- fetchGoogle(key = hiv.key)
names(hiv)[1] <- "Country"
hiv %>%
  filter(Country %in% c("United States", "France", "South Africa")) %>%
  select(Country, X1979, X1989, X1999, X2009)

##         Country      X1979 X1989 X1999 X2009
## 1        France         NA    NA   0.3   0.4
## 2  South Africa         NA    NA  14.8  17.2
## 3 United States 0.03176408    NA   0.5   0.6
```

Our data takes the form of a two-dimensional array, where each of the n rows represents a country, and each of the p years is a column. Each entry represents the percentage of adults aged 15 to 49 living with HIV in the i^{th} country in the j^{th} year. This presentation of the data has some advantages. First, it is possible (with a big enough monitor) to *see* all of the data. One can quickly follow the trend over time for a particular country, and one can also estimate quite easily the percentage of missing data present. Thus, if visual inspection is the primary analytical technique, this *spreadsheet*-style presentation can be convenient.

Alternatively, consider this presentation of that same data:

```
require(tidyr)
hivLong <- gather(hiv, key = Year, value = hiv.rate, -Country)
hivLong <- mutate(hivLong, Year = as.numeric(gsub("X", "", Year)))
hivLong %>%
  filter(Country %in% c("United States", "France", "South Africa")) %>%
  filter(Year %in% c(1979, 1989, 1999, 2009))

##          Country Year    hiv.rate
## 1         France 1979          NA
## 2   South Africa 1979          NA
## 3  United States 1979  0.03176408
## 4         France 1989          NA
## 5   South Africa 1989          NA
## 6  United States 1989          NA
## 7         France 1999  0.30000000
## 8   South Africa 1999 14.80000000
## 9  United States 1999  0.50000000
## 10        France 2009  0.40000000
## 11  South Africa 2009 17.20000000
## 12 United States 2009  0.60000000
```

While our data is still a two-dimensional array, it now has np rows and just three columns. Visual inspection of the data is now more difficult, since our data is now very long and very narrow – it's aspect ratio is not similar to that of our screen!

It turns out that there are substantive reasons to prefer the long, narrow version of this data. It is a more efficient way for the computer to store and retrieve the data. It is more convenient for the purpose of data analysis. And it is more scalable, in that the addition of a second variable simply contributes another column, whereas to add another variable to the spreadsheet presentation would require a confusing three-dimensional view (or worse, merged cells).

These gains come at a cost: we have relinquished our ability to *see all the data at once*. When data is small, being able to see it all at once can be useful, and even comforting. But in this era of big data, a reliance upon seeing all the data at once in a spreadsheet layout is a fool's errand. Learning to manipulate data via programming frees us from the click-and-drag paradigm popularized by spreadsheet applications, and allows us to work with data of arbitrary size. Recording our data

manipulation operations in code also makes them reproducible, an increasingly desirable trait in this era of collaboration. It enables us to fully separate the raw data from our analysis, two concepts too often elided in the spreadsheet application.