Introduction to data summarizing and visualization

Jeff Oliver

26 August, 2019

The R programming language provides many tools for data analysis and visualization, but all the options can be daunting. This lesson provides an introduction to wrangling data in R and using graphics to tell a story with data.

Learning objectives

1. Understand the difference between files and R objects
2. Modify data for proper data hygiene
3. Summarize information from raw data
4. Visualize data to convey information

[DESCRIPTION OR MOTIVATION; 2-4 sentences that would be used for an announcement]

Getting started

The tools: R and RStudio

For this lesson, we will use the R programming language in the RStudio environment. RStudio provides a convenient interface for working with files and packages in R. If you have not done so already, install R and RStudio; details can be found on the installation page.

Preparing our workplace

Key to successful programming is organization. In RStudio, we use Projects to organize our work (a "Project" is really just a fancy name for a folder that contains all our files). For this lesson, we'll create a new project through the File menu (File > New Project). In the first dialog, select "New Directory", and select "New Project" in the second dialog. Next you'll be prompted to provide a directory name. This will be the name of our project, so we should give it an informative name. For this lesson, we will be using data from vegetation surveys of Tumacacori National Historical Park; so give our directory a descriptive name, like "vegetation". We need also to tell RStudio where to put the lesson on our computer; for this lesson, we will place the folder on our Desktop, so it is easy to find. In your own work, you may find it better to place project folders in your Documents folder.

The last thing we need to do to set up our workspace is to use file organization that reinforces best practices. In general, there should be a one-way flow of information: we take information from *data* and write code to produce *output*. We want to avoid any output from messing up our data, so we create separate folders for each. We want to create two folders, one for our data and one for any output, which may include results of statistical analyses or data visualization. In the R console,

```
dir.create("data")
dir.create("output")
```

[Get data]

url:

Data are csv at https://irma.nps.gov/DataStore/DownloadFile/569336 Webpage with files and other information at https://irma.nps.gov/DataStore/Reference/Profile/2233448

save it in vegetation/data

Data in R

Data *outside* R

We are going to start by looking at those data we downloaded in a spreadsheet program like Excel or LibreOffice. Open your spreadsheet program and open the file we just downloaded. We can see the data are organized in columns and rows. This is a good example of what is known as "tidy data" where there is one observation per row and one type of data per column. In these data, each row corresponds to a single plant species in a single plot, and the columns have different data about that species, including the family name and the percent cover in that plot.
Take a moment to look at those data but don't change any of the values in the cells. Close the file, and make sure *not* to save any changes to the file.

Data in R

We are going to work with those same data in R. To do so we will need to read the data in to R's memory, but first we are going to start a script so all the steps we take for data analyses are going to be saved in one place. Make a new script through the File menu, via File > New File… > R Script. We'll start by adding a little bit of information at the beginning of the script:

```
# Plotting Tumacacori vegetation
# Jeffrey C. Oliver
# jcoliver@email.arizona.edu
# 2019-09-06
```

Note that RStudio is also telling us that we have unsaved changes: the text in the tab is red and there is a little asterisk up by the file name (which is probably "Untitled1", which should also be a clue). We should save this file via Ctrl-S (Window or Linux) or Cmd-S (MacOS), giving it a short but descriptive name. We'll use tumacacori-plots.R.

Now we are finally going to read data into R. When you opened the file in Excel you might have noticed that it has the file extension "csv", which stands for Comma-Separated Values. The CSV file format is common for data tables and has the benefit that it can be easily ready by many programs, which is not always the case for .xls and .xlsx files. To read the data into R, we use the `read.csv` function:

```
plant_data <- read.csv(file = "data/tumacacori-vegetation.csv")
```

The statement shows typical syntax for an R command. From an abstract perspective, most R statements look like:

$Variable\ \ Name = Function\ \ Name(Function\ \ Arguments)$

The function (`read.csv`) is given on piece of information, or "argument", in this case the name of the file we wish to read into memory. The data in the file are read into R, then stored in a variable called `plant_data`.

Quality Assurance

Whenever we start writing a new script, we *always* want to make sure the data are being read in correctly. When we are doing this initial quality check, we don't necessarily need to record all the commands we type, so we can use the R console to type commands. We'll start with `head` which shows the first six rows of data:

```
head(plant_data)
```

```
##   Plot_Code            Field_Name           Common_Name        Family
## 1 TUMC_IP01     Acacia constricta     whitethorn acacia      Fabaceae
## 2 TUMC_IP01      Acacia greggii          catclaw acacia      Fabaceae
## 3 TUMC_IP01  Amaranthus palmeri            carelessweed Amaranthaceae
## 4 TUMC_IP01   Aristida ternipes             spidergrass       Poaceae
## 5 TUMC_IP01 Bidens leptocephala   fewflower beggarticks    Asteraceae
## 6 TUMC_IP01   Boerhavia spicata     creeping spiderling Nyctaginaceae
##   Species_Code Percent_Cover     Leaf_Type    Leaf_Phenology Community
## 1       ACACON          30.0 Microphyllous Drought-deciduous Shrubland
## 2       ACAGRE          30.0 Microphyllous Drought-deciduous Shrubland
## 3       AMAPAL           3.0 Microphyllous Drought-deciduous Shrubland
## 4       ARITER           0.5 Microphyllous Drought-deciduous Shrubland
## 5       BIDLEP          18.0 Microphyllous Drought-deciduous Shrubland
## 6       BOESPI           8.0 Microphyllous Drought-deciduous Shrubland
```

Similarily, the `tail` function shows the last six rows of data:

```
tail(plant_data)
```

```
##     Plot_Code               Field_Name            Common_Name
## 286 TUMC_IPS4 Eragrostis lehmanniana        Lehmann lovegrass
## 287 TUMC_IPS4   Erioneuron avenaceum    shortleaf woollygrass
## 288 TUMC_IPS4    Opuntia engelmannii             cactus apple
## 289 TUMC_IPS4      Prosopis velutina          velvet mesquite
## 290 TUMC_IPS4           Salsola kali          Russian thistle
## 291 TUMC_IPS4                   <NA> Unknown Perennial Forb
##             Family Species_Code Percent_Cover     Leaf_Type
## 286        Poaceae       ERALEH           0.5 Microphyllous
## 287        Poaceae       ERIAVE           0.5 Microphyllous
## 288      Cactaceae       OPUENG           0.5 Microphyllous
## 289       Fabaceae       PROVEL          50.5 Microphyllous
## 290 Chenopodiaceae       SALKAL           0.5 Microphyllous
## 291           <NA>       UNKFOR           8.0 Microphyllous
##        Leaf_Phenology Community
## 286 Drought-deciduous Shrubland
## 287 Drought-deciduous Shrubland
## 288 Drought-deciduous Shrubland
## 289 Drought-deciduous Shrubland
## 290 Drought-deciduous Shrubland
## 291 Drought-deciduous Shrubland
```

Note in the output of `tail` there are rows with `<NA>` values. In R, `NA` has a special meaning: it indicates missing values in the data. We'll deal with that in a bit, but remember that missing data may require some special handling in R.

One more useful function is `str`, which stands for "structure":

```
str(plant_data)
```

```
## 'data.frame':    291 obs. of  9 variables:
```

```
##  $ Plot_Code     : Factor w/ 22 levels "TUMC_IP01","TUMC_IP02",..: 1 1 1 1 1 1 1 1 1 ...
##  $ Field_Name    : Factor w/ 71 levels "Acacia constricta",..: 1 2 5 9 12 14 18 37 44 45 ...
##  $ Common_Name   : Factor w/ 74 levels "Annual Forb",..: 73 15 14 57 31 20 55 12 72 16 ...
##  $ Family        : Factor w/ 26 levels "Acanthaceae",..: 12 12 3 20 4 17 20 7 15 12 ...
##  $ Species_Code  : Factor w/ 74 levels "ACACON","ACAGRE",..: 1 2 7 11 14 16 19 39 46 47 ...
##  $ Percent_Cover : num  30 30 3 0.5 18 8 18 0.5 0.5 3 ...
##  $ Leaf_Type     : Factor w/ 3 levels "Broad-leaved",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ Leaf_Phenology: Factor w/ 3 levels "Cold-deciduous",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ Community     : Factor w/ 5 levels "Forest","Shrubland",..: 2 2 2 2 2 2 2 2 2 2 ...
```

The output of `str` shows the size of the data, in terms of number of rows and columns, and the type of data in each column.

Cleaning up

Rarely are data "ready to go" when they are read into R. The data we are using is going to require two adjustments: removal of rows with missing data and a selection of only a portion of the data.

Missing data

To remove rows that have missing data, we use the `na.omit` function. Before we do, though, look at the "Environment" tab of your workspacek. There should be a row for `plant_data` that indicates the size of our data set. In this case, it should show "291 obs. of 9 variables", indicating we have 291 rows of data, with 9 columns. Now let's drop those rows with missing data, putting this command in the script file, not the R console:

```
plant_data <- na.omit(plant_data)
```

Look again at the Environment tab. There should only be 287 obs. because we removed those rows with missing data. The number of columns should not change at all.

Subsetting data

The dataset include observations of 26 different plant families at the plots. For our purposes, we are going to focus on a few of the families that make up most of these communities, namely the legumes (Fabaceae), grasses (Poaceae), mustards (Brassicaceae), and amaranths (Amaranthaceae).

```
# Families to focus on
families_keep <- c("Fabaceae", "Poaceae", "Brassicaceae", "Amaranthaceae")

# Create new data with only four families
subset_data <- plant_data[plant_data$Family %in% families_keep, ]

# Re-level data in the Family column
subset_data$Family <- factor(subset_data$Family)
```

In those three lines, we:

- Created a list of the names of the families we are interested in,
- Made a new variable called `subset_data` and stored the data for only those four families, and
- "Re-leveled" the data in the Family column (no need to worry too much about what this means now, it will just make plotting easier later on).

Look again at your Environment tab, you should see another item listed, this one called `subset_data`. It *should* have 164 rows (observations). If it doesn't, check your work to make sure you spelled all the family names correctly.

What about that spreadsheet file?

So we've done some manipulations to the data, dropping rows with missing data and creating a subset for four families. Did that do anything to the original spreadsheet file? Open the file in your spreadsheet program (like Excel or LibreOffice) and take a look. If you look towards the bottom, you can see that those rows with NA are still there. Compare this with the output of `tail(plant_data)`. Note the `plant_data` in R does not have the rows with NA values because we removed them with `na.omit` above. This demonstrates that modfiying data in R *does not change the data in the original data files*. Files are only changed when we explicity tell R to write changes to the hard drive. Since we do not want those changes written to our original data file, we are *not* going to have R write any data to the files.

Summarizing data

Using `summary`

```
summary(subset_data)
```

```
##      Plot_Code                      Field_Name               Common_Name
##  TUMC_IP12: 13   Prosopis velutina       :21   velvet mesquite     :21
##  TUMC_IP15: 11   Acacia greggii          :19   catclaw acacia      :19
##  TUMC_IP05: 10   Amaranthus palmeri      :17   carelessweed        :17
##  TUMC_IP14: 10   Bouteloua curtipendula  :14   sideoats grama      :14
##  TUMC_IP20: 10   Descurainia pinnata     :14   western tansymustard:14
##  TUMC_IP02:  9   Setaria                 :12   bristlegrass        :12
##  (Other)  :101   (Other)                 :67   (Other)             :67
##            Family    Species_Code Percent_Cover
##  Amaranthaceae:17   PROVEL :21    Min.   : 0.50
##  Brassicaceae :18   ACAGRE :19    1st Qu.: 0.50
##  Fabaceae     :53   AMAPAL :17    Median : 3.00
##  Poaceae      :76   BOUCUR :14    Mean   :12.73
##                     DESPIN :14    3rd Qu.:18.00
##                     SETAR  :12    Max.   :86.00
##                     (Other):67
##                    Leaf_Type             Leaf_Phenology
##  Broad-leaved           : 11   Cold-deciduous   : 11
##  Broad-leaved herbaceous:  4   Drought-deciduous:149
##  Microphyllous          :149   Herb - annual    :  4
##
##
##
##
##              Community
##  Forest            :15
##  Shrubland         :70
##  Wooded Herbaceous: 4
##  Wooded Shrubland :71
```

```
##  Woodland         : 4
##
##
```

Summary statistics for groups

```
mesquite_mean <- mean(subset_data$Percent_Cover[subset_data$Common_Name == "velvet mesquite"])
mesquite_sd <- sd(subset_data$Percent_Cover[subset_data$Common_Name == "velvet mesquite"])
```

We can see the values of each of these by typing in the variable name alone into the R console:

```
mesquite_mean
```

```
## [1] 27.07143
```

```
mesquite_sd
```

```
## [1] 21.14077
```

Get % cover for each family/plot

Ultimately, we would like to look at how much cover there is for each plant family in the Tumacacori plots. Right now the data are shown for each *species*, but we would like to know how much of the plant cover corresponds to each *family* .

In order to summarize information for each plant family, we are going to use a third-party package. What does that mean, "third-party"? When we think of software, there are generally two parties: the one that wrote the software and the one that uses the software. In this case, the R Core Team wrote the R software and we are the second party, the users of R. Third-party packages are those written by someone other than the authors of the original software. R is especially amenable to third-party development and some of the most widely used packages in R were developed by teams other than the R Core Team. Importantly, if we are using third-party R packages, we need to take two steps: first we need to install the package, second we need to load the package's functions into memory. The first (installation) only has to happen once on your machine; the second (loading into memory) has to happen every time you use R.

To install the dplyr package:

```
install.packages("dplyr")
```

Now that the package is installed, we can load it into memory with the `library` command. When using third-party packages, we generally add `library` commands to the start of the script, so anyone reading our script can tell which, if any, additional packages the script requires to run.

```
library("dplyr")
```

```
family_data <- subset_data %>%
  group_by(Plot_Code, Family, Community) %>%
  summarize(Family_Percent_Cover = sum(Percent_Cover))
```

If we consider the above code using English, it is easier to understand if we replace all the pipes (`%>%`) with the word "then":

```
Make a new variable called family_data, take the subset_data, then
  group the data by Plot_Code, Family, and Community, then
  add up all the values in the Percent_Cover column and store it
      in a column called "Family_Percent_Cover"
```

Visualizing data

The ggplot2 package

In order to visualize the data, we are going to use the ggplot2 package. Like we did for the dplyr package, we will first need to install the package:

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

Now that the package in installed, we can load the functions into memory with the `library` function:

```
library("ggplot2")
```

First rule of plots

Draw it by hand first

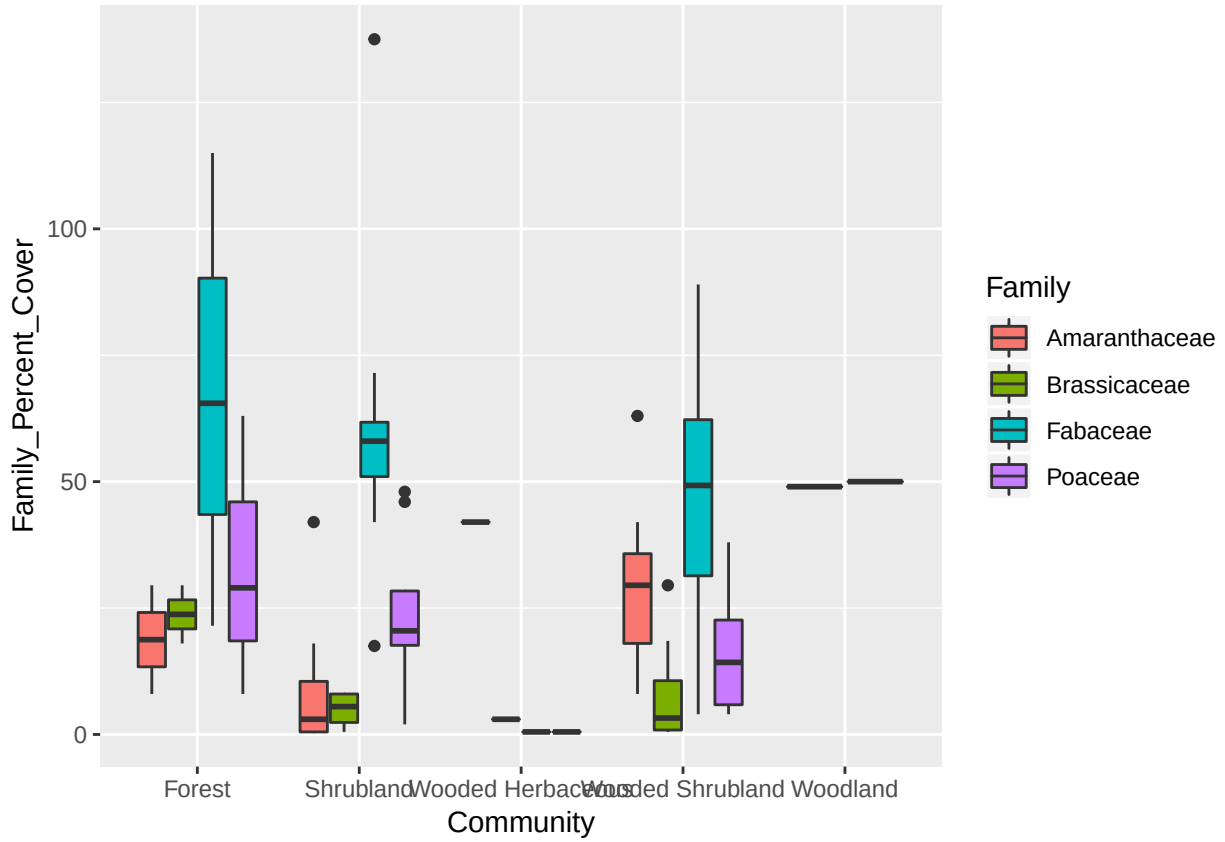
Code it second

Simple boxplot, no colors x is Community y is Percent_Cover group is family

Telling the story

Add group colors for story color is family

```
cover_plot <- ggplot(data = family_data,
                     mapping = aes(x = Community, y = Family_Percent_Cover,
                     fill = Family)) +
  geom_boxplot()
print(cover_plot)
```



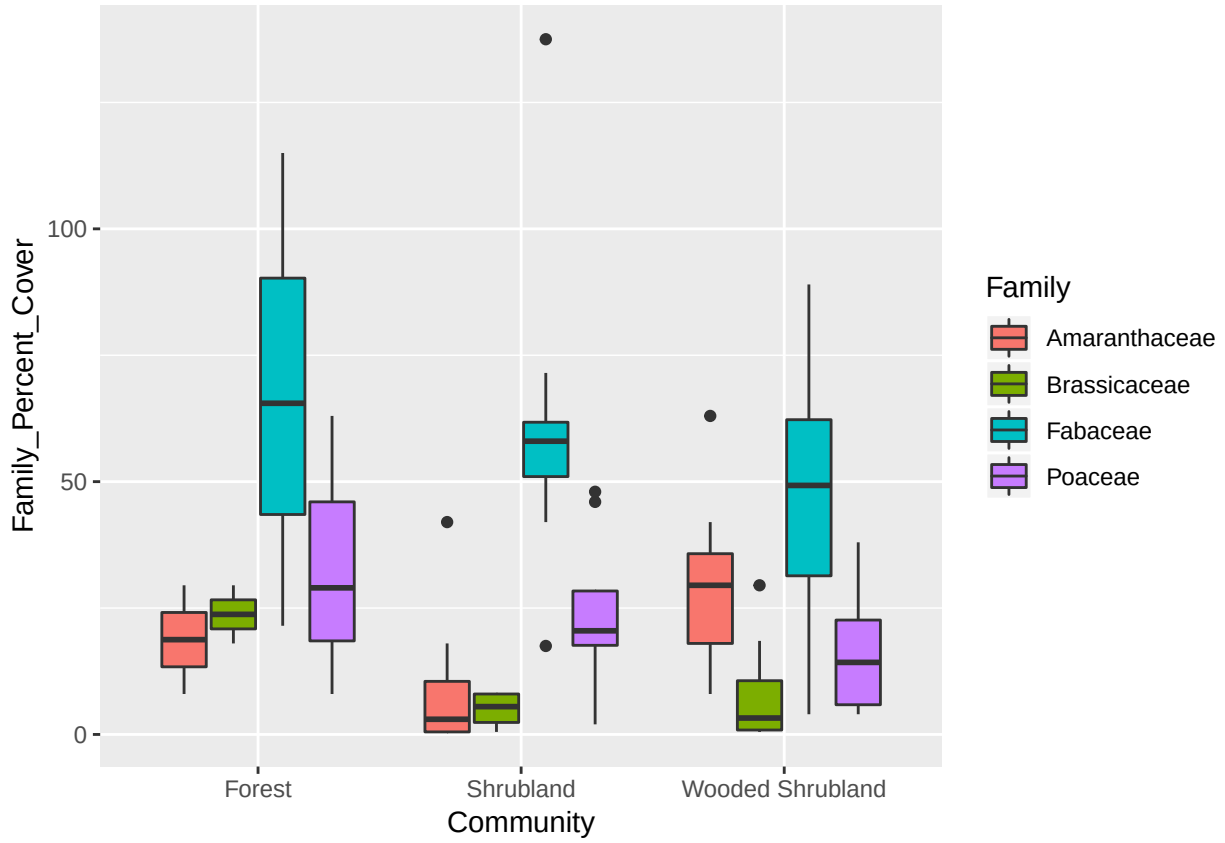
Look at the Wooded Herbaceous and Woodland - they're just lines. Why?

```
table(family_data$Family, family_data$Community)
```

```
##
##                 Forest Shrubland Wooded Herbaceous Wooded Shrubland
##   Amaranthaceae      2         7                 1                7
##   Brassicaceae       2         4                 1                8
##   Fabaceae           3         8                 1                8
##   Poaceae            3         8                 1                8
##
##                 Woodland
##   Amaranthaceae        0
##   Brassicaceae         0
##   Fabaceae             1
##   Poaceae              1
```

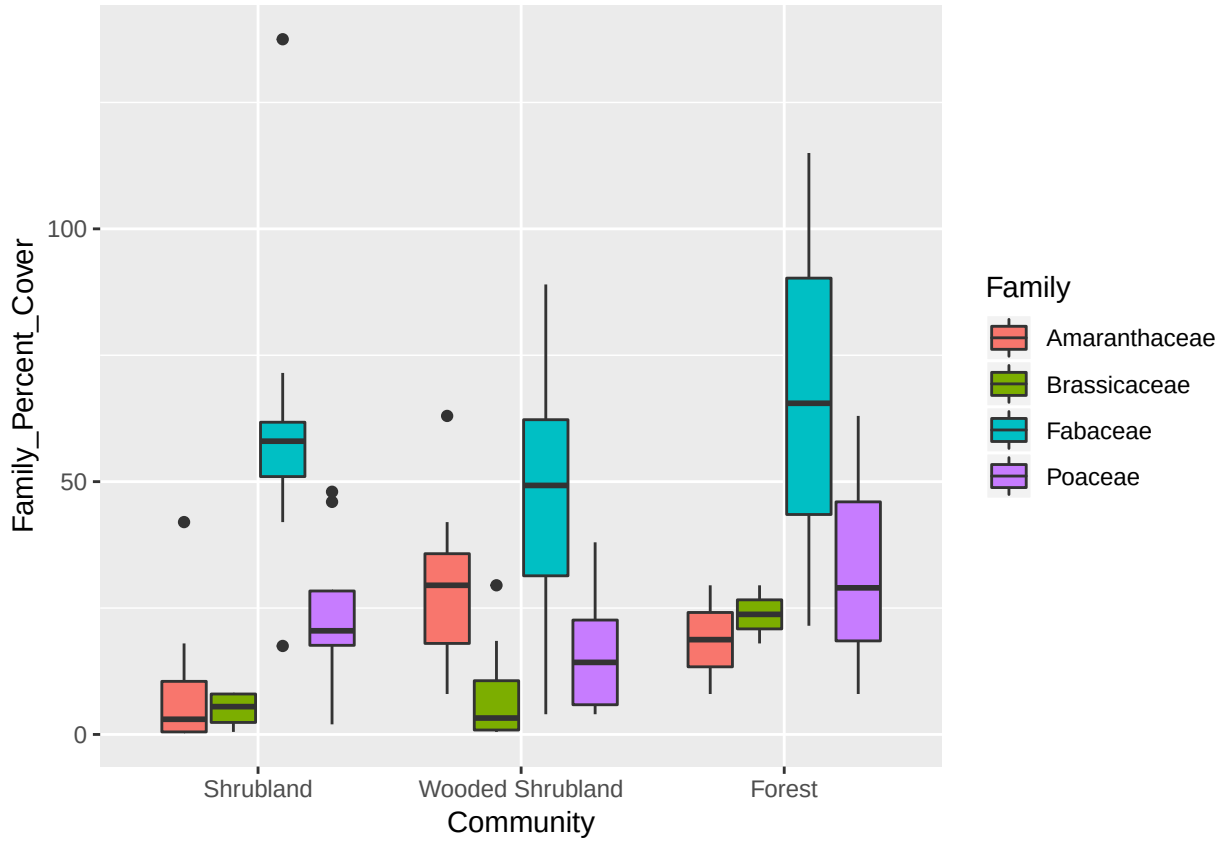
Since we only have those single plots for Wooded Herbaceous and Woodland, we should probably exclude them from our plot.

```
family_data <- family_data[!(family_data$Community %in% c("Wooded Herbaceous", "Woodland")), ]

cover_plot <- ggplot(data = family_data,
                     mapping = aes(x = Community, y = Family_Percent_Cover,
                     fill = Family)) +
  geom_boxplot()
print(cover_plot)
```



Re-level physio class so X-axis is shown in increasing tree cover

```
family_data$Community <- factor(family_data$Community,
                                levels = c("Shrubland", "Wooded Shrubland", "Forest"))

cover_plot <- ggplot(data = family_data,
                     mapping = aes(x = Community, y = Family_Percent_Cover,
                     fill = Family)) +
  geom_boxplot()
print(cover_plot)
```



Additional resources

- Official ggplot documentation
- A handy cheatsheet for ggplot
- A PDF version of this lesson

Back to learn-r main page

Questions? e-mail me at jcoliver@email.arizona.edu.