7. 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 visualize data and perform relevant statistics. 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.

1) DATA WRANGLING

Increasingly, biologists are able to ask questions with the use of large, high-throughput data sets. While this creates exciting opportunities, it also requires that one be comfortable and efficient with the handling of such data. This so-called **wrangling** of data might involve converting your entries from long to wide formats, or subsetting a data matrix based on the values found in a certain column. These types of operations are fairly standard when taking classes in programming or computer science. As of late, these skills are being introduced to the life-sciences curriculum.

1) SETUP

```
rm(list = ls())
getwd()
setwd("~/GitHub/QB-2025/1.Handouts/7.DataWrangling/")

# Recall, in Posit.cloud, set your working directory as follows:
#setwd("/cloud/project")
```

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 A **source** community refers to the species found in region that can colonize local sites within the landscape. We will be working with simulated source communities that have an even vs. uneven distribution of species. The dataset contains a site-by-species matrix with four sites from each of the source communities:

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

3) TRADITIONAL CONTROL STRUCTURES

In this section, we introduce **control structures**. Specifically, we focus on **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.

Using the`vegan` package, let us calculate Shannon's entropy ('H). Remember from alpha diversity, this metric takes into account species richness and evenness. First, we will subset the data by source 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
```

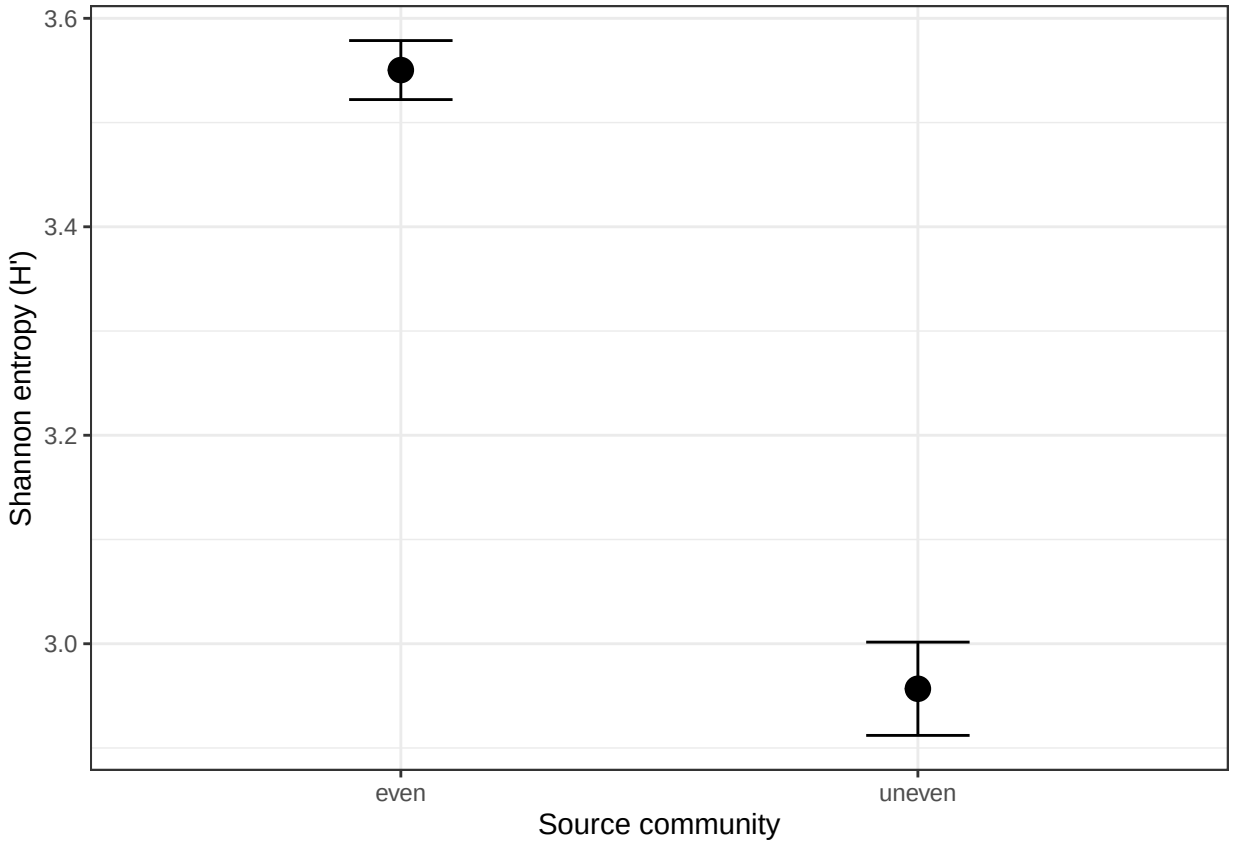
Now, we will use a for loop to obtain diversity estimates across different source communities. Then, we will calculate the mean and standard error (SE) for the two source communities.

```
# write function for SEM
SEM <- 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))

# Rename the row names to be more intelligibl
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", 'sem')

# The following for loop will iteratively calculate
# the mean and standard error for each row, one at a time.
```

```r
for(i in 1:2) {
  x = as.numeric(com_div_all[i,])
  com_div_sum[i,1] <- mean(x) # Calculate mean, save to diversity indices data
  com_div_sum[i,2] <- SEM(x)   # Calculate SEM, save to diversity indices data
}

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

Now let us plot the mean and standard errors of Shannon entropy for the source communities. In QB, most plotting will be done using the **R base package**. Alternatively, we can visualize data using the `ggplot2` package, which interfaces with some popular data wrangling tools that we will introduce below.

```r
ggplot(data = com_div_sum, aes(x = community, y = mean))+
  geom_point(size = 4)+
  geom_errorbar(aes(ymin = mean-sem, ymax = mean+sem), width = 0.2)+
  ylab("Shannon entropy (H')")+
  xlab("Source community")+
  theme_bw()
```



Take note of how **rare species** in an uneven source community influence estimates of alpha diversity.

3) ALTERNATIVE WRANGLING IN R: THE `apply` FUNCTION

Although for loops and other control structures (e.g., while loops, if-then-else) are commonly used, there are some downsides. 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 `apply()` 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., `SEM`). 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), SEM = sd(x)/sqrt(length(x)))}))
print(com_div_sum_apply)
```

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

4) ALTERNATIVE WRANGLING IN R: THE `tidyverse` PACKAGE

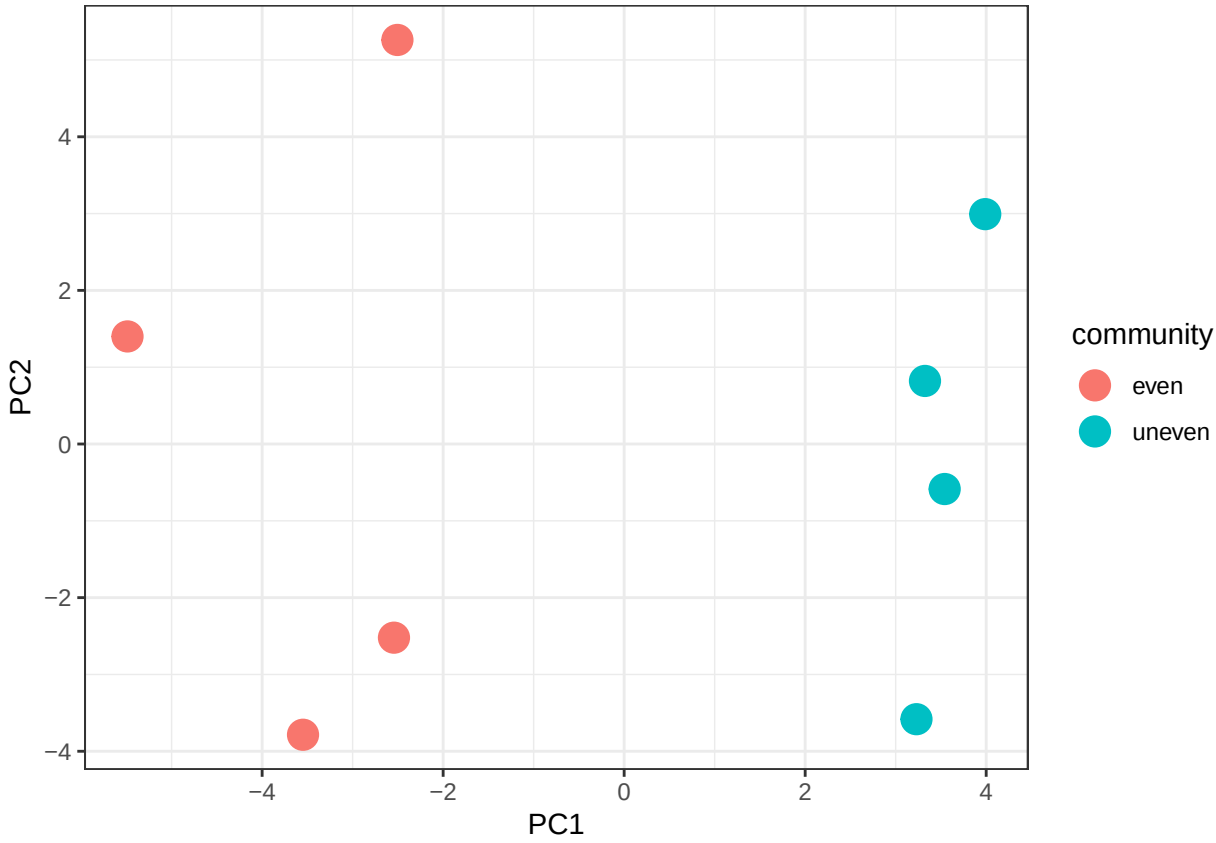
Another popular approach for data wrangling is **tidyverse**, which is a collection R packages. The `tidyverse` universe of packages is meant to simplify the rather complex aspects of wrangling data into fewer lines of code.

Below are the three of the 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 to characterize the beta diversity of the simulated source communities.

```
# Wrangle data and obtain principal components 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 out principle components > 2
            pivot_wider(names_from = PC, values_from = value) %>%
         # 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 us plot ordination results
ggplot(data = com_mat_tidy, aes(x = PC1, y = PC2, color = community))+
  geom_point(size = 5)+
  theme_bw()
```



Take note of how sites and communities cluster.