7A. Data Wrangling

Z620: Quantitative Biodiversity, Indiana University

OVERVIEW

In this lesson, we will explore different ways of wrangling data with the goal of making it easier to perform relevant statistics and generate figures on biodiversity data. Specifically, we will introduce traditional **control structures** (e.g., for loops), along with alternative and commonly used tools that are unique to the R statistical environment, such as `apply` functions and `tidyverse` packages. We will use this knowledge to analyze the data generated in `7. Diversity Synthesis` handout to summarize and visualize alpha and beta diversity of our confectionery taxa.

1) DATA WRANGLING

So, what do we mean by **data wrangling**? Most of the data that we will generate or download from publicly available databases will not be in the form we need it to be to conduct our biodiversity analyses. Therefore, most of our data will need to be **wrangled** or manipulated. Much of the power of computing stems from the efficiency of performing small tasks in an iterative and reproducible manner.

2) SETUP

```
rm(list=ls())
getwd()
setwd("~/GitHub/QB-2023/1.Handouts/7.DiversitySynthesis/")
```

A. Retrieve and set up your working directory

```
package.list <- c('vegan','tidyverse','ggplot2','dplyr','broom')
for (package in package.list) {
  if (!require(package, character.only=TRUE, quietly=TRUE)) {
    install.packages(package)
  }
  library(c(package), character.only=TRUE)
}
```

B. Load packages

C. Load data We will be working with data sampled from either an even or uneven community of confectionery taxa. Here, we will load in the .csv file of our site-by-species matrix. The dataset contains 4 sites sampled from an even and uneven **source community** (designated in the row names). A source community is an experimental unit or geographic regions where the entire community originates from.

```
dat = read.csv(file="./data/DataWrangling_Jelly.csv",header = TRUE, row.names = 1)
```

3) WRANGLING CONFECTIONERY DATA FOR BIODIVERSITY

A major hurdle in data science and quantitative biology is learning how to efficiently and reproducibly "wrangle" large date sets. This might involve converting your entries from long to wide formats, or subsetting a data matrix based on the values found in a certain column. Or, maybe you want to summarize a data set by applying a function across rows of your matrix. These types of operations are fairly standard when taking classes in programming or computer science, but are less commonly taught in as part of a life-sciences curriculum.

A. Traditional control structures with alpha diversity In this section, we will introduce some of the basics of **control structures**. Specifically, we will focus on using **for loops**, which have two parts. First, the header assigns a counter or *index* for the variables that will be acted upon. Second, the body contains instructions or operations that will be performed during each iteration.

Remember the `vegan` package that we used to calculate alpha diversity indices. Let's calculate "Shannon entropy" to practice data wrangling. First, we will subset the data by community and calculate Shannon's H. To subset, we will use the `grepl` function in base R, which is a pattern matching function that only returns true values.

```
# Subset the even community
com_e_H <- diversity(dat[grepl("_even",row.names(dat)),], index = "shannon")
print(com_e_H)
```

```
## site1_even site2_even site3_even site4_even
##   3.563075   3.616117   3.543160   3.479093
```

```
# Subset the uneven community
com_u_H <- diversity(dat[grepl("_uneven",row.names(dat)),], index = "shannon")
print(com_u_H)
```

```
## site1_uneven site2_uneven site3_uneven site4_uneven
##     2.964376     3.026223     2.828338     3.008247
```

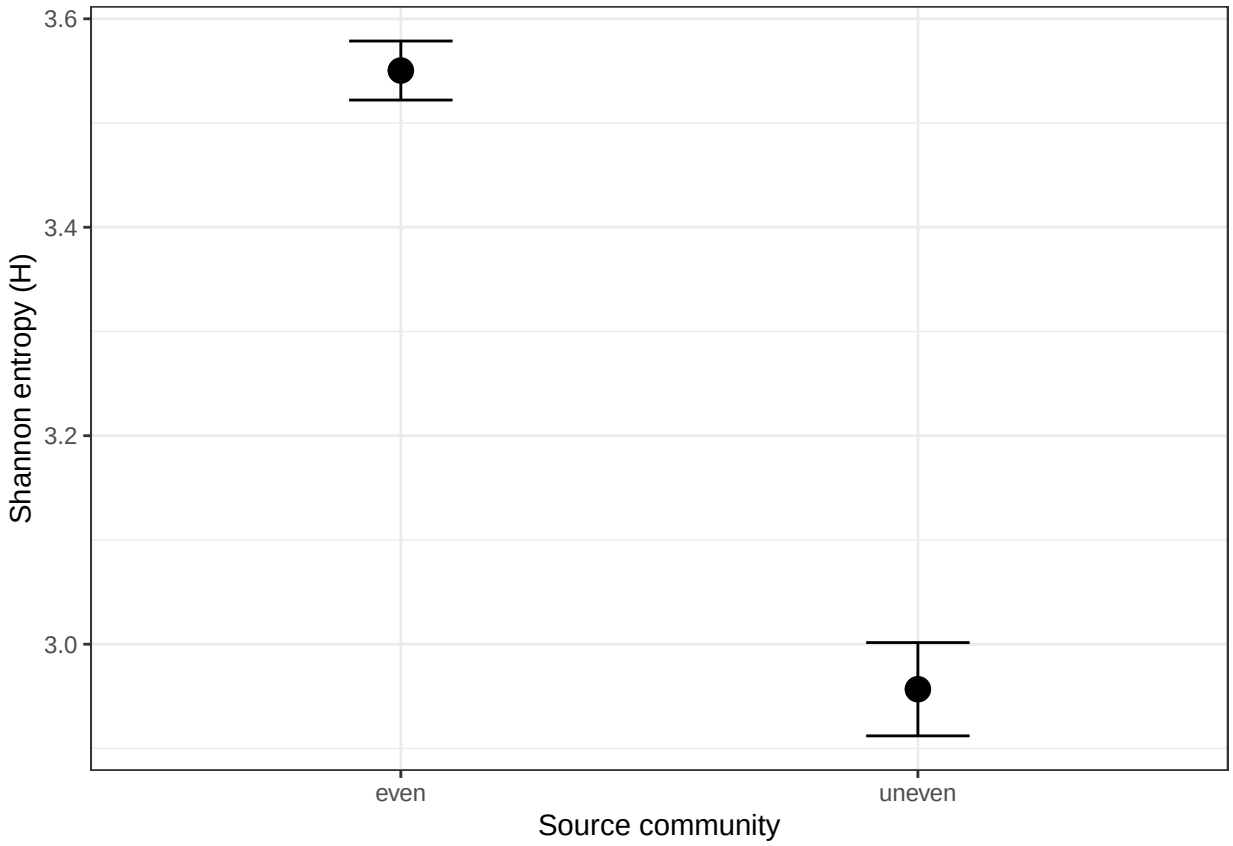
Now, we will use a for loop to obtain diversity estimates across different source communities. We will calculate the mean and standard error (SE) for each community.

```
# write function for SE equation
SE <- function(x) {
  return(sd(x)/sqrt(length(x)))
}

# The following code binds the two vectors as columns in a data frame
com_div_all <- t(cbind.data.frame(com_e_H, com_u_H))
```

```r
# Rename the row names to a more intelligible name
row.names(com_div_all) <- c("even","uneven")

# Create an empty data frame to fill with summary statistics
com_div_sum <- as.data.frame(matrix(ncol = 2 , nrow = 2))
colnames(com_div_sum) <- c("mean", 'se')

# The following for loop will iteratively calculate the mean and standard error for each row, one at a
for(i in 1:2) {
  x = as.numeric(com_div_all[i,])
  com_div_sum[i,1] <- mean(x) # Calculate the mean and save to diversity indices data
  com_div_sum[i,2] <- SE(x)   # Calculate the standard error and save to diversity indices data
}

# Let's add a grouping column to the data
com_div_sum$community <- c("even", "uneven")
```

Now let's plot mean and standard errors of Shannon entropy across the source communities. In QB, most plotting will be done in using **R base package**. However, lots of people like visualizing their data using the package *ggplot*. We invite interested students to explore `ggplot2` package. Let's use this package to graphically explore differences in diversity between different source communities.

```r
ggplot(data = com_div_sum, aes(x = community, y = mean))+
  geom_point(size=4)+
  geom_errorbar(aes(ymin = mean-se, ymax = mean+se), width=.2)+
  ylab("Shannon entropy (H)")+
  xlab("Source community")+
  theme_bw()
```



Take note of how an uneven community, which has more **rare species** - uncommon or infrequently encountered taxa that have low abundances in our sampling effort, influences mean alpha diversity values and the spread of the data (i.e., variance).

B. The `apply` function with alpha diversity Although for loops and other control structures (e.g., while loops, if-then-else) are commonly used by computational biologists as we saw above, there are some downsides to its use. People argue that they lack clarity and can easily break. This can create confusion when the goal is for code to be robust and reproducible in collaborative projects. Also, in R, control structures can be slow and inefficient. Fortunately, there are some alternatives, that we will introduce. Within the base R package, there is a family of **`apply`** functions. This allows one to apply functions across row and columns of matrices, lists, vectors, and data frames. The structure of the `apply` function differs from the for loop. In fact, it can be done with a single line of code, but it has a similar sort of logic. First, you specify the data object of interest. Second, you specify the margin that the function will operate upon (row =1, column = 2, cell = 1:2). Last, you specify the function to use, which can be a native, built-in function (e.g., `mean`) or one that you have written yourself (e.g., `SE`). Run the following code and compare to findings generated with the for loop above:

```
# if only interested in mean, can just write `mean` at end of function
# for both mean and sem, here, we specify as follows:
com_div_sum_apply <- t(apply(com_div_all, 1, FUN = function(x) {c(mean = mean(x), SE = sd(x)/sqrt(lengt
print(com_div_sum_apply)
```

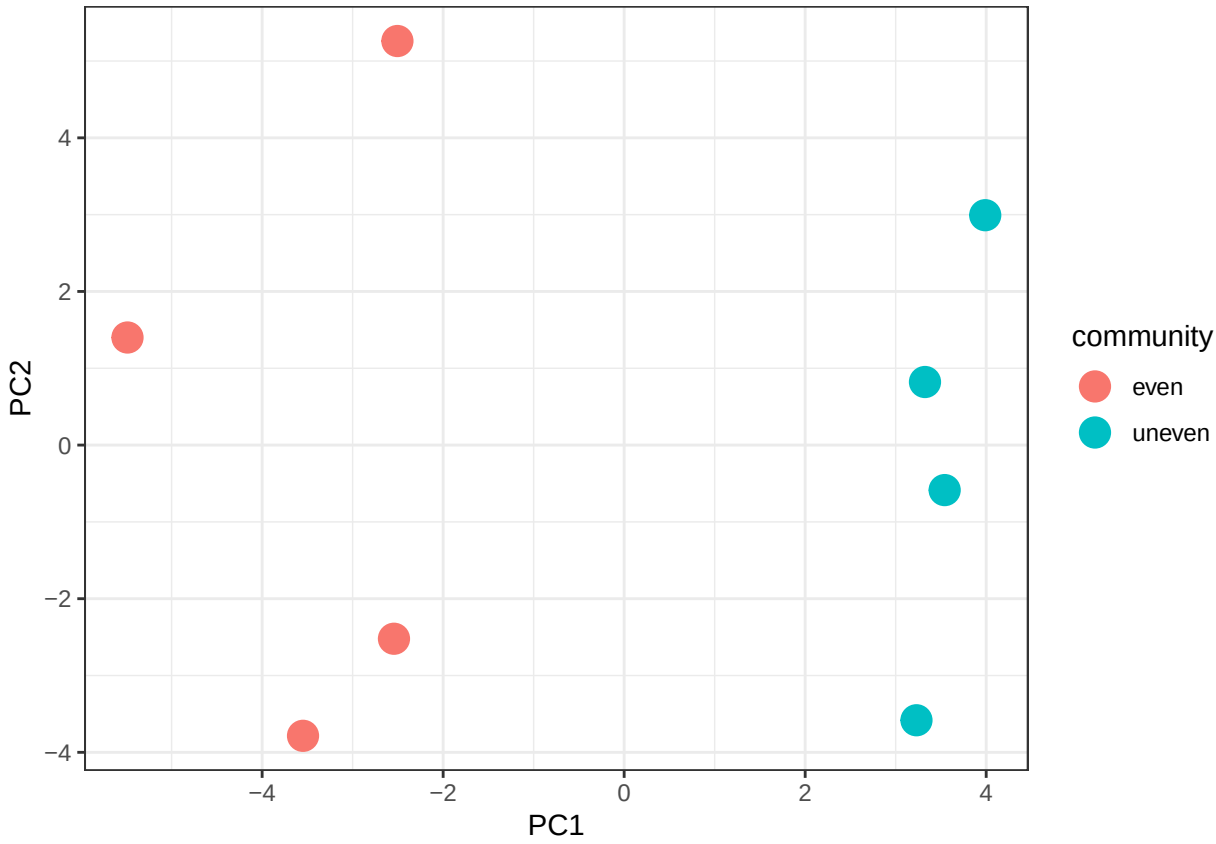
```
##            mean         SE
## even   3.550361 0.02830846
## uneven 2.956796 0.04474587
```

C. The `tidyverse` packages with beta diversity Another popular approach for data wrangling is **`tidyverse`**, which is a collection R packages. The `tidyverse` universe of packages are meant to be simple reproducible functions that simplify the rather complex aspects of wrangling data into fewer lines of code.

Below are the three most commonly used packages, but there are many more! **`dplyr`** contains a set of functions that manipulates data frames. **`tidyr`** allows one to pivot, nest, and separate data in various ways. And **`ggplot2`**, which was introduced above, is the `tidyverse` plotting package based on the "grammar" of plotting data.

Across the `tidyverse` packages, the **pipe operator**, depicted as `%>%`, is a standard transfer command that allows us to string together a sequence of functions or operations and continuously carry on the output. Below, we will use the pipe operator in combination with tidyverse functions and base R principal component function to transform our raw data into PCA scores. Then, we will plot the first and second principal component axes.

```
# Wrangle data and obtain PCA scores
com_mat_tidy <- dat %>%
            # identify data frame of interest
                mutate_if(is.integer, as.numeric) %>%
            # basic statements and operations.here we convert integers to numeric values
                prcomp(center = T, scale = T) %>%
            # run a pca with correlation matrix
                tidy('scores') %>%
            # this is a useful broom function to tidy model outputs as data frames
                filter(PC<3) %>%
            # filter principle components above 2. we need only 2 PCs for a 2D plot
                pivot_wider(names_from = PC, values_from = value) %>%
            # this converts the data to wide format, opposite function is pivot_longer
                mutate(site = rep(c('1','2','3','4'), times = 2)) %>%
            # add a site column for plotting
                mutate(community = rep(c('even', 'uneven'), each = 4)) %>%
            # add a community column for plotting
                rename('PC1' = '1', 'PC2' = '2') %>%
            # you can rename variable names
                select(community, site, PC1, PC2)
            # you can select the columns to work with

# let's plot PCA results
ggplot(data = com_mat_tidy, aes(x = PC1, y = PC2, color = community))+
  geom_point(size = 5)+
  theme_bw()
```



Take note of how each community clusters with one another.