## Lab: Do the bees at hedgerows and weedy field margins (controls) have the same distribution of traits?

```{r lab4_setup, include=FALSE}
# For reproducibility purposes
set.seed(123)
knitr::opts_chunk$set(error = TRUE, warning = TRUE, message = TRUE)
suppressPackageStartupMessages({
  if (requireNamespace("tidyverse", quietly = TRUE)) library(tidyverse)
})
```

 Our goal is to reproduce the analyses in Ponisio et al. 2016, On-farm habitat restoration counters biotic homogenization in intensively managed agriculture, [Ponisio et al. 2016](https://doi.org/10.1111/gcb.13117)

Specifically, we are going to reproduce Figure 4 panels a-b, where we examine whether the bees at hedgerows (restored habitat) and controls (weedy field margins that would have a hedgerow) are random subset of the species pool.

**Conservation/ecology Topics**

- Become familiar with different metrics of biodiversity, including trait diversity.
- Become familiar with the concept of biotic homogenization.

**Computational Topics**

- Join data
- Make histograms
- Subsetting, reshaping data using dplyr (more practice)
- Use logical indexing to subset data (more practice)
- Write functions (more practice)
- Write for loops (more practice)

**Statistical Topics**

- AB testing, simulating the null hypothesis
- Calculate empirical p-values
- Hypothesis testings: reject or accept our null hypothesis

-------------------------------

```{r}
# Load necessary libraries in R
library(tidyverse)

# For reproducibility purposes
set.seed(123)
```

```{r}
## read in the specimen data
spec <- read.csv("data/specimens-complete.csv")

## explore the columns
str(spec)
```

Each row is a uniquely identified insect specimen and all of it's attributes. If we explore the species ID-related columns in more depth, we can see that there are many different types of insects in addition to bees. There are also some specimens that have never been IDed (missed data is a blank cell in these case).

```{r}
unique(spec$Order)
unique(spec$Family)
sort(unique(spec$GenusSpecies))
```

~~~~
```

### Question 1: Practicing the data subsetting tools from dplyr

- 1a. Thinking back to the dplyr tools you learned last week, subset the data to only
bees. The families of bees are "Halictidae", "Megachilidae", "Apidae", "Colletidae",
"Andrenidae" and "Melittidae". In addition, drop all rows without anything in the
'Species' column, otherwise we could end up with specimens only identified to Genus.

Keep the data object named spec (i.e., don't create a new name for the subsetted data.)

HINT: %in% maybe useful, or remember how to string together Booleans in 'dplyr'.

```{r}
library(dplyr)
library(tidyr)

spec$Species[spec$Species == ""] <- NA

beesOnly <- spec %>%
  filter(Family %in% c("Halictidae", "Megachilidae", "Apidae", "Colletidae",
"Andrenidae", "Melittidae")) %>%
  drop_na(Species)

### The dimensions of the subsetted data should be 99097 by 40
dim(beesOnly)
```

- 1b. Now take a look at the site statuses, or the site "treatments",

```{r}
unique(beesOnly$SiteStatus)

```

This data include many more site treatments than we need to reproduce the publication
because it includes all of the sites ever surveyed in this landscape. We don't really
care about the restored sites (that is the year the hedgerow was planted so they are
between a hedgerow and a control). We don't have many "natural" sites  (we couldn't
find any). Forbs are hedgerows with additional forb plantings, also not what we need.

- 1c. Subset only to mature hedgerows and controls (weedy field margins). We also only
care about the "hand-netted" data (the column 'NetPan') and female bees ('Sex') (male
bees don't collect pollen). Do all of this subsetting in one call to filter (you will
need to combine your comparisons using & or |.

Similar to before keep the name of the data the same. The publication also includes the
maturing sites, but we are going to make our lives a little easier by keeping our
analysis to two treatments.

```{r}

beesOnlytemp <- beesOnly %>%
  filter(SiteStatus %in% c("control", "mature"), NetPan %in% c("net"), Sex %in% c("f"))

### The dimensions of the subsetted data should be 8810 by 40.
dim(beesOnlytemp)
```

- 1d. Lastly, drop all the random columns we don't care about. All we want to keep are
"UniqueID", "GenusSpecies", "Site", "SiteStatus" and "Year".

```{r}
### let's drop all the random columns we don't care about
beesOnly_compact <- beesOnlytemp %>%
select(UniqueID, GenusSpecies, Site, SiteStatus, Year)
```
~~~~

~~~
### Demo: Joining data

Next, we want to add in species traits.

```{r}
### load the traits table, this includes traits for all the bees
traits <- read.csv('data/bee.csv')
```

We will often find ourselves in a situation where we want to combine two datasets by
some shared column. In dyplyr, mutating joins add columns from y to x, matching
observations based on the keys. There are four mutating joins: the inner join, and the
three outer joins.

- Inner join: An inner_join() only keeps observations from x that have a matching key
in y. The most important property of an inner join is that unmatched rows in either
input are not included in the result, so data is dropped.

```{r}
df1 <- data.frame(x = c(1:4, NA), y = 2)
df2 <- data.frame(x = c(1:2), z = 3)
head(df1)
head(df2)

inner <- inner_join(df1, df2)
dim(inner)
head(inner)
```

- Outer joins: The three outer joins keep observations that appear in at least one of
the data frames:

- A left_join() keeps all observations in x.

```{r}
df1 <- data.frame(x = c(1:100), y = 2)
df2 <- data.frame(x = c(1:10), z = 3)
head(df1)
head(df2)

left <- left_join(df1, df2)
dim(left)
head(left)
```

- A right_join() keeps all observations in y.

```{r}
df1 <- data.frame(x = c(1:100), y = 2)
df2 <- data.frame(x = c(1:10), z = 3)
head(df1)
head(df2)

right <- right_join(df1, df2)
dim(right)
head(right)
```

- A full_join() keeps all observations in x and y.

```{r}
df1 <- data.frame(x = c(1:100), y = 2)
df2 <- data.frame(x = c(1:10, 100:200), z = 3)
df1
df2

full <- full_join(df1, df2)
dim(full)
~~~

~~~
head(full)
```

Class discussion: What type of join do we want to use to to combine the individual bee
data with their traits?

### Question 2

- 2a. Join the trait data to the individual bee data. Before joining the data, subset
the trait to only the bee IDs ('GenusSpecies'), body size ('MeanITD'), floral
specialization ('d').  Call the joined data spec.traits


```{r}
### your code here
smalltraits <- traits %>%
  select(GenusSpecies, MeanITD, d)

spec.traits <- beesOnly_compact %>%
left_join(smalltraits, by = c("GenusSpecies"))

### The dimensions should be 8810    7
dim(spec.traits)
```

- 2b. How many specimens are missing body size data?
HINT: the function 'is.na()' may be useful.
EXTRA HINT: Remember the TRUEs are counted as 1s in R.

```{r}
### your code here

sum(is.na(spec.traits$MeanITD))

```

- 2c. What are the unique species names missing body size data.
```{r}
### your code here
unique(spec.traits$GenusSpecies[(is.na(spec.traits$MeanITD))])

```

- 2d. These were relatively rare species that didn't have enough individuals to
accurately estimate body size. Since we cannot do much with them, go ahead and drop
them. Keep the data named spec.traits. You can use the tools we have learned in dyplyr,
or just clever indexing.

HINT: There are several ways you can drop the rows without body size data, try to avoid
hard coding by copy pasting the names above. What if you added new data and wanted to
re-run the code? You might miss some species if you remove them "by hand".
EXTRA HINT: Remember that ! reverses TRUEs and FALSEs (so TRUE because FALSE and vice
versa).

EXTRA EXTRA HINT: If you decide to use indexing, remember that when we index a 2D
object we need to specify the indexes of the rows and columns.
```{r}
### your code here
spec.traits <- spec.traits %>%
  drop_na(MeanITD)

### The dimensions should be 8634 7
dim(spec.traits)
```

### Lab Question 3

- 3a. Use 'group_by' and 'summarize' to calculate the average body size and
~~~

~~~
specialization for each site, year, site status combination. Create a new object called
sys.traits

```{r}
### your code here
sys.traits <- spec.traits %>%
  group_by(Site, Year, SiteStatus) %>%
  summarize(avg_bodysize = mean(MeanITD),
            avg_specialization = mean(d)
            )
```

- 3b. Visualize this data with a histograms for each trait.
HINT: 'geom_histogram()' is the function that makes histograms, and it only needs an x
aesthetic.
```{r}
### body size
library(ggplot2)
ggplot(sys.traits, aes(x = avg_bodysize)) +
  geom_histogram(bins = 15) + labs(x = "Average Body Size")
### your code here
```


```{r}
### Specialization
### your code here
ggplot(sys.traits, aes(x = avg_specialization)) +
  geom_histogram(bins = 15) + labs(x = "Average Specialization")
```
- 3c. Describe what you see.
The distribution for Average body size, is very similar to the distributuon of average
specialisation, and thus suggests that body size and specialisation have a strong
correlation.

### Demo: Do the bees at hedgerows and weedy field margins (controls) have the same
distribution of traits?

- Null hypothesis: Mature hedgerows and weedy field margins (controls) support bees
with similar traits, so the average trait values at hedgerows and controls is similar.
- Alternative hypothesis: The traits at mature hedgerows and controls are not the same.
 a) Mature hedgerows have large bees and more specialized bees than would be expected
if we were sampling randomly from the regional species pool.
 b) Controls  have smaller bees and more generalized bees than would be expected if we
were sampling randomly from the regional species pool.


In the study, we tested the hypothesis above by creating a null distribution of average
trait values by shuffling individuals across sites within a year. This enabled us to
constrain species abundance and richness at a site while changing the identity of the
individuals. Creating this type of custom null model is a bit out of the scope of what
we know how to do at this point in the class, so instead we will simplify our problem
by shuffling the treatment columns (mature hedgerow vs. control) across sites within a
year. We will then take the average trait value for the randomized data to calculate
the null expectation of controls b

**Step 0**: Let's first try to do our randomization with 1 year of data then work out
way up.
```{r, eval=FALSE}
## Let's pick 2013, it was a nice year.
year1 <- sys.traits %>%
  filter(Year == 2013)

head(year1)
```

**Step 1:**Shuffle the treatment column
~~~

~~~
```{r, eval=FALSE}
## step 1) year1.sample shuffles a column

year1.shuffle <-  sample(year1$SiteStatus,
                         nrow(year1), replace=FALSE)

year1.shuffle
```

**Step 2:** Add the shuffled column back on to the table
```{r, eval=FALSE}
year1$ShuffleSiteStatus <-  year1.shuffle
head(year1)
```

**Step 3:** Take the mean of the simulated data.

```{r, eval=FALSE}
shuffle.traits <- year1 %>%
  group_by(Year, ShuffleSiteStatus) %>%
  summarise(d=mean(d),
            MeanITD = mean(MeanITD))
```

### Lab Question 4

- 4a. Combine Steps 1-3 into a function: Now that we have worked through all the steps
with one year, write a function that takes yr and sys.traits. Don't forget to add a
nice annotation in the first few line describing what the function does. Call your
function 'randomizeTraits'

```{r}
randomizeTraits <- function(yr, sys.traits) {
# filter for one year
  just.yr <- sys.traits %>%
    filter(Year == yr)
  # shuffle site status for jus† one year
  ShuffledSiteStatus <- sample(just.yr$SiteStatus,
                  nrow(just.yr), replace = FALSE)
  #
  just.yr$ShuffledSiteStatus <- ShuffledSiteStatus
  shuffled.means <- just.yr %>%
    group_by(Year, ShuffledSiteStatus) %>%
    summarise(AVGrandomSpec = mean(avg_specialization), AVGrandomBody =
mean(avg_bodysize))
  return(shuffled.means)
}
```
- 4b.  Test out your function on a 2013. Run your function a few times. Do the mean
values for the traits change? Why?
```{r}
randomizeTraits(2013,sys.traits)
```
Yes, they change each time. They are randomly selected from the shuffled sites and
changed each time because there is no seed set so each time the function is run it
creates a new distribution from a different randomization, using a different mean for
both traits.

### Question 5
Now we want to use our function on each year of data and save our results. The perfect
situation for a for loop!

- 5a. We will want to loop over the different years in the data. Start by creating a
vector of the unique year values, call it 'years'. It would be nice to sort that vector
as well so that the years are in order.

```{r}
### your code here
~~~

~~~~
years <- sort(unique(sys.traits$Year))
```

- 5b. We will do this together in class, but first try to think it through.

The next step was a bit tricky. Unlike before in class (lab 1),  where we saved the
calculations from our function into a vector, we have a dataframe  I found that the
easiest way to get the column names right was to start to the first year of data, then
rbind() (row bind) that to the previous year. Call your randomized data random.traits

HINT: If you do as I suggested above, you will not want to include year one in your for
loop. If you have a vector x, x[-1] will drop the first element of that vector.


```{r}
random.traits <- randomizeTraits(yr=years[1],sys.traits)

for(t in years[-1]){
  new.year <- randomizeTraits(yr=t, sys.traits)
  random.traits <- rbind(random.traits, new.year)
}
dim(random.traits)
```

- 5c. Take the mean body size and specialization for each site status across years of
that simulated full dataset, call it mean.years

```{r}
mean.years <- random.traits %>%
  group_by(ShuffledSiteStatus) %>%
  summarize(AVGrandomSpec = mean(AVGrandomSpec), AVGrandomBody = mean(AVGrandomBody))

mean.years
```

- 5d. Now combine all the steps above into a single function called repRandomComm that:
- shuffles year 1 using the randomizeTraits function
- loops over the remaining years, row binding the data to prior year each time
- takes the mean of the traits across years, and returns that value.

It should have two arguments years and sys.traits

NOTE: It isn't A+ to have argument names be the same as objects in your Global
Environment (can you think of why?), but sometimes argument names can just too silly if
you keep trying to tweak them. Let's just be kinda A- with our naming to avoid
confusion.

```{r}
repRandomComm <- function(years, sys.traits){

  random.traits <- randomizeTraits(yr=years[1],sys.traits)

for(t in years[-1]){
  new.year <- randomizeTraits(yr=t, sys.traits)
  random.traits <- rbind(random.traits, new.year)
  mean.years <- random.traits %>%
  group_by(ShuffledSiteStatus) %>%
  summarize(AVGrandomSpec = mean(AVGrandomSpec), AVGrandomBody = mean(AVGrandomBody))
}
  return(mean.years)
  }

```

Try out your function!

```{r, message=FALSE}
repRandomComm(years,sys.traits)
~~~~

- 5d. Last step! Run your randomization 100 times using a for loop. Similar to our above for loop, it was helpful to run the randomization once to get the format of the dataframe correct, then just rbind the rest of the iterations onto that.

```{r, message=FALSE}
#set up number of interations
niter <- 100
randomized.comms <- repRandomComm(years, sys.traits)
#set up for loop
for(i in 1:(niter-1)){
  new.comm.means <- repRandomComm(years, sys.traits) #define new mean collumn
  randomized.comms <- rbind(randomized.comms, new.comm.means) #join data frame
}

### The dimensions should be 200 3 (two rows are created each iteration)
dim(randomized.comms)
```

###  Demo: Interpreting the results
To determine whether our randomized communities have similar trait values as our observed communities, we will calculate an empirical p-value.

Empirical p-value is calculated as the number of permutation tests with a 'more extreme' (larger or smaller, depending on the null hypothesis) observed test statistic.

In our case, our test statistic is the mean body size (MeanITD) and mean specialization (d).

We will start by calculating our observed test statistic for both traits.

```{r, eval=FALSE}
obs.means.yr <- sys.traits %>%
  group_by(Year, Site, SiteStatus) %>%
  summarise(avg_specialization=mean(avg_specialization),
            avg_bodysize = mean(avg_bodysize))

obs.means <- obs.means.yr %>%
  group_by(SiteStatus) %>%
  summarise(avg_specialization=mean(avg_specialization),
            avg_bodysize = mean(avg_bodysize))


obs.means
```

To visualize our observed data in relation to our randomized data, we will plot a histogram of our simulated data and add a vertical line for our observed data.

```{r, eval=FALSE}
ggplot(randomized.comms, aes(x=AVGrandomBody, fill=ShuffledSiteStatus)) +
  geom_histogram(bins = 12) +
    geom_vline(obs.means,
    mapping =aes(xintercept =avg_bodysize, linetype = SiteStatus)) + labs(x="Avergae
Body Size shuffled")

```
Our observed data looks quite different than our randomized data!

If our null hypothesis that bees at mature hedgerows and controls have similar traits is true what is the probability we get bees as small as we did at the controls? Let's calculate that probability. We need to calculate how many times we got values equal to or more extreme than our observed value when we randomized the treatments (mature hedgerow, control) in the data.

Remember that we can sum the TRUEs.
```{r, eval=FALSE}
```

~~~
## The base R way
sum(randomized.comms$AVGrandomBody[
  randomized.comms$ShuffledSiteStatus == "control"] <=
obs.means$avg_bodysize[obs.means$SiteStatus == "control"])/
  sum(randomized.comms$ShuffledSiteStatus == "control")

## or the tidyverse way
subsetted.randomized <- randomized.comms %>%
  filter(ShuffledSiteStatus == "control") %>%
  select(AVGrandomBody)

obs.body.size <- obs.means %>%
  filter(SiteStatus == "control") %>%
  select(avg_bodysize)

sum(subsetted.randomized <= obs.body.size$avg_bodysize)/
  nrow(subsetted.randomized)
```

This is our empirical p-value. That is a pretty low probability that we would get our
observed statistic if the null hypothesis (the traits in mature and control communities
the same) was true.

We generally reject our null hypothesis if our p-value is less than 0.05.  We can
therefore reject our null hypothesis.

Now for the mature communities:

```{r, eval=FALSE}
sum(randomized.comms$AVGrandomBody[
  randomized.comms$ShuffledSiteStatus == "mature"] >=
obs.means$avg_bodysize[obs.means$SiteStatus == "mature"])/
  nrow(randomized.comms)

```

Again, that is a pretty low probability that we would get our observed statistic if the
null hypothesis (the traits in mature and control communities the same) was true. Our p
value is again less than 0.03. We can therefore reject our null hypothesis.

###  Question 6

Follow the same steps for the specialization (d)
- 6a. Make a histogram with the randomized and observed values.
```{r}
ggplot(randomized.comms, aes(x=AVGrandomSpec, fill=ShuffledSiteStatus)) +
  geom_histogram(bins = 12) +
    geom_vline(obs.means,
    mapping =aes(xintercept =avg_specialization, linetype = SiteStatus)) +
labs(x="Avergae Specialization shuffled")

```

- 6b.  What is the probability that bees at controls are as generalized (so small
values of d) as what we observed, given then null hypothesis is true?
```{r}

# I like the tidyverse way :)
random_subset <- randomized.comms %>%
  filter(ShuffledSiteStatus == "control") %>%
  select(AVGrandomSpec)

obs.spec <- obs.means %>%
  filter(SiteStatus == "control") %>%
  select(avg_specialization)

sum(random_subset <= obs.spec$avg_specialization)/
~~~

~~~
  nrow(random_subset)
```

- 6c. Do we accept or reject our null hypothesis?
We can reject our null hypothesis because the p value is very low (almost 0), and
therefore there is little probability that we would get our observed statistic if the
null hypothesis (the traits in mature and control communities the same) was true.

- 6d.  What is the probability that bees at mature hedgerows are as specialized (so
large values of d) as what we observed, given then null hypothesis is true?
```{r}
random_subset <- randomized.comms %>%
  filter(ShuffledSiteStatus == "mature") %>%
  select(AVGrandomSpec)

obs.spec <- obs.means %>%
  filter(SiteStatus == "mature") %>%
  select(avg_specialization)

sum(random_subset <= obs.spec$avg_specialization)/
  nrow(random_subset)
```
- 6f. Do we accept or reject our null hypothesis?
Accept!

With these analyses we have approximated the analysis of Ponisio et al. 2016.
Congratulations :)
~~~