Managing Data in R

KEL - Quantitative Methods

Housekeeping

- You need some data for this class (assignment one)
- If you still do not have data, and do not have a plan to acquire data (e.g. chatting with your advisor, surfing dryad, using some from a cool paper you recently read), we need to speak about your options ASAP.
- Please email me `klangwig@vt.edu` if you are worried about this.
- **I need your github username turned in before next class**

Change in Assignment

- Please turn in your GitHub name and the paragraph about your data on canvas.
  - There is a text entry box to do this under the “assignments” tab. (Don’t send me via email or a canvas message.)
- You can turn in your code and **data file** from assignment 1 using GitHub on Thursday.

Optional Code session?

- For those new to R, would this be helpful?

Goals

You should be able to

- read data into R
- understand and control how R represents those data
    - numbers, characters, factors, missing values
- examine the data visually, numerically, textually, etc.

Representations

Numeric and character types are fairly straightforward, and you rarely have to worry about when and whether R represents things as integers or *floating point*.

You do need to know about **factors**, and to be aware when your variables are being treated as such. See lecture 1 for more about factors.

Missing values

When you input data, you need to be aware of `NA` (“not available”). Your read function has an option called `na.strings` which you can use to communicate between R and your CSV files, for example. You need to know that

- use `is.na()` to test for `NA` values, `na.omit()` to drop them, and the optional `na.rm` argument in some functions (`mean`, `sum`, `median` . . .)

Changing representations

- R has a big suite of functions for creating, testing and changing representations.

-These have names like `factor()`, `as.numeric()` and `is.character()`.

Examination

You should think creatively, and early on, about how to check your data. Is it internally consistent? Are there extreme outliers? Are there typos? Are there certain values that really mean something else?

An American Airlines memo about fuel reporting from the 1980s complained of multiple cases of:

- Reported departure fuel greater than aircraft capacity
- Reported departure fuel less than minimum required for trip
- Reported arrival fuel greater than reported departure fuel

You should think about what you can test, and what you can fix if it's broken.

Visualizing data with graphs

Graphical approaches are really useful for data cleaning; we will discuss this more later on.

To get you started here are just a few:

- `hist`: will make a histogram plot

Example

```
batdat=read.csv("/Users/klangwig/Dropbox/teaching/quant gra
head(batdat)
```

```
##      swab_id gd          gdL swab_type state          site
## 1 KL15WI0002  1 0.00007560       BAT    WI HORSESHOE BAY
## 2 KL15WI0003  1 0.47879100       BAT    WI HORSESHOE BAY
## 3 KL15WI0004  0         NA       BAT    WI HORSESHOE BAY
## 4 KL15WI0005  1 0.00000551       BAT    WI HORSESHOE BAY
## 5 KL15WI0006  1 0.00003560       BAT    WI HORSESHOE BAY
## 6 KL15WI0007  1 0.00003160       BAT    WI HORSESHOE BAY
##   temp count
## 1   NA     3
## 2   NA  1110
## 3   NA  1110
## 4   NA  1110
## 5   NA  1110
## 6   NA  1110
```

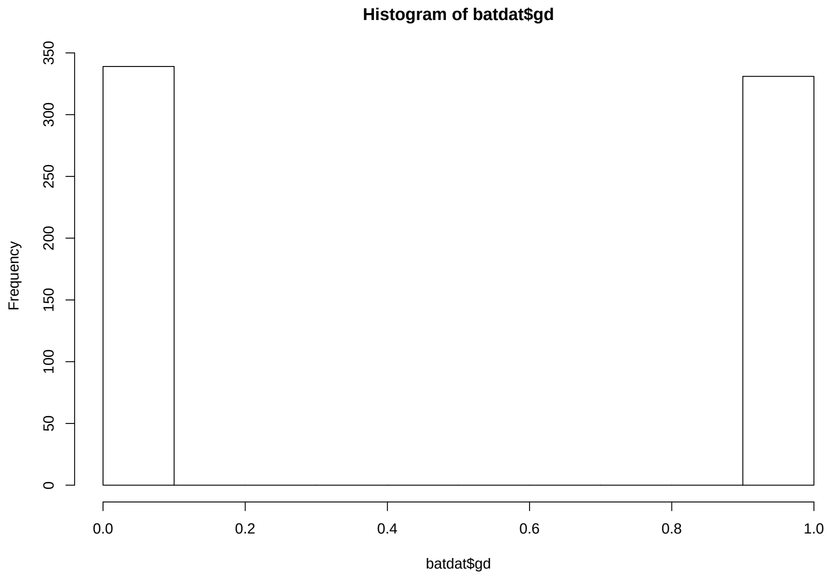
Example Cont.

```r
unique(batdat$species)
```

```
## [1] MYSE      MYLU      PESU      EPFU      SUBSTRATE
## Levels: EPFU MYLU MYSE PESU SUBSTRATE
```

Example Cont.

```
hist(batdat$gd)
```



Some other useful tools

- `dim`: gives the dimensions of the dataframe
- `str`: gives the structure of each variable
- `glimpse`: a dyplr function, that allows for preview as much of each column as possible
- `head`: get the first 6 rows
- `tail`: get the last 6 rows

How do you clean data?

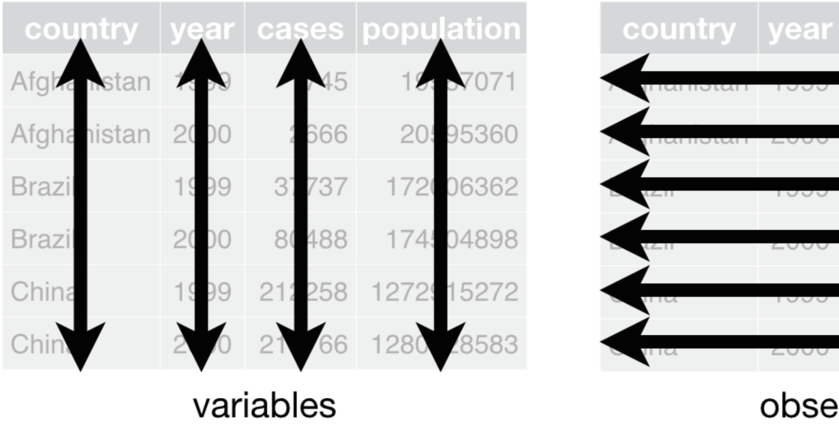
What R functions do you know that are useful for examination?
What are your strategies?

Tidy(ing) data

Hadley Wickham has defined a concept of tidy data, and has recently introduced the `tidyr` package.

- Each variable is in a column
- Each observation is in a row
- "Long" rather than "wide" form
- Sometimes duplicates data
- Statistical modeling tools and graphical tools (especially the **ggplot2** package) in R work best with long form

An example of tidy data



Putting your data in tidy format

- Discerning what is a variable can be hard when making data files
- For example, species in my bat dataset is usually a single variable
- I usually also include a “count” column (the number of individuals at a site)
- But what if I wanted to test the effect of the count of one species (e.g.MYSE) on another? Now MYSE count is actually a variable.

Example with bat data

What if I wanted to test how the count of MYSE influenced infection in MYLU? I need to MYSE to be a variable

Spread and Gather

- the reshape2 package (also by Hadley Wickham) provides some useful tools for this kind of problem
- You can find more information about using melt and cast here:`https://www.statmethods.net/management/reshape.html`

Here, we will use spread and gather

```
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.4.4
```

```
batdat$lgdL=log10(batdat$gdL)#log the amount of fungus
batcounts<-aggregate(count~species+site+date,data=batdat, F
#make a df of bat counts
batcounts.wide<-spread(batcounts, species,count,convert=T)
#spread that dataframe
```

What do these look like?

```
##   species          site     date count
## 1    MYLU      ST. JOHN 11/20/15    87
## 2    MYLU HORSESHOE BAY  11/7/15   646
## 3    MYSE HORSESHOE BAY  11/7/15     1
## 4    MYLU    BEAR CREEK  11/9/15   116
## 5    MYSE    BEAR CREEK  11/9/15     7
## 6    PESU    BEAR CREEK  11/9/15    50
```

```
##            site    date EPFU MYLU MYSE PESU
## 1    BEAR CREEK 11/9/15   NA  116    7   50
## 2    BEAR CREEK 3/10/17   NA   38   NA   22
## 3    BEAR CREEK  3/4/15    9   97    0   55
## 4    BEAR CREEK  3/7/16    5  122   16   50
## 5 HORSESHOE BAY 11/7/15   NA  646    1   NA
## 6 HORSESHOE BAY 2/27/15   NA 1110    3    2
```

We can make identical dataframes for loads

```
##      species           site     date      lgdL
## 1       MYLU       ST. JOHN 11/20/15 -3.702218
## 2       MYLU HORSESHOE BAY  11/7/15 -3.181897
## 3       MYSE HORSESHOE BAY  11/7/15 -2.568128
## 4       MYLU HORSESHOE BAY  2/27/15 -3.629430
## 5       MYSE HORSESHOE BAY  2/27/15 -4.021487
## 6 SUBSTRATE HORSESHOE BAY  2/27/15 -4.406571
```

```
##              site    date      EPFU      MYLU      MYSE
## 1    BEAR CREEK 3/10/17        NA -1.404181        NA -
## 2    BEAR CREEK  3/7/16 -4.434528 -3.484241 -4.142065 -
## 3 HORSESHOE BAY 11/7/15        NA -3.181897 -2.568128
## 4 HORSESHOE BAY 2/27/15        NA -3.629430 -4.021487
## 5 HORSESHOE BAY  3/1/17        NA -1.338297        NA -
## 6 HORSESHOE BAY  3/3/16 -1.854368 -1.172071        NA
```

Now, merge dataframes together for wide format

```
batwide=merge(batloads.wide,batcounts.wide,by=c("site","dat
#merge df together by site and date
head(batwide)
```

```
##               site    date    EPFU.x    MYLU.x    MYSE.x
## 1    BEAR CREEK 3/10/17        NA -1.404181        NA -1
## 2    BEAR CREEK  3/7/16 -4.434528 -3.484241 -4.142065 -5
## 3 HORSESHOE BAY 11/7/15        NA -3.181897 -2.568128
## 4 HORSESHOE BAY 2/27/15        NA -3.629430 -4.021487
## 5 HORSESHOE BAY  3/1/17        NA -1.338297        NA -1
## 6 HORSESHOE BAY  3/3/16 -1.854368 -1.172071        NA
##   EPFU.y MYLU.y MYSE.y PESU.y
## 1     NA     38     NA     22
## 2      5    122     16     50
## 3     NA    646      1     NA
## 4     NA   1110      3      2
## 5     NA     10     NA     10
## 6      4    188     NA     NA
```

Here's another example (by Ben Bolker)

Look at some example data that comes with the tidyr package:

```
smiths
```

```
## # A tibble: 2 x 5
##   subject     time   age weight height
##   <chr>      <dbl> <dbl>  <dbl>  <dbl>
## 1 John Smith     1    33     90   1.87
## 2 Mary Smith     1    NA     NA   1.54
```

Gather

The default `gather()` operation squashes everything too far, including the subject name and time in the value column ...

```r
gather(smiths)
```

```
## # A tibble: 10 x 2
##    key     value
##    <chr>   <chr>
##  1 subject John Smith
##  2 subject Mary Smith
##  3 time    1
##  4 time    1
##  5 age     33
##  6 age     <NA>
##  7 weight  90
##  8 weight  <NA>
##  9 height  1.87
## 10 height  1.54
```

Gathering variables

We can specify that we only want to gather the age and `weight` variables (however, we have to specify the name of key and value columns explicitly).

```
print(smelt <- gather(smiths, key="var", value="value",
       c(age,weight)))
```

```
## # A tibble: 4 x 5
##   subject     time height var    value
##   <chr>      <dbl>  <dbl> <chr>  <dbl>
## 1 John Smith     1   1.87 age       33
## 2 Mary Smith     1   1.54 age       NA
## 3 John Smith     1   1.87 weight    90
## 4 Mary Smith     1   1.54 weight    NA
```

Make a column for each subject (= a row for each measurement) using Spread

```
spread(smelt, key=subject, value)
```

```
## # A tibble: 4 x 5
##    time height var    `John Smith` `Mary Smith`
##   <dbl>  <dbl> <chr>         <dbl>        <dbl>
## 1     1   1.54 age              NA           NA
## 2     1   1.54 weight           NA           NA
## 3     1   1.87 age              33           NA
## 4     1   1.87 weight           90           NA
```

Make a column for each value (= a row for each person):

```r
spread(smelt, key=var, value)
```

```
## # A tibble: 2 x 5
##   subject      time height   age weight
##   <chr>       <dbl>  <dbl> <dbl>  <dbl>
## 1 John Smith      1   1.87    33     90
## 2 Mary Smith      1   1.54    NA     NA
```

Take the mean for each variable:

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
smelt %>% group_by(var) %>% summarise(mean=mean(value, na.r
```

```
## Warning: package 'bindrcpp' was built under R version 3.
```

```
## # A tibble: 2 x 2
##   var     mean
##   <chr>  <dbl>
## 1 age       33
## 2 weight    90
```

Report how many values are in each mean:

```
smelt %>% group_by(var) %>%
    summarise(mean=mean(value,na.rm=TRUE),
              n=length(na.omit(value)))
```

```
## # A tibble: 2 x 3
##   var     mean     n
##   <chr>  <dbl> <int>
## 1 age       33     1
## 2 weight    90     1
```

So how do we create tidy datasets?

- Make your data as tidy as possible
- Learn to manipulate data in R and hardcode these changes into your scripts
- There is no perfect method - each dataset is unique
- Manipulating data in R is hard, sometimes harder than excel. But learning to do it SO worth it because you will save hours of time for each project you do.

Tools

base R

- `reshape`: wide-to-long and vice versa
- `merge`: join data frames
- `ave`: compute averages by group
- `subset`, `[`-indexing: select obs and vars
- `transform`: modify variables and create new ones
- `aggregate`: split-apply-summarize
- `split`, `lapply`, `do.call(rbind())`: split-apply-combine
- `sort`

The tidyverse

- tidyr package: gather, spread
- dplyr package:
    - mutate
    - select
    - filter
    - group_by
    - summarise
    - arrange

Group by, Mutate, and Summarise

- group_by is my favorite tidyverse command which has cut my need to write loops in half
- group_by allows you to do calculations on groups of things, for example, by species or year

```
batdat %>%
  group_by(species) %>%
    summarise(mean.fungal.loads=mean(lgdL,na.rm=TRUE))
```

```
## # A tibble: 5 x 2
##   species   mean.fungal.loads
##   <fct>                 <dbl>
## 1 EPFU                  -3.64
## 2 MYLU                  -3.03
## 3 MYSE                  -3.69
## 4 PESU                  -2.04
## 5 SUBSTRATE             -4.11
```

Summarise versus Mutate

- `summarise` creates a new dataframe
- `mutate` does a calculation where it add a new column to your existing dataframe

```
batdat_with_sample_size = batdat %>%
  #create a new dataframe  called batdat_with_sample_size
    group_by(site,species,date) %>%
  #you can group_by multiple things
    mutate(sample.size=length(swab_id))
#this adds a column to the dataframe
```

What does our dataframe look like now?

```
head(batdat_with_sample_size[c(1,6,7,8,12)])
```

```
## # A tibble: 6 x 5
## # Groups:   site, species, date [2]
##   swab_id    site           date    species sample.size
##   <fct>      <fct>          <fct>   <fct>         <int>
## 1 KL15WI0002 HORSESHOE BAY 2/27/15 MYSE              4
## 2 KL15WI0003 HORSESHOE BAY 2/27/15 MYLU             20
## 3 KL15WI0004 HORSESHOE BAY 2/27/15 MYLU             20
## 4 KL15WI0005 HORSESHOE BAY 2/27/15 MYLU             20
## 5 KL15WI0006 HORSESHOE BAY 2/27/15 MYLU             20
## 6 KL15WI0007 HORSESHOE BAY 2/27/15 MYLU             20
```

```
#this is just showing a few columns for effect
```

Managing Pipelines in R

- Pipelines are ways of carefully recording and systematizing the steps you take to work with your data
- The idea is that you should be able to delete any results of computer calculations and be able to quickly re-do them
- Ideally your project will depend on:
- Some data files
- Some scripts
- Something that tells you how these things go together (RMarkdown is helpful for this), at minimum a README file

Advantages of this approach

- Clarity: we aren't confused about the 600 pages of information stored with our projects
- Reproducibility: we can always re-do something we did
- Flexibility : we can use different data and re-create the same thing

Spreadsheets

- Spreadsheets are a useful (and obvious) tool for working with R
- `read.csv` and `write.csv` are very useful commands for working with spreadsheets
- when using `write.csv` use `row.names=F` to avoid line numbers
- Importantly, spreadsheets are for storing data, NOT FOR MANIPULATING DATA
- Your goal should be to take data from a spreadsheet and manipulate it entirely using scripts.
- Avoid spreadsheet addiction: `http://www.burns-stat.com/documents/tutorials/spreadsheet-addiction/`
- The jist is: friends don't let friends use excel for statistics.

Database

- Your spreadsheet is a database (just because it isn't stored in microsoft access doesn't mean it isn't!)
- "small" databases are usually considered to be fewer than 1000 observations of 10-20 vars
- "medium" databases are about 1000 to 100,000 observations of about 10-50 vars. These are most helpful with data handling packages.
- "large" means millions of observations and potentially 1000s of variables. These may need to be stored in an external application.

Working in Github

- Git is version control system, with the original purpose of allowing groups to work collaboratively on software projects
- Git manages the evolution of a set of files - called a repository
- A repository is essentially a folder where you store your stuff
- Version control works a bit like “Track Changes” in word, Git will track the changes we make to our code so we can return to previous versions
- It also allows collaboration so I can look at your code and make changes - a bit like a more complicated version of Google Docs

Will this hurt?

- Maybe!
- But, I think this important enough that we NEED exposure to this. This is the future!

But I only code alone!

- You need to carefully document your steps if the only person you are sharing code with is the future version of yourself
- In addition, most journals require publicly available data and code - open code is the norm, not the exception.
- Using Git has gotten easier. We used to have to use command line to communicate with Git, but now we can just use RStudio!

Terminology

- repository: A directory or storage space where your projects can live. Sometimes GitHub users shorten this to “repo.” (If you’re cool like that.) It is usually a local folder on your computer. You can keep code files, text files, image files, you name it, inside a repository.
- commit: This is the command that gives Git its power. When you commit, you are taking a “snapshot” of your repository at that point in time, giving you a checkpoint to which you can reevaluate or restore your project to any previous state.When you first start “commiting”, it is important to remember this is taking the picture, not SENDING the picture. (Sending is called “pushing”)

Terminology cont.

- branch: How do multiple people work on a project at the same time without Git getting them confused? Usually, they “branch off” of the main project with their own versions full of changes they themselves have made. After they’re done, it’s time to “merge” that branch back with the “master,” the main directory of the project. Because we’ll be working within our own repos, we don’t need to worry too much about branching but is good to know for future.
- push: This is how you upload your file to GitHub. Remember, you need to both commit and push for your file to be sent to GitHub.

Sending your files to our class repository

- We have an “organization” account for our class
- Normally, we would have to pay for private repositories, but I emailed github and they are giving us UNLIMITED private repositories. That’s pretty awesome.
- Why should we want things open-source? Why not?

Installing Git

- I’ll be absent. Email me when you’ve done this successfully!

Installing Git

- Just kidding.
- Please try to start this before our next class.
- Here is a link: `http://happygitwithr.com/install-git.html#install-git`
- Please follow instructions to get started with git.
- Try to install github in the most scientific way possible - if one way doesn't work, try the next, and google your mistakes!