

# 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](mailto:klangwig@vt.edu) if you are worried about this.
- ▶ **I need your github username turned in before next class**

# Assignment Reminder

- ▶ 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. (You don’t need to send me an email or a canvas message.)

# 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.

# Getting Started with Data

- ▶ Save files as .csv
- ▶ IMPORTANT – saving an excel file as a CSV means that you will lose some data
- ▶ For example, if you used excel to calculate a formula, the formula will be gone as R will just store this as plain text
- ▶ DON'T USE EXCEL TO DO CALCULATIONS – JUST ADD THIS TO YOUR CODE IN R
- ▶ Use smart column names. R can't handle spaces in your column names, so get rid of those. Also don't use a bunch of capitals unnecessarily because it slows down your coding. e.g. use “species” not “Species”

## Making your excel file

- ▶ Excel files should have a list of column names at the top only and variable values
- ▶ Your excel file should not look like your field data sheet

What is wrong with this entry?

	A	B	C	D	E
1	Site	Neda Mine	Date	1/13/18	State
2					
3	sample id	section	species		
4	1	A	pesu		
5	2	A	pesu		
6	3	B	mylu		
7					
8					
9					
10					

Corrected entry

	A	B	C	
1	sample id	section	species	s
2	1	A	pesu	n
3	2	A	pesu	n
4	3	B	mylu	n
5				
6				



# 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.

## Date reminder

Working with dates can be a bit frustrating because as time units get larger, they become more variable. For example, at what day does the 75th percentile of the month fall?

An important note – macs and windows machines often handle dates differently and the default is different in excel.

One a mac the default is mo/day/two digit year – e.g. 01/13/18 is January 13, 2018, but on a PC the default is “01/13/2018”. This can result in some frustration between people sharing scripts!

## 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 ...`)
- ▶ in the tidyverse, you can use `drop_na()` to remove NA

# 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.

# Things to fix in excel

- ▶ naming inconsistencies (see maple example last lecture)
- ▶ column name issues (spaces)
- ▶ use excel's find and replace and filter function to find these

# 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/Desktop/VT/teaching/quant
```

```
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
##	country	count				
## 1	u.s.	3				
## 2	u.s.	1110				
## 3	u.s.	1110				
## 4	u.s.	1110				
## 5	u.s.	1110				
## 6	u.s.	1110				



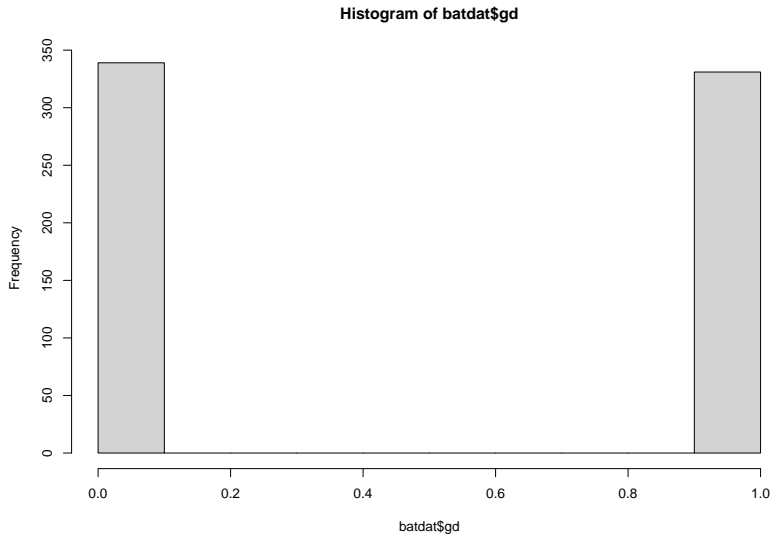
## Example Cont.

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

```
## [1] "MYSE"      "MYLU"      "PESU"      "EPFU"      "SUFU"
```

## 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

## Some other useful base R tools

- ▶ `aggregate`: creating summary dfs using various functions on a set of variables
- ▶ `match`: match a value from one dataframe into another df given a common column. Only the value you want to copy is matched.
- ▶ `merge`: merge two dataframes together based on some common columns, all columns are merged.
- ▶ Note `merge` is made more versatile by the `join` functions in `tidyverse`

## Example with bat data

```
batdat=read.csv("/Users/klangwig/Desktop/VT/teaching/quant  
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
##	country	count				
## 1	u.s.	3				
## 2	u.s.	1110				
## 3	u.s.	1110				
## 4	u.s.	1110				
## 5	u.s.	1110				
## 6	u.s.	1110				

Here, we will use aggregate

```
batcounts<-aggregate(count~species+site+date,data=batdat, FUN=
#make a df of bat counts
head(batcounts)
```

##	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

## We can make identical dataframes for loads

```
batdat$lgdL=log10(batdat$gdL)#log the amount of fungus  
batloads<-aggregate(lgdL~species+site+date,data=batdat, FUN  
head(batloads)
```

##	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

## We can “match” the loads column into our count df

```
batloads$unique.row.id = paste(batloads$species,batloads$site)
batcounts$unique.row.id = paste(batcounts$species,batcounts$site)
#dataframe you are bringing to first, and the one you match to
batloads$count = batcounts$count[match(batloads$unique.row.id,batcounts$unique.row.id)]
head(batloads)
```

##	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

##	count
## 1	87
## 2	646
## 3	1
## 4	1110
## 5	3



Alternatively, we can merge dataframes together for wide format

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

```
##           site    date species.x    lgdL           un  
## 1 BEAR CREEK 3/10/17    MYLU -1.404181    MYLU BEAR  
## 2 BEAR CREEK 3/10/17    MYLU -1.404181    MYLU BEAR  
## 3 BEAR CREEK 3/10/17    PESU -1.784292    PESU BEAR  
## 4 BEAR CREEK 3/10/17    PESU -1.784292    PESU BEAR  
## 5 BEAR CREEK 3/10/17 SUBSTRATE -4.127488 SUBSTRATE BEAR  
## 6 BEAR CREEK 3/10/17 SUBSTRATE -4.127488 SUBSTRATE BEAR  
## species.y count.y           unique.row.id.y  
## 1    MYLU      38 MYLU BEAR CREEK 3/10/17  
## 2    PESU      22 PESU BEAR CREEK 3/10/17  
## 3    MYLU      38 MYLU BEAR CREEK 3/10/17  
## 4    PESU      22 PESU BEAR CREEK 3/10/17  
## 5    MYLU      38 MYLU BEAR CREEK 3/10/17
```

# 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 introduced the `tidyverse` 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

country	year	cases	population
Afghanistan	1999	17745	19337071
Afghanistan	2000	2666	20395360
Brazil	1999	37737	172006362
Brazil	2000	80488	174304898
China	1999	212258	1272915272
China	2000	217766	128028583

variables

country
Afghanistan
Afghanistan
Brazil
Brazil
China
China

# Learning about the tidyverse

- ▶ <https://www.tidyverse.org>

## 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?

## Example with bat count data

```
batcounts<-aggregate(count~species+site+date,data=batdat, FUN=
head(batcounts)
```

##	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

## Testing the effect of one species on another

- ▶ What if I wanted to test how the count of MYSE influenced counts of MYLU? I need to MYSE to be a variable



# Pivoting

- ▶ Here is a link to vignette:  
<https://tidyr.tidyverse.org/articles/pivot.html>
- ▶ Using `pivot_wider()` and `pivot_longer()` we can specify how the metadata stored become data variables
- ▶ This has replaced `spread` and `gather`

Let's 'pivot' (make wider)

```
library(tidyr)
batcounts.wide<- batcounts %>% #this says - make a new df
  pivot_wider(names_from = species, values_from = count)
##make columns for each of the values in the species column
```

## What does our new df look like?

```
head(batcounts.wide)
```

```
## # A tibble: 6 x 6
```

##	site	date	MYLU	MYSE	PESU	EPFU
##	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	ST. JOHN	11/20/15	87	NA	NA	NA
## 2	HORSESHOE BAY	11/7/15	646	1	NA	NA
## 3	BEAR CREEK	11/9/15	116	7	50	NA
## 4	HORSESHOE BAY	2/27/15	1110	3	2	NA
## 5	HORSESHOE BAY	3/1/17	10	NA	10	NA
## 6	BEAR CREEK	3/10/17	38	NA	22	NA

## Here's another example using "pivot"

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

```
fish_encounters
```

```
## # A tibble: 114 x 3
##   fish station seen
##   <fct> <fct>   <int>
## 1 4842 Release     1
## 2 4842 I80_1       1
## 3 4842 Lisbon     1
## 4 4842 Rstr       1
## 5 4842 Base_TD     1
## 6 4842 BCE        1
## 7 4842 BCW        1
## 8 4842 BCE2       1
## 9 4842 BCW2       1
## 10 4842 MAE       1
## # ... with 104 more rows
```

## Pivot\_wider

Using pivot\_wider()

```
fish_encounters %>%  
  pivot_wider(names_from = station, values_from = seen)
```

## # A tibble: 19 x 12

##	fish	Release	I80_1	Lisbon	Rstr	Base_TD	BCE	BCW
##	<fct>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
##	1 4842	1	1	1	1	1	1	1
##	2 4843	1	1	1	1	1	1	1
##	3 4844	1	1	1	1	1	1	1
##	4 4845	1	1	1	1	1	NA	NA
##	5 4847	1	1	1	NA	NA	NA	NA
##	6 4848	1	1	1	1	NA	NA	NA
##	7 4849	1	1	NA	NA	NA	NA	NA
##	8 4850	1	1	NA	1	1	1	1
##	9 4851	1	1	NA	NA	NA	NA	NA
##	10 4854	1	1	NA	NA	NA	NA	NA
##	11 4855	1	1	1	1	1	NA	NA

## Fill in 0's

```
fish_encounters %>%  
  pivot_wider(  
    names_from = station,  
    values_from = seen,  
    values_fill = list(seen = 0)  
  )
```

```
## # A tibble: 19 x 12
```

##	fish	Release	I80_1	Lisbon	Rstr	Base_TD	BCE	BCW
##	<fct>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
##	1 4842	1	1	1	1	1	1	1
##	2 4843	1	1	1	1	1	1	1
##	3 4844	1	1	1	1	1	1	1
##	4 4845	1	1	1	1	1	0	0
##	5 4847	1	1	1	0	0	0	0
##	6 4848	1	1	1	1	0	0	0
##	7 4849	1	1	0	0	0	0	0
##	8 4850	1	1	0	1	1	1	1
##	9 4851	1	1	0	0	0	0	0

## Making a dataframe long (e.g. tidy)

Let's look at an example of an untidy dataframe.

```
head(relig_income)
```

```
## # A tibble: 6 x 11
##   religion `<$10k` ` $10-20k` ` $20-30k` ` $30-40k` ` $40-5
##   <chr>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 Agnostic      27        34        60        81
## 2 Atheist       12        27        37        52
## 3 Buddhist      27        21        30        34
## 4 Catholic    418       617       732       670
## 5 Don't kn~     15        14        15        11
## 6 Evangeli~    575       869      1064       982
## # ... with 3 more variables: $100-150k <dbl>, >150k <dbl>
## #   Don't know/refused <dbl>
```

Make a row for the number of individuals for each religion by income category

```
relig_income %>%  
  pivot_longer(-religion, names_to = "income", values_to =
```

```
## # A tibble: 180 x 3  
##   religion income      count  
##   <chr>    <chr>    <dbl>  
## 1 Agnostic <$10k      27  
## 2 Agnostic $10-20k    34  
## 3 Agnostic $20-30k    60  
## 4 Agnostic $30-40k    81  
## 5 Agnostic $40-50k    76  
## 6 Agnostic $50-75k   137  
## 7 Agnostic $75-100k  122  
## 8 Agnostic $100-150k 109  
## 9 Agnostic >150k     84  
## 10 Agnostic Don't know/refused 96  
## # ... with 170 more rows
```



## Another example using temporal data:

The billboard dataset has a row for every week and the rank of that song

```
billboard
```

```
## # A tibble: 317 x 79
```

```
##   artist      track    date.entered   wk1    wk2    wk3    wk4
##   <chr>      <chr>    <date>         <dbl> <dbl> <dbl> <dbl>
## 1 2 Pac      Baby D~ 2000-02-26      87     82     72     77
## 2 2Ge+her    The Ha~ 2000-09-02      91     87     92     NA
## 3 3 Doors~   Krypto~ 2000-04-08      81     70     68     67
## 4 3 Doors~   Loser    2000-10-21      76     76     72     69
## 5 504 Boyz   Wobble~ 2000-04-15      57     34     25     17
## 6 98^0       Give M~ 2000-08-19      51     39     34     26
## 7 A*Teens    Dancin~ 2000-07-08      97     97     96     95
## 8 Aaliyah    I Don'~ 2000-01-29      84     62     51     41
## 9 Aaliyah    Try Ag~ 2000-03-18      59     53     38     28
## 10 Adams, ~  Open M~ 2000-08-26      76     76     74     69
## # ... with 307 more rows, and 68 more variables: wk9 <dbl>
```

We want week to be temporal data, but it has letters in it

- ▶ We want names to be a variable called “week” and the values to be a variable called “rank”
- ▶ We want to remove NAs because not all songs stay on the charts for 76 weeks

## Billboard

```
billboard %>%  
  pivot_longer(  
    cols = starts_with("wk"),  
    names_to = "week",  
    names_prefix = "wk",  
    values_to = "rank",  
    values_drop_na = TRUE  
  )
```

```
## # A tibble: 5,307 x 5
```

```
##   artist      track      date.entered week  r  
##   <chr>      <chr>      <date>      <chr> <dbl>  
## 1 2 Pac      Baby Don't Cry (Keep... 2000-02-26 1  
## 2 2 Pac      Baby Don't Cry (Keep... 2000-02-26 2  
## 3 2 Pac      Baby Don't Cry (Keep... 2000-02-26 3  
## 4 2 Pac      Baby Don't Cry (Keep... 2000-02-26 4  
## 5 2 Pac      Baby Don't Cry (Keep... 2000-02-26 5  
## 6 2 Pac      Baby Don't Cry (Keep... 2000-02-26 6  
## 7 2 Pac      Baby Don't Cry (Keep... 2000-02-26 7
```

## Changing something to an integer

- ▶ We want to turn week into an integer so we can easily determine how long a song was on the charts

```
billboard %>%  
  pivot_longer (  
    cols = starts_with("wk"),  
    names_to = "week",  
    names_prefix = "wk",  
    values_to = "rank",  
    values_drop_na = TRUE  
  )
```

```
## # A tibble: 5,307 x 5
```

```
##   artist      track      date.entered week  rank  
##   <chr>      <chr>      <date>      <chr> <dbl>  
## 1 2 Pac      Baby Don't Cry (Keep... 2000-02-26 1  
## 2 2 Pac      Baby Don't Cry (Keep... 2000-02-26 2  
## 3 2 Pac      Baby Don't Cry (Keep... 2000-02-26 3  
## 4 2 Pac      Baby Don't Cry (Keep... 2000-02-26 4
```

## 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
- ▶ sort

# The tidyverse

- ▶ `pivot_longer`, `pivot_wider`
- ▶ `mutate`: add a column
- ▶ `select`: select columns
- ▶ `filter`: select rows
- ▶ `group_by`: group then do something (usually mutate or summarise)
- ▶ `summarise`: make a summary table
- ▶ `arrange`: sort
- ▶ `left_join`: merge see other join options

## 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



## First load the package

```
library(tidyverse)
```

```
## -- Attaching packages -----
```

```
## v ggplot2 3.3.5      v dplyr    1.0.7
```

```
## v tibble  3.1.6      v stringr 1.4.0
```

```
## v readr   2.1.1      v forcats 0.5.1
```

```
## v purrr   0.3.4
```

```
## -- Conflicts -----
```

```
## x dplyr::filter() masks stats::filter()
```

```
## x dplyr::lag()    masks stats::lag()
```

## Group by species

```
batdat$lgdL = log10(batdat$gdL)
batdat %>%
  group_by(species) %>%
  summarise(mean.fungal.loads=mean(lgdL,na.rm=TRUE))
```

```
## # A tibble: 5 x 2
##   species    mean.fungal.loads
##   <chr>          <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=n())  
  #this adds a column to the dataframe
```

## What does our dataframe look like now?

```
head(batdat_with_sample_size[c(1,6,7,8,12,13)])
```

```
## # A tibble: 6 x 6
```

```
## # Groups:   site, species, date [2]
```

```
##   swab_id      site      date    species    lgdL sample
```

```
##   <chr>      <chr>      <chr>    <chr>      <dbl>
```

```
## 1 KL15WI0002 HORSESHOE BAY 2/27/15 MYSE    -4.12
```

```
## 2 KL15WI0003 HORSESHOE BAY 2/27/15 MYLU    -0.320
```

```
## 3 KL15WI0004 HORSESHOE BAY 2/27/15 MYLU     NA
```

```
## 4 KL15WI0005 HORSESHOE BAY 2/27/15 MYLU    -5.26
```

```
## 5 KL15WI0006 HORSESHOE BAY 2/27/15 MYLU    -4.45
```

```
## 6 KL15WI0007 HORSESHOE BAY 2/27/15 MYLU    -4.50
```

```
#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 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 “committing”, 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

- ▶ 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 git in the most scientific way possible - if one way doesn't work, try the next, and google your mistakes!