Stats workshop

Old Section

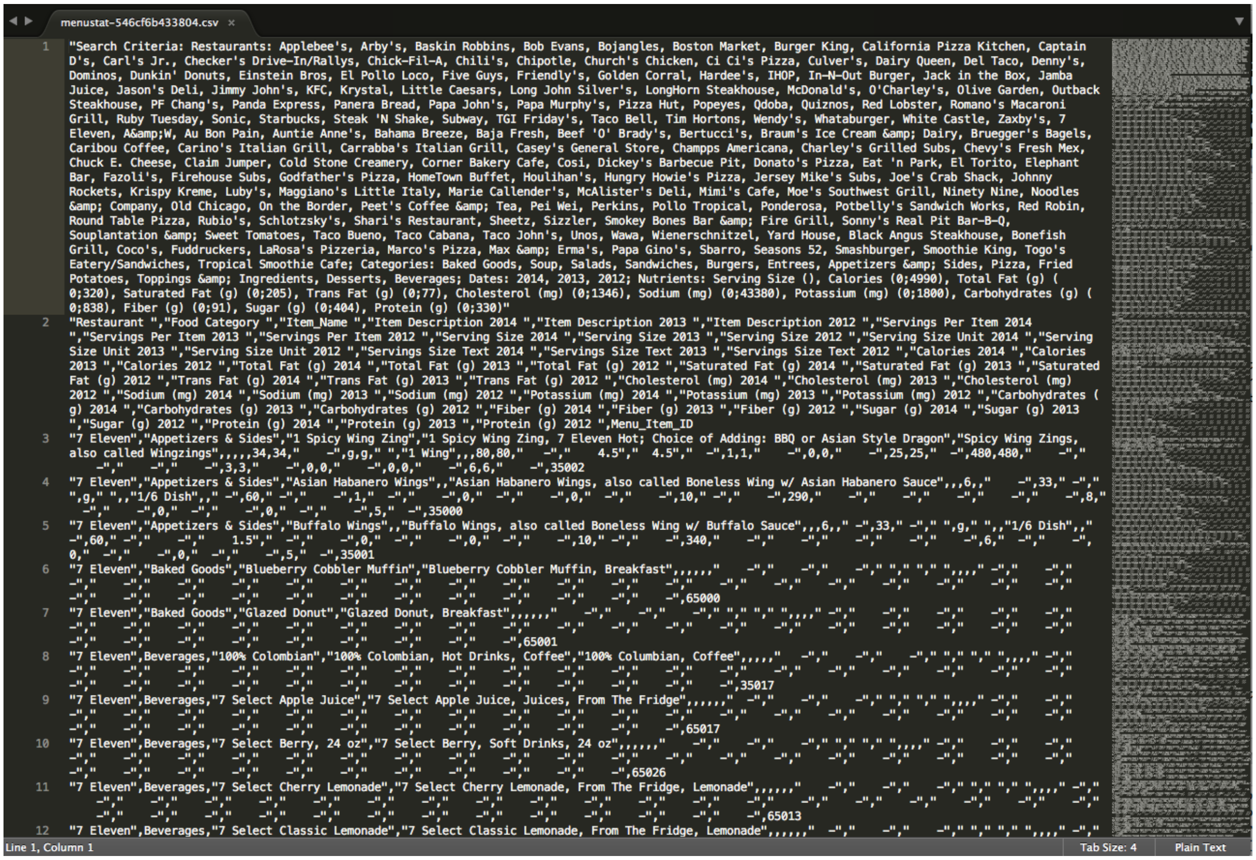
I resisted deleting this. It isn't necessary for what I'm under the impression ya'll want. But maybe you'll still find it interesting or fun. I've commented out all the commands because some of them take a while to run. If you want to run it yourself, just remove the comment tags (#) on all lines with the exception of the first code chunk. The first one results in an error and will choke knitr. If you don't care, you can skip down to the part that says "Mark's Data"

I've put together a document which should guide the work we're doing in our stats workshop. This is an applied statistics approach in which we will emphasize the day-to-day work of data analysis. First, we will need some data! Where do we get it? The typical approach in academic psychology is to run some kind of study. This isn't true for every discipline, and arguably, our own discipline doesn't take enough advantage of the data which can be found free for the taking in the real world. For example, I took the following data from menustat.org. Let's try to read it in.

```
#menus<-read.csv('menustat-546cf6b433804.csv')
#this doesn't work.  Throws an error:

#Error in read.table(file = file, header = header, sep = sep, quote = quote,  :
#  more columns than column names
```

Aw damn. Well, that didn't work. Why not? The error message says that there's a mismatch between the number of columns, and the number of names it has for the columns. Usually, this function reads the column names from the first line of the csv document, so I'm guessing that something is wrong with the first line of the csv. Opening it up in a text editor shows the following:



See the big chunk of text next to the 1 on the lefthand side? That's all the stuff that appears in the first line, and we can see that it's just kind of a description of the file. We can get rid of that. Line 2 seems to have the columns we want. So we're going to take a slightly different approach:

1. Read the file, line by line as one big vector
2. Remove the first line.
3. Turn the vector into a dataframe.

```
#menus<-readLines('menustat-546cf6b433804.csv')
#menus<-menus[-1]
#menus<-read.csv(textConnection(menus)) #this takes a minute...
```

Now we have a datafile for our menus data! This is a new datafile for all of us, so let's just get some information about it.

```
#dim(menus)
```

Serious data, ya'll. We've got 60,238 observations across 52 different variables. Let's get a little information about those variables.

```
#names(menus)
```

We can now see the names for our variables. Looks like there's one for restaurant, one for food category, one for the item, and then we have a bunch of variables which are repeated for each of 3 years: 2014, 2013, 2012. These variables describe the item, give us a serving size, and then the nutrient information (e.g. calories, fat, carbs, protein, etc.).

Mark's Data

This wont just be a *lesson* about what to do when one has new data. This will be an actual example of what I would do when I have new data. Mark has sent me some of his data and I haven't done anything but download it yet. So, this is actually me playing around with new data. Let's see what I'm gonna do (even I don't know!). =)

A quick digression on organization This doesn't matter much at first, but once you have multiple projects, and are collaborating with multiple people, and these projects drag out over $_{\text{months}}$ years, organization can really become difficult to maintain. There are ways to keep organized, however! I recently found a nice, short, readable guide on all the stuff I've spent a while figuring out on my own. Some of what you will find in there you will find more or less relevant (check out chapters 3 and 4 in particular), but I'll emphasize a couple of things:

1. Use version control. Github is easy to use, especially when you download their gui. You'll never have to worry about having multiple, conflicting, unclear analysis files again for fear of losing something you wrote previously.

2. Write your analysis code so that you can run the whole thing from start to finish. It helps even more if you write it in markdown (or something similar) with prose explaining what you're doing and what you're thinking. From there, it's only a short step to turn it into a methods/results section.

Of course, I should put caveats all around this thing because I often don't really know what I'm doing either. I've never had a data set where I was doing all the same old things. It's a constant process of learning something new. And while I've been fighting the organization war for years, it's only in the last 6 months or so that I've really decided that this was something I had to win. =)

Get the data in and look at what we've got

First, let's get it in. I'm sure you're all familiar with how to do this. However, it helps to keep things organized. Everything related to this project should all be in the same folder. I've put Mark's data in the same folder I'm keeping this script. Before doing anything, you should click Session -> Set Working Directory -> To Source File Location.

I wish there were an automatic way to do that. Unfortunately, I don't think there is. Anyway, once that's done, we can move on to read in his data and get the dimensions.

```
breach <- read.csv('breachmeans.csv')
dim(breach)
```

```
## [1] 20 32
```

Okay! We've got 20 observations for each of 32 variables. Let's look a little more closely at what those variables are. Next, I'm going to get the names of each variable:

```
names(breach)
```

```
##  [1] "subj"           "locomotion1"    "locomotion2"    "difflocomotion"
##  [5] "assessment1"    "assessment2"    "diffassessment" "promotion1"
##  [9] "promotion2"     "diffpromotion"  "prevention1"    "prevention2"
## [13] "diffprevention" "stress1"        "stress2"        "diffstress"
```

```
## [17] "flourishing"    "swls"           "lie2"           "lie1"
## [21] "diffLie"        "LxA1"           "LxA2"           "diffLxA"
## [25] "age"            "exp"            "apride"         "lpride"
## [29] "eom"            "rankindiv"      "rank21"         "averank"
```

Now, a bit of a closer look at what these things contain:

```
head(breach)
```

```
##   subj locomotion1 locomotion2 difflocomotion assessment1 assessment2
## 1    1        5.00        4.58        -0.4167        4.17        3.67
## 2    2        4.50        4.58         0.0833        4.25        3.92
## 3    3        5.25        4.83        -0.4167        4.00        4.00
## 4    4        4.83        5.00         0.1667        4.00        3.83
## 5    5        5.08        4.92        -0.1667        4.50        3.83
## 6    6        5.08        4.75        -0.3333        5.00        4.08
##   diffassessment promotion1 promotion2 diffpromotion prevention1
## 1        -0.5000       3.50       4.00         0.500         1.8
## 2        -0.3333       4.00       4.17         0.167         2.2
## 3         0.0000       4.17       3.83        -0.333         1.0
## 4        -0.1667       4.00       4.50         0.500         2.2
## 5        -0.6667       4.83       4.83         0.000         3.4
## 6        -0.9167       4.00       4.33         0.333         2.4
##   prevention2 diffprevention stress1 stress2 diffstress flourishing swls
## 1         2.2            0.4    1.62    1.50     -0.125        6.50  6.0
## 2         2.2            0.0    2.00    2.50      0.500        5.25  5.8
## 3         1.4            0.4    2.38    2.25     -0.125        6.25  6.2
## 4         2.0           -0.2    2.00    2.62      0.625        6.88  6.8
## 5         3.0           -0.4    2.00    2.00      0.000        6.25  6.2
## 6         2.2           -0.2    1.75    2.12      0.375        6.00  5.6
##   lie2 lie1 diffLie LxA1 LxA2 diffLxA age exp apride lpride eom rankindiv
## 1 1.67 1.83  -0.167 25.0 21.0  -3.993  23   3     37     46  40        10
## 2 2.67 2.33   0.333 20.2 21.0   0.757  26   6     38     38  35         4
## 3 2.00 2.67  -0.667 27.6 23.4  -4.201  22   4     32     40  43         6
## 4 2.33 3.00  -0.667 23.4 25.0   1.639  26   2     40     46  42        16
## 5 1.83 1.83   0.000 25.8 24.2  -1.667  26   4     45     44  38         3
## 6 3.00 3.17  -0.167 25.8 22.6  -3.278  29   3     38     44  34         5
##   rank21 averank
## 1   13.0   11.50
## 2    3.5    3.75
## 3    5.0    5.50
## 4   16.0   16.00
## 5    3.5    3.25
## 6    7.0    6.00
```

```
summary(breach)
```

```
##       subj        locomotion1     locomotion2    difflocomotion
##  Min.   : 1.00   Min.   :3.830   Min.   :3.580   Min.   :-1.7500
##  1st Qu.: 5.75   1st Qu.:4.647   1st Qu.:4.560   1st Qu.:-0.3542
##  Median :10.50   Median :5.000   Median :4.920   Median :-0.2084
##  Mean   :10.50   Mean   :4.949   Mean   :4.816   Mean   :-0.1333
##  3rd Qu.:15.25   3rd Qu.:5.122   3rd Qu.:5.270   3rd Qu.: 0.2708
```

```
##  Max.   :20.00   Max.   :5.750   Max.   :5.420   Max.   : 0.5000
##
##   assessment1     assessment2    diffassessment      promotion1
##  Min.   :3.750   Min.   :3.500   Min.   :-1.3333   Min.   :3.500
##  1st Qu.:4.000   1st Qu.:3.830   1st Qu.:-0.5417   1st Qu.:3.830
##  Median :4.290   Median :3.920   Median :-0.3333   Median :4.000
##  Mean   :4.351   Mean   :4.017   Mean   :-0.3333   Mean   :4.058
##  3rd Qu.:4.543   3rd Qu.:4.122   3rd Qu.:-0.1459   3rd Qu.:4.210
##  Max.   :5.170   Max.   :4.920   Max.   : 1.1667   Max.   :4.830
##
##    promotion2    diffpromotion       prevention1    prevention2
##  Min.   :3.170   Min.   :-0.83300   Min.   :1.00   Min.   :1.4
##  1st Qu.:3.830   1st Qu.:-0.33300   1st Qu.:2.20   1st Qu.:2.2
##  Median :4.000   Median : 0.08350   Median :2.60   Median :2.7
##  Mean   :4.041   Mean   :-0.01665   Mean   :2.53   Mean   :2.6
##  3rd Qu.:4.330   3rd Qu.: 0.33300   3rd Qu.:3.00   3rd Qu.:3.0
##  Max.   :4.830   Max.   : 0.50000   Max.   :3.80   Max.   :3.4
##
##  diffprevention     stress1         stress2        diffstress
##  Min.   :-0.80   Min.   :1.380   Min.   :1.000   Min.   :-0.75000
##  1st Qu.:-0.25   1st Qu.:1.685   1st Qu.:1.718   1st Qu.:-0.12500
##  Median : 0.20   Median :1.880   Median :2.000   Median : 0.00000
##  Mean   : 0.07   Mean   :2.013   Mean   :2.068   Mean   : 0.08553
##  3rd Qu.: 0.40   3rd Qu.:2.250   3rd Qu.:2.530   3rd Qu.: 0.37500
##  Max.   : 0.80   Max.   :3.500   Max.   :3.120   Max.   : 0.62500
##                  NA's   :1                       NA's   :1
##   flourishing         swls           lie2            lie1
##  Min.   :4.880   Min.   :4.40   Min.   :1.000   Min.   :1.000
##  1st Qu.:5.750   1st Qu.:5.75   1st Qu.:2.290   1st Qu.:2.290
##  Median :6.000   Median :5.80   Median :2.670   Median :2.500
##  Mean   :5.976   Mean   :5.91   Mean   :2.675   Mean   :2.632
##  3rd Qu.:6.250   3rd Qu.:6.20   3rd Qu.:3.042   3rd Qu.:3.000
##  Max.   :7.000   Max.   :7.00   Max.   :4.670   Max.   :4.500
##
##     diffLie             LxA1            LxA2          diffLxA
##  Min.   :-0.6670   Min.   :14.70   Min.   :12.80   Min.   :-16.479
##  1st Qu.:-0.2085   1st Qu.:21.60   1st Qu.:20.80   1st Qu.: -3.790
##  Median : 0.0000   Median :25.00   Median :24.20   Median : -1.760
##  Mean   : 0.0416   Mean   :24.72   Mean   :23.45   Mean   : -1.258
##  3rd Qu.: 0.2085   3rd Qu.:26.25   3rd Qu.:27.80   3rd Qu.:  2.675
##  Max.   : 1.0000   Max.   :33.10   Max.   :29.30   Max.   :  5.167
##
##       age            exp            apride          lpride
##  Min.   :21.0   Min.   :0.00   Min.   :28.00   Min.   :29.00
##  1st Qu.:24.0   1st Qu.:2.00   1st Qu.:36.00   1st Qu.:38.00
##  Median :26.0   Median :3.00   Median :38.00   Median :41.50
##  Mean   :25.9   Mean   :3.25   Mean   :37.45   Mean   :41.35
##  3rd Qu.:28.0   3rd Qu.:4.00   3rd Qu.:39.00   3rd Qu.:46.00
##  Max.   :31.0   Max.   :8.00   Max.   :46.00   Max.   :49.00
##
##       eom          rankindiv         rank21         averank
##  Min.   :34.00   Min.   : 1.00   Min.   : 1.00   Min.   : 1.500
##  1st Qu.:37.00   1st Qu.: 5.75   1st Qu.: 5.75   1st Qu.: 5.875
##  Median :38.50   Median :10.50   Median :11.00   Median :10.500
```

```
##  Mean   :38.45   Mean   :10.50   Mean   :11.00   Mean   :10.750
##  3rd Qu.:40.00   3rd Qu.:15.25   3rd Qu.:16.25   3rd Qu.:16.500
##  Max.   :43.00   Max.   :20.00   Max.   :21.00   Max.   :20.000
##
```

These are quick ways to examine what your dataframe looks like. `head` will return the first 5 rows of the dataframe (you can also use `tail` to get the last 5 rows). `summary`, when called on a dataframe, will return a description of each variable. Mark's dataframe is a little unusual for psych studies in that it doesn't seem to have any factor variables. We can see this in another way by asking for the structure, using the `str` command.

```
str(breach)
```

```
## 'data.frame':    20 obs. of  32 variables:
##  $ subj          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ locomotion1   : num  5 4.5 5.25 4.83 5.08 5.08 4.67 5 5.58 5.08 ...
##  $ locomotion2   : num  4.58 4.58 4.83 5 4.92 4.75 5.08 4.33 5.25 5.33 ...
##  $ difflocomotion: num  -0.4167 0.0833 -0.4167 0.1667 -0.1667 ...
##  $ assessment1   : num  4.17 4.25 4 4 4.5 5 4 4.5 4.42 4.25 ...
##  $ assessment2   : num  3.67 3.92 4 3.83 3.83 4.08 3.5 4 3.67 3.92 ...
##  $ diffassessment: num  -0.5 -0.333 0 -0.167 -0.667 ...
##  $ promotion1    : num  3.5 4 4.17 4 4.83 4 3.83 3.83 4 4.67 ...
##  $ promotion2    : num  4 4.17 3.83 4.5 4.83 4.33 4.17 3.5 3.83 4 ...
##  $ diffpromotion : num  0.5 0.167 -0.333 0.5 0 0.333 0.333 -0.333 -0.167 -0.667 ...
##  $ prevention1   : num  1.8 2.2 1 2.2 3.4 2.4 2.8 3 2.8 2.6 ...
##  $ prevention2   : num  2.2 2.2 1.4 2 3 2.2 3 2.8 3.2 2.8 ...
##  $ diffprevention: num  0.4 0 0.4 -0.2 -0.4 -0.2 0.2 -0.2 0.4 0.2 ...
##  $ stress1       : num  1.62 2 2.38 2 2 1.75 3.5 2.25 1.75 1.75 ...
##  $ stress2       : num  1.5 2.5 2.25 2.62 2 2.12 3.12 2.88 1.75 1.75 ...
##  $ diffstress    : num  -0.125 0.5 -0.125 0.625 0 0.375 -0.375 0.625 0 0 ...
##  $ flourishing   : num  6.5 5.25 6.25 6.88 6.25 6 7 6 5.75 6 ...
##  $ swls          : num  6 5.8 6.2 6.8 6.2 5.6 7 5.8 5.8 5.8 ...
##  $ lie2          : num  1.67 2.67 2 2.33 1.83 3 4.67 2.33 1 3.17 ...
##  $ lie1          : num  1.83 2.33 2.67 3 1.83 3.17 3.83 2.33 1 2.83 ...
##  $ diffLie       : num  -0.167 0.333 -0.667 -0.667 0 -0.167 0.833 0 0 0.333 ...
##  $ LxA1          : num  25 20.2 27.6 23.4 25.8 25.8 21.8 25 31.2 25.8 ...
##  $ LxA2          : num  21 21 23.4 25 24.2 22.6 25.8 18.8 27.6 28.4 ...
##  $ diffLxA       : num  -3.993 0.757 -4.201 1.639 -1.667 ...
##  $ age           : int  23 26 22 26 26 29 28 24 27 22 ...
##  $ exp           : int  3 6 4 2 4 3 0 3 4 1 ...
##  $ apride        : int  37 38 32 40 45 38 38 33 36 37 ...
##  $ lpride        : int  46 38 40 46 44 44 49 40 38 38 ...
##  $ eom           : int  40 35 43 42 38 34 39 41 38 37 ...
##  $ rankindiv     : int  10 4 6 16 3 5 12 9 11 19 ...
##  $ rank21        : num  13 3.5 5 16 3.5 7 21 10 8 15 ...
##  $ averank       : num  11.5 3.75 5.5 16 3.25 6 16.5 9.5 9.5 17 ...
```

This highlights that within R, the dataframe is really just a collection of vectors. Mark has two types: `numeric` and `integer`.

Question: 1. We wont always be fortunate enough to have .csv files. How can we read in excel files? Spss? STATA? Word?

```
library(xlsx)
read.xlsx()
library(foreign)
read.spss()
read.dta()
```

Microsoft word is possible (everything is *possible*), but a bit more complicated (see `tm` package)

Data types

From an R standpoint, there are few differences between `numeric` and `integer` types. It's good to know they exist, because the type of object you pass to functions can be a major source of headaches (e.g. if a function expects a `factor` and you try to pass it a `numeric`), but for now, we can generally treat `numeric` and `integers` as the same thing - quantitative variables. This can be contrasted with two other types that we do not see here: `factors` and `characters`. A factor is a categorical variable, and a character is just a collection of text strings. Let's go ahead and make a couple of these. First, we'll make a variable indicating whether someone is happy or not:

```
breach$happy <- rep(c("yes", "no"), each=10)
breach$happy #there's only 20 observations, so we can just print out all of them.
```

```
##  [1] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "no"
## [12] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
```

```
str(breach$happy)
```

```
##  chr [1:20] "yes" "yes" "yes" "yes" "yes" "yes" "yes" ...
```

```
summary(breach$happy)
```

```
##    Length     Class      Mode
##        20 character character
```

Ah! A `character` variable! But that's just because we fed it some text strings, isn't it? As psychologists, we know that this is actually a categorical variable. People fall into one of two categories: Those who are happy, and those who have failed research projects. ;)

Anyway, since this is a categorical variable, we want this to be a `factor`.

```
breach$happy <- as.factor(breach$happy)
breach$happy
```

```
##  [1] yes yes yes yes yes yes yes yes yes yes no  no  no  no  no  no  no
## [18] no  no  no
## Levels: no yes
```

```
str(breach$happy)
```

```
##  Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
```

```
summary(breach$happy)
```

```
##  no yes
##  10  10
```

You can see that calling some of the same functions on data which is identical in appearance but *stored* in two different ways can lead to some very different output. It's key to keep this in mind no matter what you're doing in R. If something doesn't work the way you expect it to, maybe it's because you're feeding in an unexpected data type. I think I spent about a month in my third year figuring this out. A long, painful month.

Questions:

1. Let's say Mark ran a study in which his participants recieved 2, 4, or 8 'happy' primes. We create a factor variable which distinguishes these groups:

   `a <- as.factor(rep(c(2,4,8), each=10))`

   For some purposes, this will be fine. But let's say for some reason we want to treat this as a numeric value rather than a factor. We run the following command:

   `a <- as.numeric(a)`

   What happened? Did it work as you expected?

2. How are factor variables represented internally in R (*hint*: Why, when we call `str(breach$happy)`, do we get a bunch of numbers)? What's the logic behind their representation?

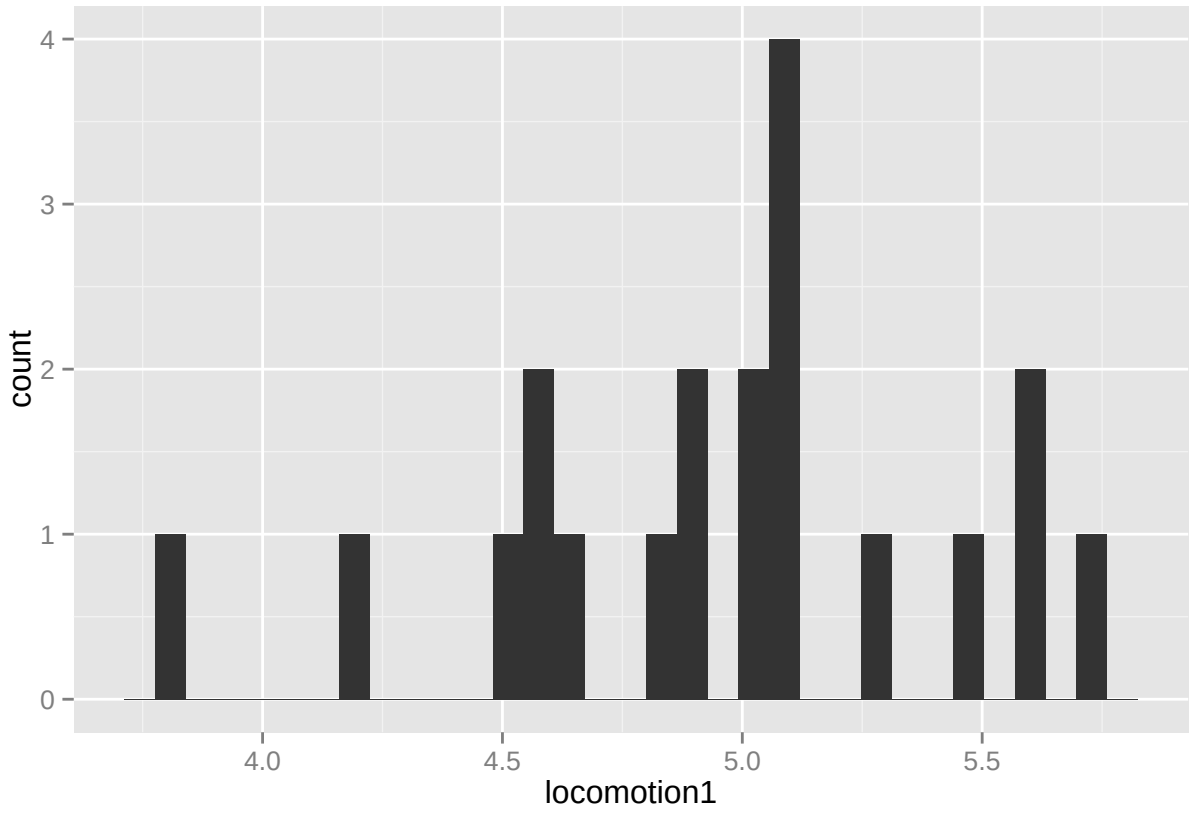
Now that we know a little bit about data types, let's think of what types of things we might be interested in looking at. It's time to know a bit about `ggplot`, don't you think?

Preliminary visualizations

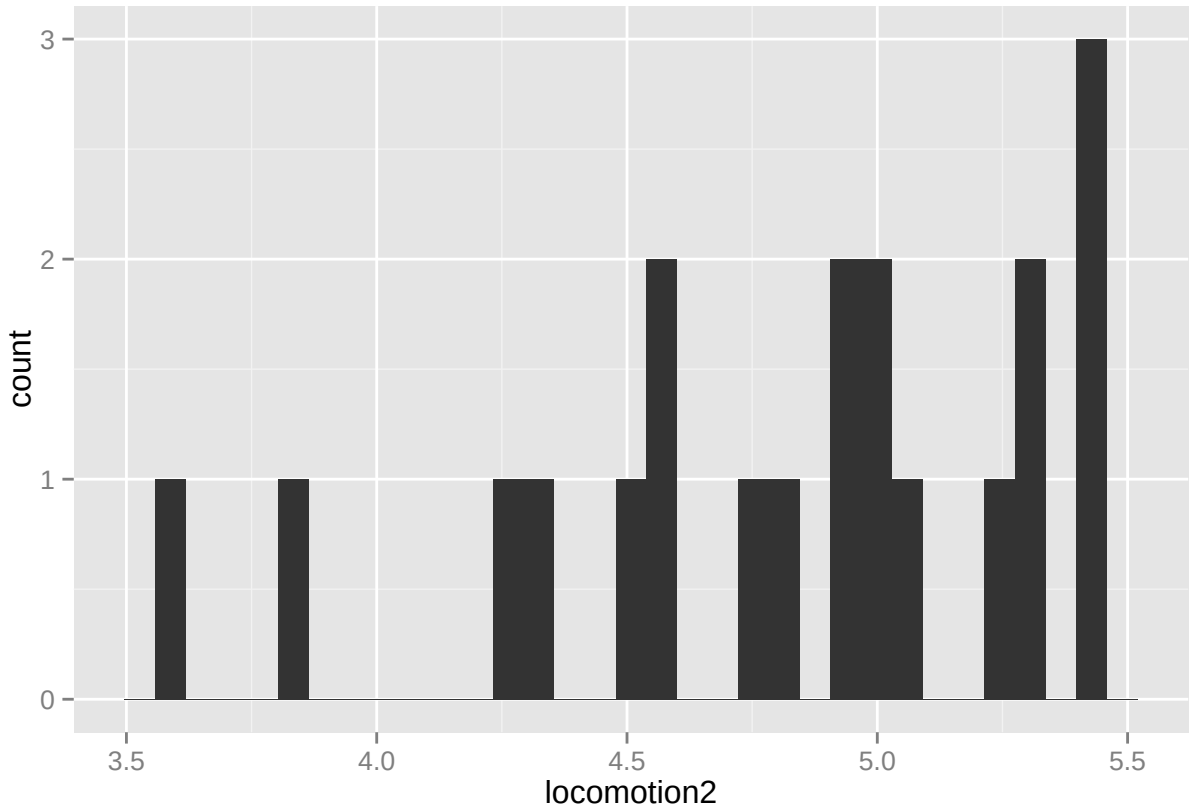
ggplot is a package built by the incredibly productive Hadley Wickham. It follows a general plotting philosophy published and described elsewhere called *Grammer of Graphics*. To create a ggplot object, you must first decide what you're plotting. Since I see that we have two scores for most of our measurements (plus I've spoken to Mark about this work a few times), I'm going to treat these as two measurements of the same concept on the same individual. One thing we obviously might be interested in is how these measurements changed between session one and session two.

Please note that there's no hypothesis testing going on yet! We're just looking at data! You should spend quite a lot of time doing this. Anyway, on with it. Let's make a couple of histograms for the locomotion variable.

```
library(ggplot2)
plot <- ggplot(breach, aes(x=locomotion1))
plot + geom_histogram()
```




```
plot <- ggplot(breach, aes(x=locomotion2))
plot + geom_histogram()
```



Before we pull apart the figures, let's look at what we've done here. For each plot, there are two lines of code. The first line from the first plot:

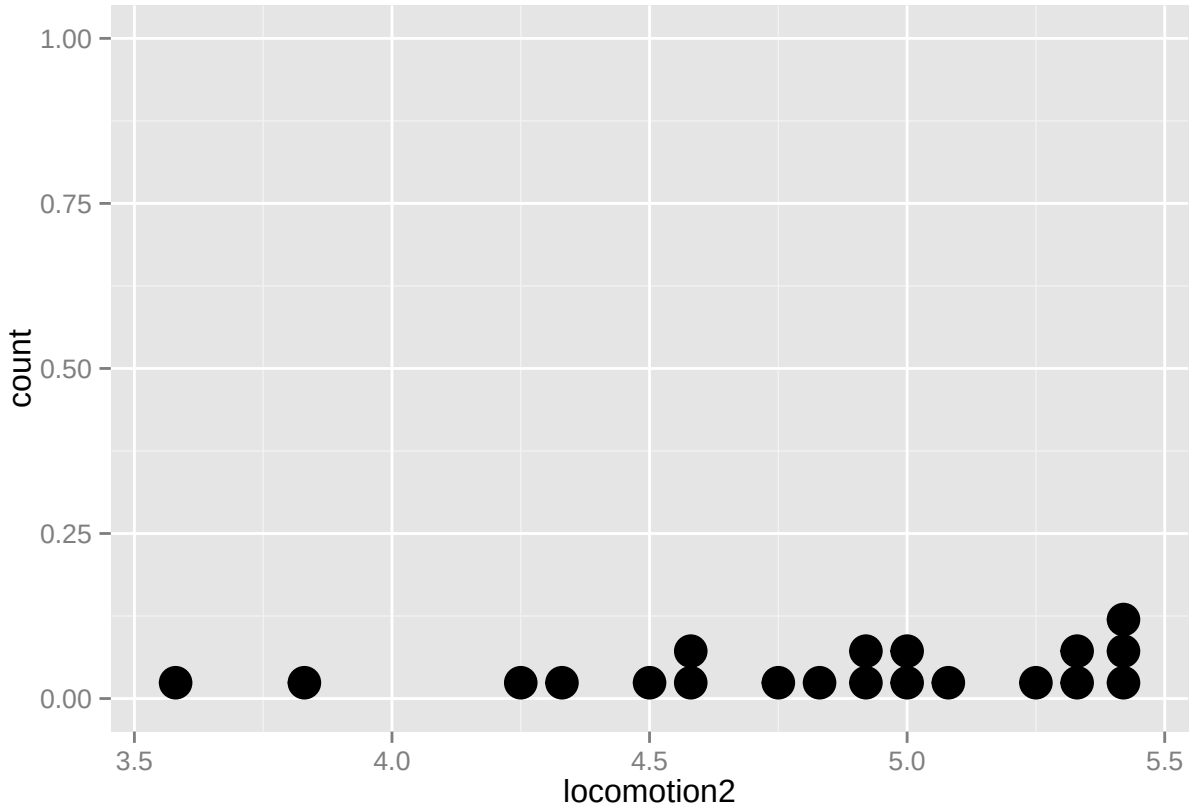
```
plot <- ggplot(breach, aes(x=locomotion1))
```

This makes an object we've called `plot`. This object is stored internally as a list, but we don't really care about that. What we care about is that this is a ggplot object made from the `breach` dataframe (1st argument). We've also assigned some aesthetics (via `aes`) for this particular plot. In particular, we've told it that we want the variable `locomotion1` to appear on the x axis (2nd argument). This will create an empty plot, onto which we can begin placing layers. Each layer added to the plot will do something to change the superficial appearance. However, in general, after the first line, we've already given R **all** the data we need to make the figure we're going for. Now, all we need to do is specify exactly what we want the data to look like. The next line:

```
plot + geom_histogram()
```

Here, we've taken the empty plot object, and added a layer. This layer takes the form of a histogram, but it could have just as easily been something else. There aren't very many layers which will work without some additional information (like a second variable), but just for kicks, here's an example of something else we *could* have done:

```
plot + geom_dotplot()
```



We've taken the exact same underlying object (`plot`), and added a different layer. It's clear that the underlying data is the same, but this lets us visualize it in a slightly different way.

The original objective, however, was to compare the first locomotion score to the second. While we can do that with these plots, it isn't ideal for this purpose for a couple of reasons:

1. Having two separate plots forces us to shift our eyes back and forth between the two. It would be much better if they were plotted together.

2. The sparsity of our data leads our histogram to be not-very-smooth. Kind of chunky looking. I love chunky peanut butter, but histograms, not so much.

So, we can fix this by plotting both of these variables on the same object, letting them overlap somehow, and using a different kind of layer. But wait! How can we do this? Look back at the creation of the plotting object:

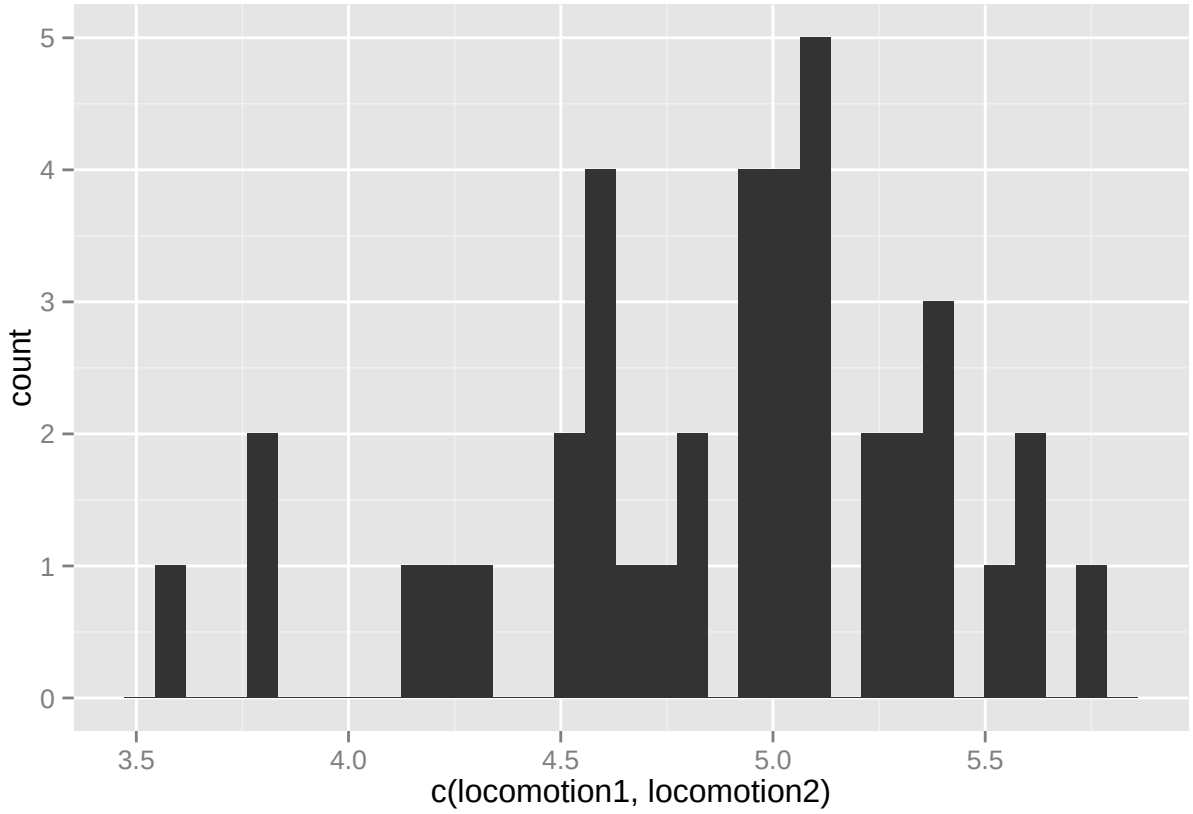
```
plot <- ggplot(breach, aes(x=locomotion1))
```

We've assigned a variable to x, but what we want is to have TWO variables on x. Your intuition may say to do this:

```
plot <- ggplot(breach, aes(x=c(locomotion1, locomotion2)))
```

Your intuition would be wrong. This will plot just fine, but it isn't what we want. Here, R has no way of knowing which scores are from `locomotion1` and `locomotion2`, and so will just plot them all together. Observe:

```
plot <- ggplot(breach, aes(x=c(locomotion1, locomotion2)))
plot + geom_histogram()
```



Side note: using `c()` will combine things into a vector. As soon as you perform `c()` on something, all the component parts are treated as coming from the same thing.

We can get around this issue, but it requires some wrangling of our data. To preview, we will eventually aim to have two vectors in our dataframe: one of the values to be plotted along the x axis, and another indicating which of two groups each value belongs to.

Questions:

1. There are other reasons we might not like the plots we just produced. For example, maybe we want the bins into which the observations are sorted to be a bit larger. Can you do this? What setting do you think produces the best visualization for the data? (*hint*: `?geom_histogram`)

2. In the last plot, the x axis label is horrible. Change it to something more appropriate. You might find this helpful.

Reshaping

We're going to reshape our data so that we can feed it into ggplot. Mark's data is currently organized in the 'wide' format. In psychology studies, this usually means that each row is an individual, and everything we know or have collected from that individual is on that row. For many statistical computing purposes, it is common to use a 'long' format. Here, the rows take the form of observations on an individual. Since we're observing many of our variables twice on the same individual, this implies that each individual will have two rows. If we had done an experiment with, let's say, 200 reaction time trials, then we would have 200 rows for each individual in a 'long' format.

In order to get this data from wide to long, we can use the `reshape2` package, created by....yep, that's right. Our dear friend Hadley Wickham. (I know! This guy! Jeez! Everytime I load one of his packages I think about sending him one of the books on his amazon wishlist).

There are two main functions in the `reshape2` package: `melt` and `cast`. The former takes wide data and converts it to long, and the latter does the converse. So we need to use melt.

```
library(reshape2)
breach <- melt(breach,
     id.vars = c('subj', 'flourishing', 'swls', 'age', 'exp', 'apride',
                 'lpride', 'eom', 'rankindiv', 'rank21', 'averank'),
     measure.vars = c('locomotion1', 'locomotion2', 'assessment1', 'assessment2',
                      'promotion1', 'promotion2', 'prevention1', 'prevention2',
                      'stress1', 'stress2', 'lie2', 'lie1', 'LxA1', 'LxA2'))
str(breach)
```

```
## 'data.frame':    280 obs. of  13 variables:
##  $ subj       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ flourishing: num  6.5 5.25 6.25 6.88 6.25 6 7 6 5.75 6 ...
##  $ swls       : num  6 5.8 6.2 6.8 6.2 5.6 7 5.8 5.8 5.8 ...
##  $ age        : int  23 26 22 26 26 29 28 24 27 22 ...
##  $ exp        : int  3 6 4 2 4 3 0 3 4 1 ...
##  $ apride     : int  37 38 32 40 45 38 38 33 36 37 ...
##  $ lpride     : int  46 38 40 46 44 44 49 40 38 38 ...
##  $ eom        : int  40 35 43 42 38 34 39 41 38 37 ...
##  $ rankindiv  : int  10 4 6 16 3 5 12 9 11 19 ...
##  $ rank21     : num  13 3.5 5 16 3.5 7 21 10 8 15 ...
##  $ averank    : num  11.5 3.75 5.5 16 3.25 6 16.5 9.5 9.5 17 ...
##  $ variable   : Factor w/ 14 levels "locomotion1",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ value      : num  5 4.5 5.25 4.83 5.08 5.08 4.67 5 5.58 5.08 ...
```

```
summary(breach)
```

```
##       subj        flourishing         swls           age
##  Min.   : 1.00   Min.   :4.880   Min.   :4.40   Min.   :21.0
##  1st Qu.: 5.75   1st Qu.:5.750   1st Qu.:5.75   1st Qu.:24.0
```

```
##  Median :10.50   Median :6.000   Median :5.80   Median :26.0
##  Mean   :10.50   Mean   :5.976   Mean   :5.91   Mean   :25.9
##  3rd Qu.:15.25   3rd Qu.:6.250   3rd Qu.:6.20   3rd Qu.:28.0
##  Max.   :20.00   Max.   :7.000   Max.   :7.00   Max.   :31.0
##
##       exp           apride          lpride           eom
##  Min.   :0.00   Min.   :28.00   Min.   :29.00   Min.   :34.00
##  1st Qu.:2.00   1st Qu.:36.00   1st Qu.:38.00   1st Qu.:37.00
##  Median :3.00   Median :38.00   Median :41.50   Median :38.50
##  Mean   :3.25   Mean   :37.45   Mean   :41.35   Mean   :38.45
##  3rd Qu.:4.00   3rd Qu.:39.00   3rd Qu.:46.00   3rd Qu.:40.00
##  Max.   :8.00   Max.   :46.00   Max.   :49.00   Max.   :43.00
##
##    rankindiv         rank21         averank              variable
##  Min.   : 1.00   Min.   : 1.00   Min.   : 1.500   locomotion1: 20
##  1st Qu.: 5.75   1st Qu.: 5.75   1st Qu.: 5.875   locomotion2: 20
##  Median :10.50   Median :11.00   Median :10.500   assessment1: 20
##  Mean   :10.50   Mean   :11.00   Mean   :10.750   assessment2: 20
##  3rd Qu.:15.25   3rd Qu.:16.25   3rd Qu.:16.500   promotion1 : 20
##  Max.   :20.00   Max.   :21.00   Max.   :20.000   promotion2 : 20
##                                                   (Other)    :160
##      value
##  Min.   : 1.000
##  1st Qu.: 2.600
##  Median : 3.830
##  Mean   : 6.367
##  3rd Qu.: 4.830
##  Max.   :33.100
##  NA's   :1
```

Okay, we're getting closer. We can see that we've now got 280 observations across 13 different variables. How did this happen and what are we left with? Let's look at the melt function, line-by-line.

```
breach <- melt(breach,
```

Nothing mind-blowing here. We've just called the `melt` function, and then fed it our dataframe as the first argument. The next line is the `id.vars` argument:

```
id.vars = c('subj', 'flourishing', 'swls', 'age', 'exp', 'apride', 'lpride', 'eom',
'rankindiv', 'rank21', 'averank'),
```

If we look closely at the updated breach dataframe, it's clear that these are the variables which get repeated. That is, while there are only 20 subjects, each subject now has their data spread across 14 rows. All the other variables listed in this argument are similarly repeated. Finally, we have the `measure.vars` argument:

```
measure.vars = c('locomotion1', 'locomotion2', 'assessment1', 'assessment2', 'promotion1',
'promotion2', 'prevention1', 'prevention2', 'stress1', 'stress2', 'lie2', 'lie1', 'LxA1',
'LxA2'))
```

What happened to all of these in our new dataframe? it seems they've all been dumped into two new variables. One has the name `variables` and the other has the name `value`. Let's examine the data for subject 2.

```
breach[breach$subj == 2,] #read:  from the dataframe breach, give me the rows where subject = 2.  After
```

```
##    subj flourishing swls age exp apride lpride eom rankindiv rank21
## 2     2        5.25  5.8  26   6     38     38  35         4    3.5
```

```
## 22     2        5.25  5.8  26   6     38     38  35         4    3.5
## 42     2        5.25  5.8  26   6     38     38  35         4    3.5
## 62     2        5.25  5.8  26   6     38     38  35         4    3.5
## 82     2        5.25  5.8  26   6     38     38  35         4    3.5
## 102    2        5.25  5.8  26   6     38     38  35         4    3.5
## 122    2        5.25  5.8  26   6     38     38  35         4    3.5
## 142    2        5.25  5.8  26   6     38     38  35         4    3.5
## 162    2        5.25  5.8  26   6     38     38  35         4    3.5
## 182    2        5.25  5.8  26   6     38     38  35         4    3.5
## 202    2        5.25  5.8  26   6     38     38  35         4    3.5
## 222    2        5.25  5.8  26   6     38     38  35         4    3.5
## 242    2        5.25  5.8  26   6     38     38  35         4    3.5
## 262    2        5.25  5.8  26   6     38     38  35         4    3.5
##     averank    variable value
## 2      3.75 locomotion1  4.50
## 22     3.75 locomotion2  4.58
## 42     3.75 assessment1  4.25
## 62     3.75 assessment2  3.92
## 82     3.75  promotion1  4.00
## 102    3.75  promotion2  4.17
## 122    3.75 prevention1  2.20
## 142    3.75 prevention2  2.20
## 162    3.75     stress1  2.00
## 182    3.75     stress2  2.50
## 202    3.75        lie2  2.67
## 222    3.75        lie1  2.33
## 242    3.75        LxA1 20.20
## 262    3.75        LxA2 21.00
```

This (I think) really clearly illustrates what's going on here. Two things happen with all the variables listed in the `measure.vars` argument. First, the name of each variable gets put into a new vector called, unfortunately `variable`. Second, the corresponding value for that variable is listed in a second new vector labeled 'value'. Thus, since we had 14 variables listed in `measure.vars`, we now have 14 lines for each subject.

But, like I pointed out before, this isn't quite what we wanted. Remember our goal: We want a histogram where we can plot two variables on the x axis. And remember how I said we would do it:

2 vectors - *Vector 1: values to be plotted* - *Vector 2: indicates which of two groups each value belongs to.*

I know we just turned our 'wide' data into 'long', but now we need to go back. Our data is long, but it's *too* long... (yeah, yeah... har har)

```
#breach <- dcast(breach, )
```