6. Diversity Sampling and Data Wrangling

Z620: Quantitative Biodiversity, Indiana University

OVERVIEW

Instead of generating biodiversity data from nature, we are going to use gummies to reinforce principles of biodiversity. In the process, you will construct a site-by-species matrix and quantify alpha and beta diversity of our communities. Last, you will test hypotheses about how sampling different community compositions can influence estimates of diversity. The goal of the exercise today is to:

1. Reflect on previous lectures of alpha and beta diversity and test concepts on data we generate ourselves and,
2. Apply several tools to wrangle and analyze data.

##EXERCISE SUMMARY

We will provide you three bags of 200 gummy candies as different species pools. You will divide into three working groups, and every one of you will sample a proportion of individuals. We will provide a species key, so you can easily determine the species and create a vector for a site-by-species matrix. Every group will combine these vectors to asses the differences between each sampling. We constructed these communities with different richness and evenness. So you can compare your results and discuss as a group after analyzing your data. Below we summarized the sampling steps, and provided a guideline for analyzes.

SAMPLING PROTOCOL

1. Divide into three groups. Each of group will have a species pool: A, B, and C. Make sure to keep track of your which community you are sampling from.
2. Each student will randomly sample species using a scoop to obtain approximately 50 individuals. Then, identify the species and quantify their abundances. Each student will report our their own sampling vector to serve as replicate sampling efforts.
3. With the gummies you sampled, we will then generate data manually for a rarefaction curve to understand if we have sufficently captured the source community. To do this, mix your gummy individuals. Then, sub-sample your gummy collection by selecting 5 individuals, count the number of species present, and put your gummies back into your mix.
4. Repeat step 3 multiple times but increase the number of individuals subsamples by five until you have sampled your entire collection.
5. Using Excel, make two spreadsheets that will serve as a site-by-species matrix and your rarefaction curve data. Make sure you include a sample identifier (student name) as well as what community you sampled from. One person should add this to their student repository and make a pull request.
6. Using the data, you will then explore topics of alpha and beta diversity based on the methods you have learned in the rest of the handout and accompanying worksheet. You will use different data wrangling tools explored in this handout to test a hypothesis to explain differences between source communities and sites using the hypothesis testing methods.

DATA WRANGLING OVERVIEW

Now that we have sampled our gummy communities and have generated data, let's actually analyze the data. To show examples of ways to incorporate alpha and beta diversity analyzes with data wrangling, we will use simulated data to show ways to analyze gummy data collected.

1) DATA WRANGLING

So, what do we mean by **data wrangling**? Much of the power of computing stems from the efficiency of performing small tasks in an iterative manner. As part of this lesson, we will explore different ways of manipulating or "wrangling" data with the goal of making it easier to perform relevant statistics and generate figures. 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 learn about these tools while also gaining exposure to R packages that allow us to simulate and sample biodiversity.

2) SETUP

```r
rm(list=ls())
getwd()
setwd("~/GitHub/QB-2021/1.Handouts/7.ControlStructures")
```

```r
package.list <- c('mobsim','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)
}
```

A. Retrieve and set up your working directory

```
## This is vegan 2.6-4
```

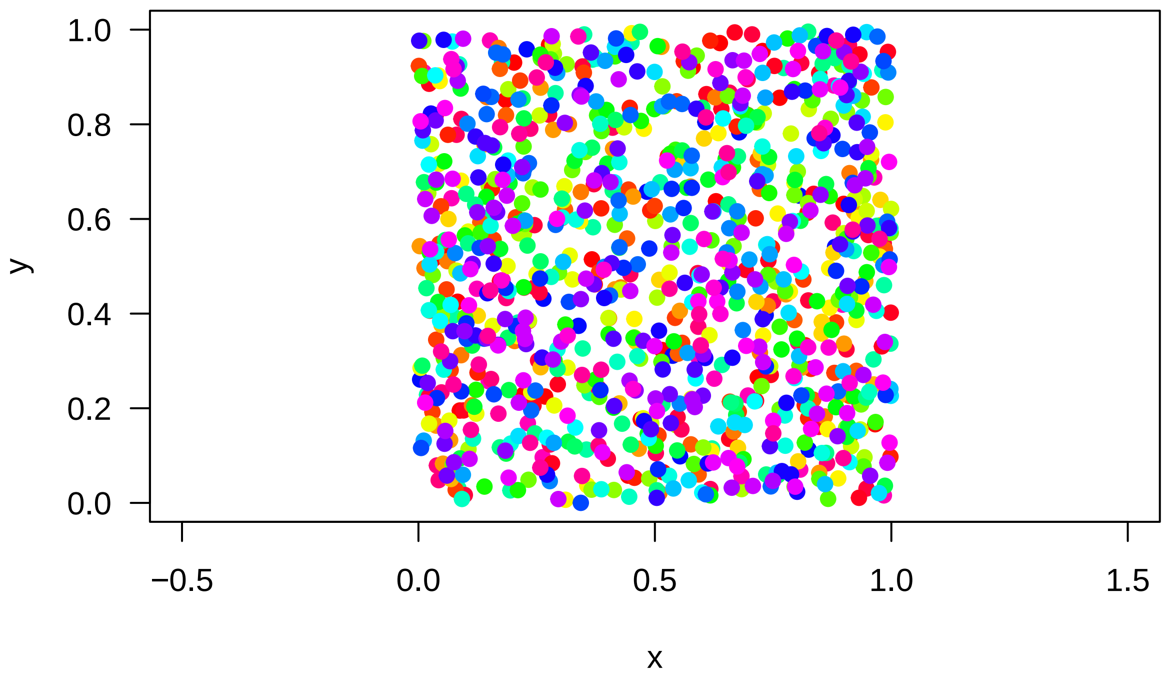
```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   1.0.0
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

B. Simulating Biodiversity To reinforce principles of alpha and beta diversity, we are going to use the `mobsim` package, which was designed for the simulation and measurement of biodiversity across spatial scales: https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/2041-210X.12986. The simulated individuals will mirror what we are doing in the above exercise with gummies.

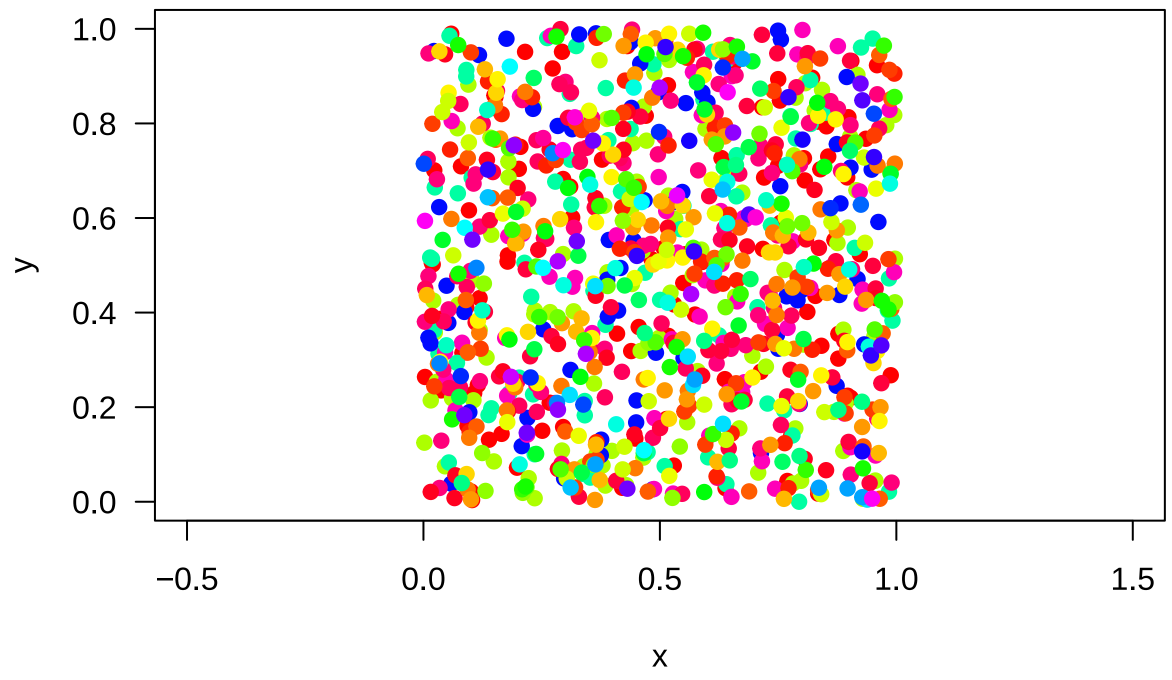
Often, simulations are initiated by taking a random draw of individuals from a *source community*, which is intended to reflect the regional pool of species. In this way, the number of individuals (N) and species (S) can be specified so that we create **local communities** that we will then sample.

Simulate source communitities with random spatial positions The following code simulates at containing 1000 individuals (N) belonging to 50 species (S). These individuals are drawn from source community with a log-normal *species abundance distribution (SAD)* with mean = 2 and standard deviation = 0 (even) and standard deviation = 1 (uneven) communities.

```
# simulate an even local community
com1 <- sim_poisson_community(s_pool = 50, n_sim = 1000, sad_type = "lnorm",
        sad_coef = list("meanlog" = 2, "sdlog" = 0))
# visualize the even local community
plot(com1)
```

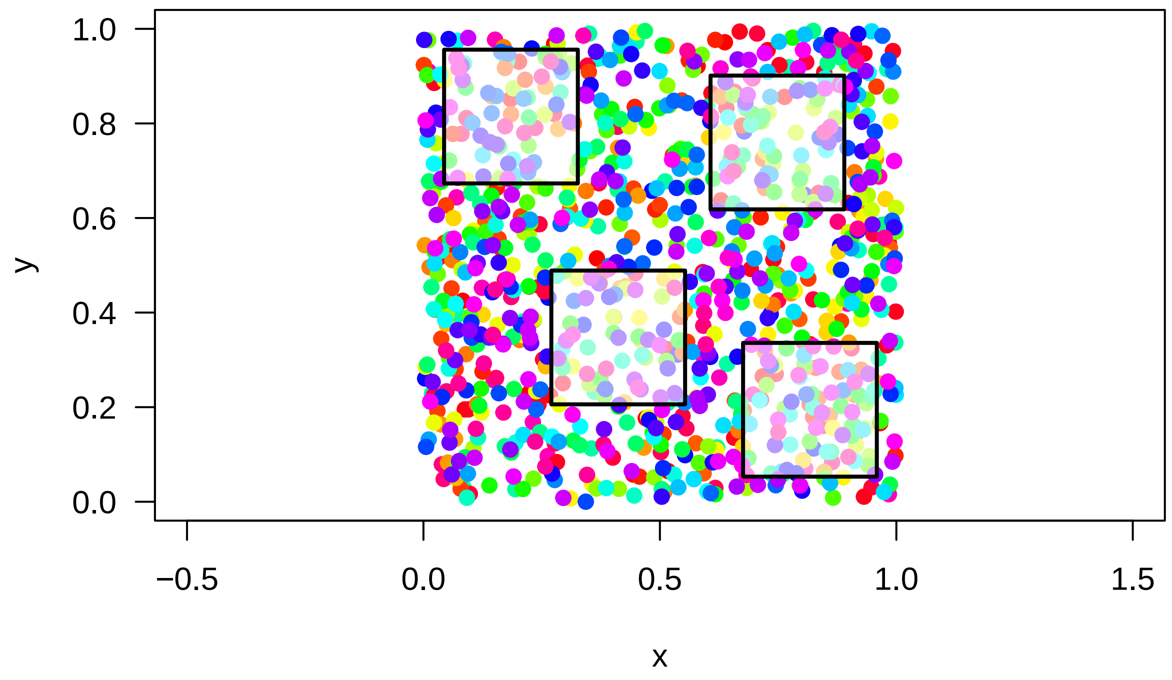


```
# simulate an uneven local community
com2 <- sim_poisson_community(s_pool = 50, n_sim = 1000, sad_type = "lnorm",
        sad_coef = list("meanlog" = 2, "sdlog" = 1))
# visualize the uneven local community
plot(com2)
```

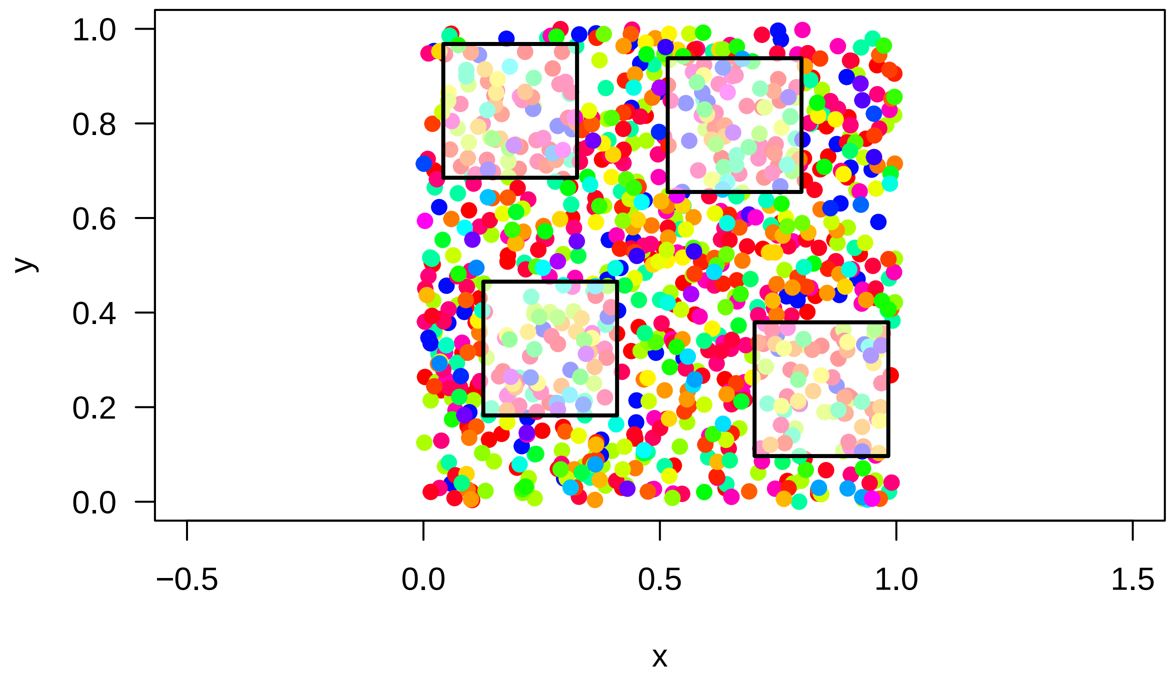


C. Sample species diversity in the simulated community Using functions from the `mobsim` package, we can sample the communities using methods that are commonly used in the field. For example, the `sample_quadrats` function allow us to collect plot-based samples from random, transect, or grid sampling techniques in a spatially explicit census. From the communities we created above, we will now randomly sample 4 different non-overlapping quadrats with area size of 0.08, which represents the proportion of the area sampled in the local community we are interested in.

```
# Lay down sampling quadrants on your communities

com_mat1 <- sample_quadrats(com1, n_quadrats = 4, quadrat_area = 0.08,
             method = "random", avoid_overlap = T)
```



```
com_mat2 <- sample_quadrats(com2, n_quadrats = 4, quadrat_area = 0.08,
                method = "random", avoid_overlap = T)
```



```r
# Assess the species by site matrix generated automatically from our sampling efforts
print(com_mat1$spec_dat)
```

```
##       species1 species10 species11 species12 species13 species14 species15
## site1        1         4         2         4         2         2         1
## site2        1         1         2         1         2         2         1
## site3        2         1         1         2         2         3         0
## site4        2         1         0         2         1         0         0
##       species16 species17 species18 species19 species2 species20 species21
## site1         1         1         0         3        1         1         3
## site2         2         1         3         2        1         3         1
## site3         0         2         0         1        2         2         1
## site4         1         1         1         5        1         3         2
##       species22 species23 species24 species25 species26 species27 species28
## site1         2         1         1         3         3         0         2
## site2         2         0         5         1         0         0         0
## site3         3         0         1         1         2         2         1
## site4         1         1         6         1         3         2         1
##       species29 species3 species30 species31 species32 species33 species34
## site1         3        0         2         3         3         2         3
## site2         2        2         3         1         0         0         0
## site3         0        2         1         0         1         1         1
## site4         2        0         1         2         2         4         0
##       species35 species36 species37 species38 species39 species4 species40
## site1         0         0         1         1         1        1         3
## site2         0         1         1         1         1        2         2
## site3         1         2         7         1         1         0         2
## site4         3         0         0         0         2         2         1
##       species41 species42 species43 species44 species45 species46 species47
## site1         0         1         0         2         1         3         3
## site2         1         1         3         2         3         3         1
## site3         2         2         1         2         1         0         2
## site4         1         2         2         1         0         1         1
##       species48 species49 species5 species50 species6 species7 species8
## site1         3         4        4         2        1        0        2
## site2         0         2        0         3        0        2        0
## site3         1         2        0         4        1        2        1
## site4         1         2        1         4        0        2        0
##       species9
## site1        2
## site2        1
## site3        0
## site4        4
```

```
print(com_mat2$spec_dat)
```

```
##       species1 species10 species11 species12 species13 species14 species15
## site1        3         0         0         0         5         1         2
## site2        7         2         4         4         1         5         1
## site3        9         6         3         2         2         1         2
## site4        8         3         3         0         1         2         0
##       species16 species17 species18 species19 species2 species20 species21
## site1         2         3         1         0        6         1         2
## site2         2         4         2         1        8         0         0
## site3         1         2         0         2        6         0         0
## site4         1         1         2         1        5         0         1
##       species22 species23 species24 species25 species26 species27 species28
## site1         0         2         0         2         0         1         0
## site2         1         0         0         0         1         1         0
## site3         1         1         1         0         0         0         1
## site4         2         0         0         0         3         3         1
##       species29 species3 species30 species31 species32 species33 species34
## site1         0       10         0         1         0         2         0
## site2         3        3         0         1         0         0         0
## site3         0        6         1         0         1         0         0
## site4         1        5         1         1         0         0         1
##       species35 species36 species37 species38 species39 species4 species40
## site1         0         1         0         1         1        3         0
## site2         0         0         0         0         0        6         1
## site3         0         1         0         0         0        4         0
## site4         1         0         0         0         1        4         1
##       species41 species42 species43 species44 species45 species46 species47
## site1         0         0         0         1         1         1         0
## site2         1         1         0         0         0         0         0
## site3         1         0         0         1         0         0         0
## site4         1         0         0         1         0         0         0
##       species48 species49 species5 species50 species6 species7 species8
## site1         0         0        4         0        3        5        6
## site2         0         0        5         0        3        1        5
## site3         1         1        2         1        3        8        2
## site4         1         0        3         0        8        4        2
##       species9
## site1        6
## site2        3
## site3        3
## site4        1
```

D. Measure within sample diversity Remember the `vegan` package that we used to calculate diversity indices. Let's calculate "Shannon entropy" to practice data wrangling.

```
com1_div <- diversity(com_mat1$spec_dat, index = "invsimpson")
print(com1_div)
```

```
##    site1    site2    site3    site4
## 34.29004 30.32484 28.82353 28.03883
```

```
com2_div <- diversity(com_mat2$spec_dat, index = "invsimpson")
print(com2_div)
```

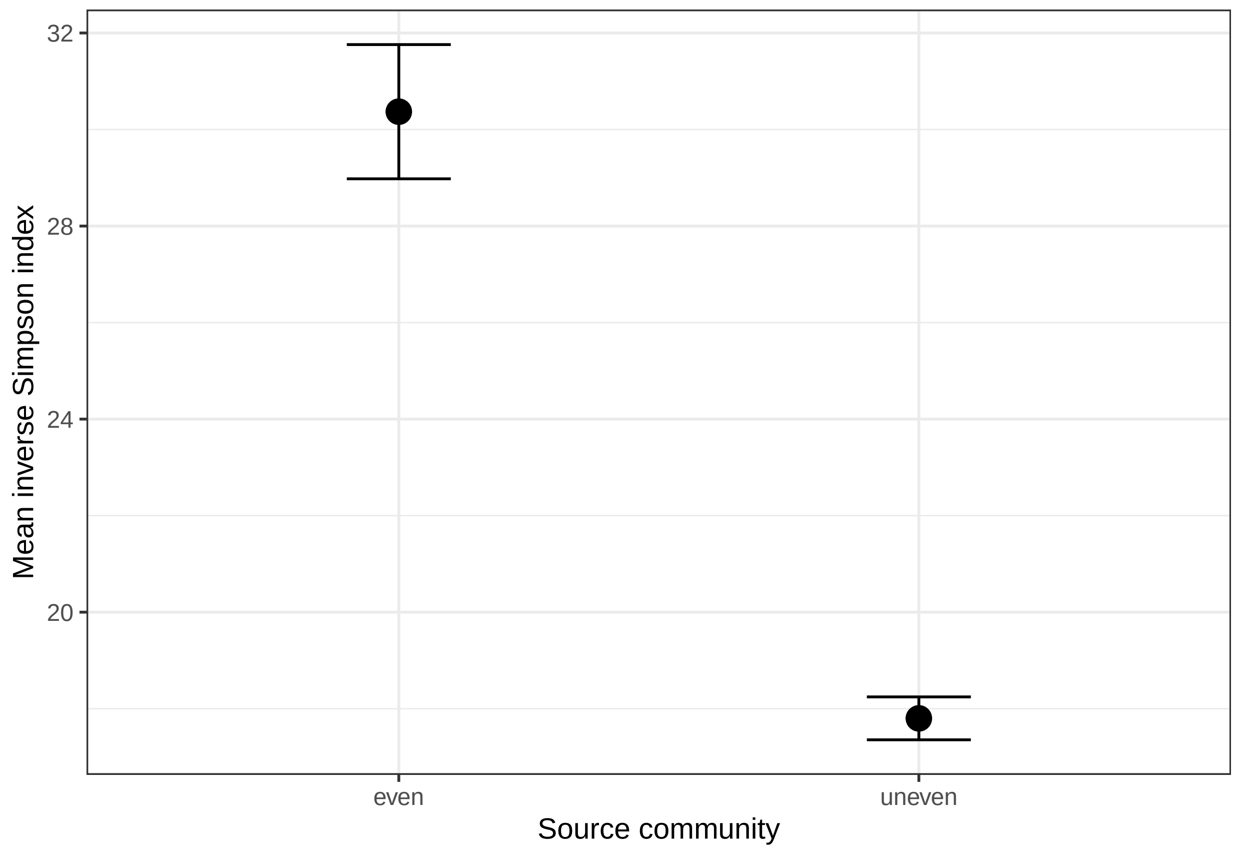
```
##    site1    site2    site3    site4
## 17.18551 17.91239 17.08876 19.01389
```

Summarize diversity data with a for loop Now, we will use control structures, specifically, 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(com1_div, com2_div))

# 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("Mean inverse Simpson index")+
  xlab("Source community")+
  theme_bw()
```



Summarize diversity data with the `apply` function 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:

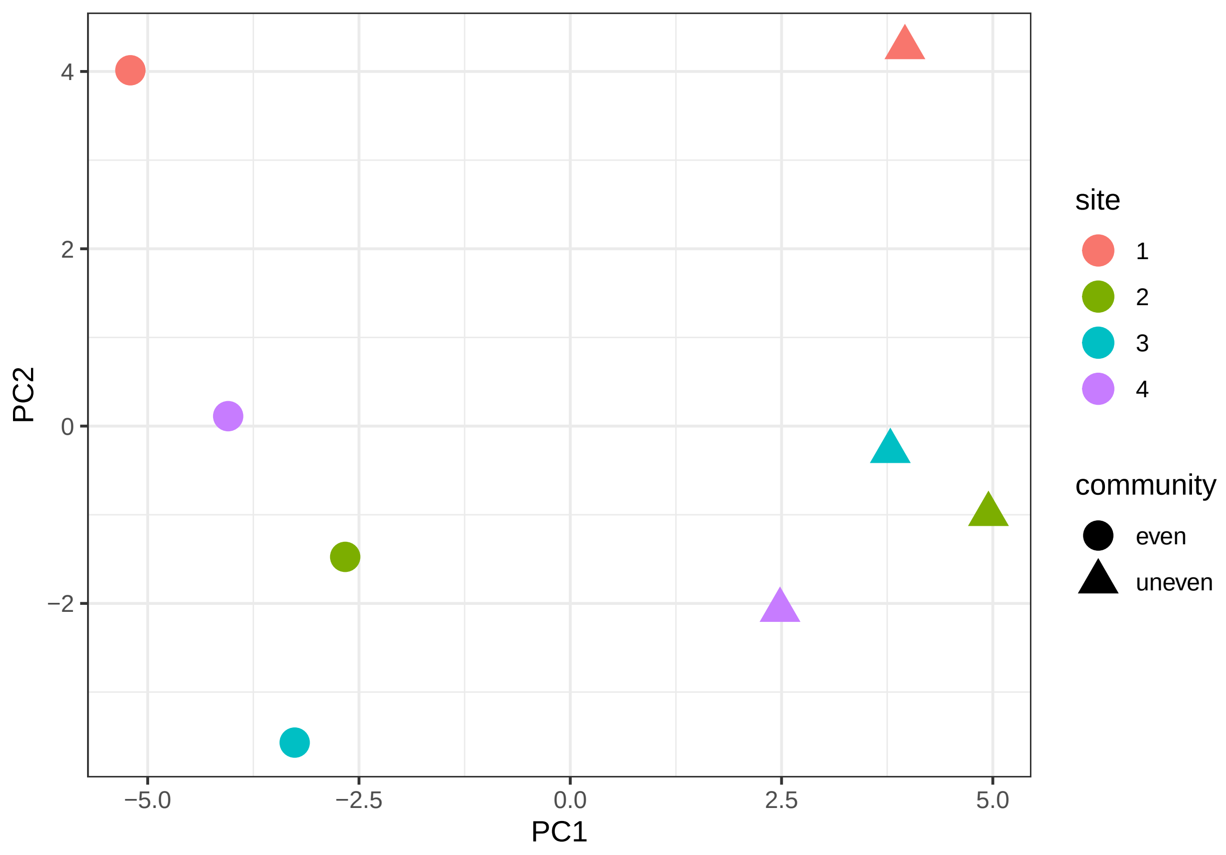
```r
# if only interested in mean, can just write `mean` at end of function
# for both mean and sem, here, we specify as follows:
com_div_sum_apply <- apply(com_div_all, 1, FUN = function(x) {c(mean = mean(x[x>0]), SE = sd(x[x>0])/sq
print(com_div_sum_apply)
```

```
##       com1_div   com2_div
## mean 30.369312 17.8001351
## SE    1.390284  0.4443745
```

E. Measure between sample diversity We have seen different multivariates statistical approaches to compare biodiversity between samples. Here we will incorporate an ordination to measure the differences between our even and uneven communities.

Wrangle data with `tidyverse` Another popular approach for data wrangling is **`tidyverse`**, which is a collection R packages. **`dplyr`** contains a set of functions that manipulates data frames. **`tidyr`** allow one to pivot, nest, and separate data in various ways. And **`ggplot2`** is an alternative to plotting in base R that was designed to interface with `tidyverse`. `tidyverse` uses the **pipe operator**, depicted as `%>%`, to string together a sequence of functions or operations. Below, we will use the pipe operator in combination to `shape` and `summarize` our diversity data, before making a plot.

```
# Wrangle data to plot PCA results
com_mat_tidy <- com_mat1$spec_dat %>% # identify data frame of interest
                bind_rows(com_mat2$spec_dat) %>% # bind with another dataset
                replace(is.na(.), 0) %>% # you can replace NAs with 0 abundance
                mutate_if(is.integer, as.numeric) %>% # basic operations
                prcomp(center = T, scale = T) %>% # run a pca
                tidy('scores') %>% # this is a useful broom function
             # to tidy model outputs as data frames
                filter(PC<3) %>% # filter principle components above 2
                pivot_wider(names_from = PC, values_from = value) %>%
             # this converts the data to wide formate, opposite 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)

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


PC2
4
2
0
−2
PC1
−5.0
−2.5
0.0
2.5
5.0
site
1
2
3
4
community
even
uneven