alex’s slides

Read in data

```
batdat<-read.csv("/Users/alexg8/Desktop/IBRC workshops/R Crash course/bat_data.csv")
```

Naming conventions

The R environment is very sensitive to how you name objects and slight differences are important. For example, “m” and “M” are considered different names and R would recognize them as different objects. The names of R objects also have to start with a letter, not a number. Below are some additional good naming practices:

- Use mostly numbers, underscores (_), and dots (.) in your names.
- Don’t use potentially confusing names like I or O
- Don’t use built-in names (like c, list, or data) for variables.
- Make readable variable names using camelCase, snake_case, or kebab.case
- Avoid variableNamesThatAreExcessivelyLong

Data types

There are several types of data that R recognizes, listed below:

- Logical: True/False data
- Numeric: All real numbers with or without decimal values
- Integer: Real numbers without decimal values.
- Character: String data. Thing of this as any unique string of values, such as “1dkdl;” or “Apple_Pie”
- Factor: Used to describe items that can have a known set of values (gender, social class, etc.). Categorical variables, in statistical terms.
- Date: calendar date, e.g. “10-22-1994”

What type of data do we have?

A quick way to check what kind of data we have in our dataframe is with the str() function:

```
str(batdat)
```

```
## 'data.frame':    670 obs. of  11 variables:
##  $ swab_id  : chr  "KL15WI0002" "KL15WI0003" "KL15WI0004" "KL15WI0005" ...
##  $ gd       : int  1 1 0 1 1 1 1 1 0 1 ...
##  $ gdL      : num  7.56e-05 4.79e-01 NA 5.51e-06 3.56e-05 ...
##  $ swab_type: chr  "BAT" "BAT" "BAT" "BAT" ...
##  $ state    : chr  "WI" "WI" "WI" "WI" ...
##  $ site     : chr  "HORSESHOE BAY" "HORSESHOE BAY" "HORSESHOE BAY" "HORSESHOE BAY" ...
##  $ date     : chr  "2/27/15" "2/27/15" "2/27/15" "2/27/15" ...
##  $ species  : chr  "MYSE" "MYLU" "MYLU" "MYLU" ...
##  $ temp     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ country  : chr  "u.s." "u.s." "u.s." "u.s." ...
##  $ count    : int  3 1110 1110 1110 1110 1110 1110 1110 1110 1110 ...
```

Dates

Dates are notoriously difficult to work with in R and their format when being read in with a dataset depends on if you are using a Mac or PC. On 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!

Additionally, dates will often be read into R as factor data, so it is often necessary to define the format of your date data yourself.

Dates continued

As you can see, currently the “date” column in our dataframe is being recognized as a character rather than a date:

```
str(batdat)
```

```
## 'data.frame':    670 obs. of  11 variables:
##  $ swab_id  : chr  "KL15WI0002" "KL15WI0003" "KL15WI0004" "KL15WI0005" ...
##  $ gd       : int  1 1 0 1 1 1 1 1 0 1 ...
##  $ gdL      : num  7.56e-05 4.79e-01 NA 5.51e-06 3.56e-05 ...
##  $ swab_type: chr  "BAT" "BAT" "BAT" "BAT" ...
##  $ state    : chr  "WI" "WI" "WI" "WI" ...
##  $ site     : chr  "HORSESHOE BAY" "HORSESHOE BAY" "HORSESHOE BAY" "HORSESHOE BAY" ...
##  $ date     : chr  "2/27/15" "2/27/15" "2/27/15" "2/27/15" ...
##  $ species  : chr  "MYSE" "MYLU" "MYLU" "MYLU" ...
##  $ temp     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ country  : chr  "u.s." "u.s." "u.s." "u.s." ...
##  $ count    : int  3 1110 1110 1110 1110 1110 1110 1110 1110 1110 ...
```

Dates continued

To fix that, we need to manually define the date column as being date data, using the function as.Date():

```
batdat$date<-as.Date(batdat$date, format="%m/%d/%y")
str(batdat)
```

```
## 'data.frame':    670 obs. of  11 variables:
##  $ swab_id  : chr  "KL15WI0002" "KL15WI0003" "KL15WI0004" "KL15WI0005" ...
##  $ gd       : int  1 1 0 1 1 1 1 1 0 1 ...
##  $ gdL      : num  7.56e-05 4.79e-01 NA 5.51e-06 3.56e-05 ...
##  $ swab_type: chr  "BAT" "BAT" "BAT" "BAT" ...
##  $ state    : chr  "WI" "WI" "WI" "WI" ...
##  $ site     : chr  "HORSESHOE BAY" "HORSESHOE BAY" "HORSESHOE BAY" "HORSESHOE BAY" ...
##  $ date     : Date, format: "2015-02-27" "2015-02-27" ...
##  $ species  : chr  "MYSE" "MYLU" "MYLU" "MYLU" ...
##  $ temp     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ country  : chr  "u.s." "u.s." "u.s." "u.s." ...
##  $ count    : int  3 1110 1110 1110 1110 1110 1110 1110 1110 1110 ...
```

Long vs. wide format of data

The shape of your data matters to R! The easiest way for R to understand your data is if every row in your dataframe is a unique observation. Sometimes, when we are entering data into excel, we have a tendency to add new observations or data as new columns (wide), rather than new rows (long). **This will cause you headaches down the line**

Luckily, the package "tidyverse" has some helpful functions to change the shape of your data.

```
#install.packages(tidyverse)
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──
```

```
## ✓ ggplot2 3.3.3     ✓ purrr   0.3.4
## ✓ tibble  3.1.1     ✓ dplyr   1.0.5
## ✓ tidyr   1.1.3     ✓ stringr 1.4.0
## ✓ readr   1.4.0     ✓ forcats 0.5.1
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
```

Long vs wide format of data

Let's take a look at our data:

```
head(batdat)
```

```
##      swab_id gd        gdL swab_type state          site       date species
## 1 KL15WI0002  1 0.00007560       BAT    WI HORSESHOE BAY 2015-02-27    MYSE
## 2 KL15WI0003  1 0.47879100       BAT    WI HORSESHOE BAY 2015-02-27    MYLU
## 3 KL15WI0004  0         NA       BAT    WI HORSESHOE BAY 2015-02-27    MYLU
## 4 KL15WI0005  1 0.00000551       BAT    WI HORSESHOE BAY 2015-02-27    MYLU
## 5 KL15WI0006  1 0.00003560       BAT    WI HORSESHOE BAY 2015-02-27    MYLU
## 6 KL15WI0007  1 0.00003160       BAT    WI HORSESHOE BAY 2015-02-27    MYLU
##   temp country count
## 1   NA    u.s.     3
## 2   NA    u.s.  1110
## 3   NA    u.s.  1110
## 4   NA    u.s.  1110
## 5   NA    u.s.  1110
## 6   NA    u.s.  1110
```

As you can see, each row here is an observation (long format), which is what we want.

Long vs wide format of data

However, if for some reason we wanted our data in wide format, we can use the tidyverse function spread() to do so. In the below example, we are going to make a unique column for each date and fill our dataframe with gdL (fungal load data):

```
wide.data <- spread(data=batdat, date, gdL)
head(wide.data)
```

```
##      swab_id gd swab_type state          site species temp country count
## 1 KL15WI0002  1       BAT    WI HORSESHOE BAY    MYSE   NA    u.s.     3
## 2 KL15WI0003  1       BAT    WI HORSESHOE BAY    MYLU   NA    u.s.  1110
## 3 KL15WI0004  0       BAT    WI HORSESHOE BAY    MYLU   NA    u.s.  1110
## 4 KL15WI0005  1       BAT    WI HORSESHOE BAY    MYLU   NA    u.s.  1110
## 5 KL15WI0006  1       BAT    WI HORSESHOE BAY    MYLU   NA    u.s.  1110
## 6 KL15WI0007  1       BAT    WI HORSESHOE BAY    MYLU   NA    u.s.  1110
##   2015-02-27 2015-03-04 2015-03-11 2015-11-07 2015-11-09 2015-11-20 2016-03-03
## 1 0.00007560         NA         NA         NA         NA         NA         NA
## 2 0.47879100         NA         NA         NA         NA         NA         NA
## 3         NA         NA         NA         NA         NA         NA         NA
## 4 0.00000551         NA         NA         NA         NA         NA         NA
## 5 0.00003560         NA         NA         NA         NA         NA         NA
## 6 0.00003160         NA         NA         NA         NA         NA         NA
```

Long vs wide format of data

If we want to change our dataframe back to its original shape (long format), we can use the tidyverse function gather():

```
long.data <- gather(data=wide.data, "2015-02-27", "2015-03-04", "2015-03-11", "2015-11-07",
```

Checking Data

Plot tools to detect errors