R Crash Course

Download and read in data

Please navigate to this URL to download the data we will be working with today:

https://github.com/VTQuantMethodsEEB/ibrc_workshop

Once you open the project (the .Rproj file), you can read in the data. To do so, there's a simple function, read.csv(), that allows you to easily bring your .csv into the R environment. Simply run below line of code to read in your data:

```
batdat <- read.csv("bat_data.csv")
```

Once you've read this file in, it should be stored as a dataframe object called "batdat"

Let's get started

First thing you should always do before working in R: Clear your workspace! This ensures that your R environment is clean and has no objects or functions defined during previous coding sessions. Think of this as cleaning your whiteboard before you start coding.

```
rm(list = ls())
```

Load and install packages

R packages are the "toolkits" of the R computing world. They offer users unique tools suited to whatever it is they want to do, from cleaning data, to building beautiful figures, to creating apps, to analyzing biological data. In order to use these toolboxes, you will need to install and then load them into the R environment.

```
#install.packages("ggplot2")
#library(ggplot2)
```

Confused? Get help!

The R package you just installed has many tools, called "functions", that allow you to make plots and figures in R. But what should you do if you're not sure how to use one of these functions? One of the most useful tools in the R environment is the '?' tool. When you are unsure what a specific function does, you can ask R! For example, if I am not sure what the function unique() does, I can run the below line of code and R will tell me:

```
?unique
```

Examine your data

One of the most important things to do is, of course, to examine your data. Being familiar with the shape and contents of your dataset is essential, and R has many built-in functions that can help you do that!

```
View(batdat)
unique(batdat$species) # Tells you all the unique forms of the data in the specified column ($species)
str(batdat) # Tells you what kind of data is contained in your dataframe
head(batdat) # Returns the first several rows of data in your dataframe
tail(batdat) # Returns the last several rows of data in your dataframe
dim(batdat) # Tells you the size of your dataframe
names(batdat) # Tells you the column names in your dataframe
nrow(batdat) # Returns the number of rows in your dataframe
ncol(batdat) # Returns the number of columns in your dataframe.
```

Add a column to your dataframe

Often times, we want to add additional data to the dataframe we have read into the R environment. This is fairly straightforward, as long as we remember some simple notation. The money symbol ($) is how we tell R exactly which column we are referring to in a dataframe, and parentheses following a word always refers to a function. Below, I will add a column to the batdat dataframe in which I log-transform my fungal load column, gd:

```
batdat$log.loads<-log10(batdat$gdL)
```

Subset data

Another very useful thing to do in R is subsetting data. This allows us to filter down a dataframe to only look at the parts we are interested in. For example, in the batdat dataframe, what if we wanted to make a whole new dataframe that just contains the data from Myotis septentrionalis?

```
MYSE.dat = subset(batdat, species=="MYSE") #a factor/character, so need == and quotes
head(MYSE.dat)
dim(MYSE.dat)
```

As you can see, this new dataframe only contains 12 observations from Myotis septentrionalis. What if we wanted to filter numerical data? For example, what if we only wanted data from bats that were roosting at 6 degrees celsius or warmer?

```
warm.temps = subset(batdat, temp>6) #a number, so no quotes
head(warm.temps)
dim(warm.temps)
```

As you can see here, this new dataframe contains 258 observations, only from bats that were found roosting at 6C or warmer. Often times, you will see code that looks a little different, but accomplishes the same thing by using brackets [ ]. Brackets are a way of telling R exactly which part of a dataframe you are referring to. We can subset our batdat dataframe in the same way we did above, but using the bracket system:

```
MYSE.dat = batdat[batdat$species=="MYSE",]
warm.temps = batdat[batdat$temp>6,]
```

Summarize data using aggregate()

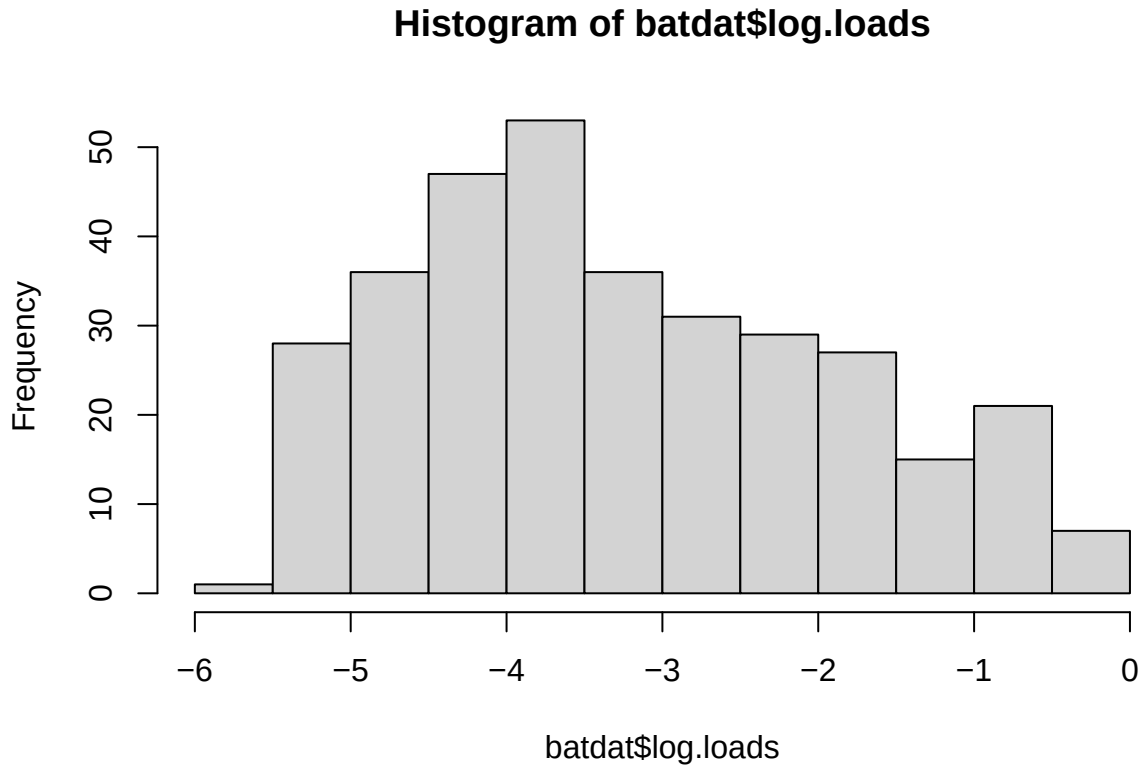
The function aggregate() is an extremely useful function to continue exploring your data. It allows you to summarize your dataset by applying a function to subsets of data individually. For example, say that I wanted to know what the mean log fungal load (batdat$log.loads) was on my bats for each species. The aggregate() function allows me to apply the mean() function (finds the average of data) to find a separate average value for each species:

```
bat.summary = aggregate(log.loads~species, FUN=mean, data = batdat) #aggregate data using the mean to g
bat.summary
```

Make a histogram of a column in your data

Another useful function to visualize the shape of your data is hist(), which will make a histogram with the data that you feed it. In the example below, we will make a histogram of the raw log fungal loads (batdat$log.loads) contained in our batdat dataframe:

```
hist(batdat$log.loads)
```



Write a .csv file

We have created several new dataframes from our original batdat dataframe. If we want to save these new dataframes as a .csv file, the function write.csv() will come in handy. In the example below, I will write the MYSE.dat dataframe as a new .csv file:

```
#write.csv(MYSE.dat, "MYSE_dat.csv", row.names = F)
```