R Data Viz Handout

By Emma Slayton (eslayton@andrew.cmu.edu)

Welcome to the “Advanced” R workshop offered by the Carnegie Mellon University Libraries. This workshop will focus on introducing data visualization techniques in RStudio, including various introductory exercises, and organizing or cleaning data in RStudio. This handout will focus on these exercises, including walking you step by step through creating various visualizations.

The following exercises assume that you have a basic understanding of R coding, and will focus on introducing students to data visualization in R only.

You will need to have downloaded RStudio to complete these exercises: https://www.rstudio.com/products/rstudio/download/

[Setup instructions](#_krb6l5huwy9h)

[Installing and loading libraries](#_a6hgjcz38gpr)

[Exercise 1: Setting up a project in RStudio](#_gxn5ejw5q2iv)

[Exercise 2: Data Wrangling and Cleaning](#_yqn0i8qu11s2)

[Loading Data](#_90fvytvh54t)

[Defining Clean Data](#_jd5blkz2abkg)

[Finding Null Values](#_ujnbbz6tyrax)

[Checking What is in the Data Set](#_9pxatqwvflyi)

[D.1 Renaming](#_cu702tmou7uo)

[D.2 Filtering and Sorting Data](#_1pwexkr596om)

[D.3 Split, Analyze, and Combine](#_n5v4p4wq9p1r)

[Exercise 3: Adding Other data](#_3sx6rttj24uu)

[Exercise 4: Data Visualisations in R](#_gb8qx6qrxlyb)

[Making a bar chart](#_xnyijrmye2dv)

[Making a Histogram](#_b1lryeiexjcl)

[Installing tidyverse](#_3nb4o9gunx03)

[D. Histograms using ggplot2](#_hvadq5bqt7ax)

[E. Basic Scatter Plot using ggplot2](#_rce31y95eju9)

[F. Creating a more complex scatter plot](#_hs7nkacr2n53)

[F.1 Hexplots](#_tyggmyfdzolj)

[F.2 Building plots in stages](#_ba469yorqzaa)

[F.3 Cleaning out our data within a visual frame](#_90olvphgon0z)

[G: Box Plot using ggplot2](#_xtns5v5ahckp)

[G.1. Basic Box Plot](#_l7zq4ixjco5h)

[G.2 Basic Violin Plot](#_5v6ixj1rqa7l)

[H: Basic line graphs (as time series) in ggplot2](#_xf0fuu5mpy5u)

[I: Series of Small Multiples](#_v8lp6qahqz60)

[J: Changing the background of plots](#_s6kjc9dye4g8)

[J: Saving your plots](#_ufd4ag1bjmu0)

[Additional Information](#_9y547kxr9j48)

[References used to create this workshop:](#_c3g96aws6z1)

### Setup instructions

**R** and **RStudio** are separate downloads and installations. R is the underlying statistical computing environment, but using R alone is no fun. RStudio is a graphical integrated development environment (IDE) that makes using R much easier and more interactive. You need to install R before you install RStudio. After installing both programs, you will need to install the **tidyverse** package from within RStudio. Follow the instructions below for your operating system, and then follow the instructions to install **tidyverse**.

Windows

If you already have R and RStudio installed

* Open RStudio, and click on “Help” > “Check for updates”. If a new version is available, quit RStudio, and download the latest version for RStudio.
* To check which version of R you are using, start RStudio and the first thing that appears in the console indicates the version of R you are running. Alternatively, you can type sessionInfo(), which will also display which version of R you are running. Go on the [CRAN website](https://cran.r-project.org/bin/windows/base/) and check whether a more recent version is available. If so, please download and install it. You can [check here](https://cran.r-project.org/bin/windows/base/rw-FAQ.html#How-do-I-UNinstall-R_003f) for more information on how to remove old versions from your system if you wish to do so.

If you don’t have R and RStudio installed

* Download R from the [CRAN website](https://cran.r-project.org/bin/windows/base/release.htm).
* Run the .exe file that was just downloaded
* Go to the [RStudio download page](https://www.rstudio.com/products/rstudio/download/#download)
* Under *Installers* select **RStudio x.yy.zzz - Windows Vista/7/8/10** (where x, y, and z represent version numbers)
* Double click the file to install it
* Once it’s installed, open RStudio to make sure it works and you don’t get any error messages.

macOS

If you already have R and RStudio installed

* Open RStudio, and click on “Help” > “Check for updates”. If a new version is available, quit RStudio, and download the latest version for RStudio.
* To check the version of R you are using, start RStudio and the first thing that appears on the terminal indicates the version of R you are running. Alternatively, you can type sessionInfo(), which will also display which version of R you are running. Go on the [CRAN website](https://cran.r-project.org/bin/macosx/) and check whether a more recent version is available. If so, please download and install it.

If you don’t have R and RStudio installed

* Download R from the [CRAN website](https://cran.r-project.org/bin/macosx/).
* Select the .pkg file for the latest R version
* Double click on the downloaded file to install R
* It is also a good idea to install [XQuartz](https://www.xquartz.org/) (needed by some packages)
* Go to the [RStudio download page](https://www.rstudio.com/products/rstudio/download/#download)
* Under *Installers* select **RStudio x.yy.zzz - Mac OS X 10.6+ (64-bit)** (where x, y, and z represent version numbers)
* Double click the file to install RStudio
* Once it’s installed, open RStudio to make sure it works and you don’t get any error messages.

Linux

* Follow the instructions for your distribution from [CRAN](https://cloud.r-project.org/bin/linux), they provide information to get the most recent version of R for common distributions. For most distributions, you could use your package manager (e.g., for Debian/Ubuntu run sudo apt-get install r-base, and for Fedora sudo yum install R), but we don’t recommend this approach as the versions provided by this are usually out of date. In any case, make sure you have at least R 3.3.1.
* Go to the [RStudio download page](https://www.rstudio.com/products/rstudio/download/#download)
* Under *Installers* select the version that matches your distribution, and install it with your preferred method (e.g., with Debian/Ubuntu sudo dpkg -i rstudio-x.yy.zzz-amd64.deb at the terminal).
* Once it’s installed, open RStudio to make sure it works and you don’t get any error messages.

For everyone

After installing R and RStudio, you need to install the tidyverse and RSQLite packages.

* After starting RStudio, at the console type: install.packages(c("tidyverse"))
* You can also do this by going to Tools -> Install Packages and typing the names of the packages separated by a comma.

### Installing and loading libraries

Before we can use some of the functions we will need to work through the following exercises, we need to load an R library, tidyverse. For more information about this package, visit <https://www.tidyverse.org>

To instal packages in RStudio, we need to use the function install.packages(). Once you have installed the package, you can access it through any RStudio project.

Instal the tidyverse for use in our project:

1. Type: install.packages(“tidyverse”)

To instal packages in RStudio, we need to use the function library(). Everytime you start a new project, you will need to reload all of the packages you wish to use in that session.

Load the tidyverse for use in our project:

1. Type: library(tidyverse)

It is now possible to move on to working with data from outside RStudio.

## Exercise 1: Setting up a project in RStudio

Let’s start by creating a new R directory and R project. Within RStudio go to ‘File’ , ‘New Project’. This will open up a new window ‘Create Project’. Choose ‘New Directory’.

This will open up a new option ‘Project Type’. Select ‘New Project’. You will now have the option to set up a new project giving it a ‘Directory name’. You are also offered the opportunity to select where this project will be stored. You will now be able to work within this project for the remainder of the workshop.

In order to make a clean working procedure, create new files within this project where we can store our scripts. You can do this by clicking on the ‘New Folder’ button in the sub-window at the bottom right hand corner of your RStudio page. Create new files called ‘data’, ‘outputs’ .

## Exercise 2: Data Wrangling and Cleaning

### Loading Data

The first step with any R project is to procure and store your data. First we need to make sure that our RStudio project has access to the tidyverse. The tidyverse package of functions is discussed more in depth in the CMU Libraries Introduction to R Workshop Handout.

install. packages(‘tidyverse’)

install. packages(‘dplyr’)

library(tidyverse)

library(dplyr)

First, we need to use the tidyverse function download.file() to pull our data from the web.

download.file("<https://ndownloader.figshare.com/files/22031487>",

"data/books.csv", mode = "wb")

Next, we need to make sure we upload the data set we want to work with (“books.csv”) into our RStudio project. To do that, use the following code, making sure that you are including the correct file path name.

books <- read\_csv("./data/books.csv")

### Defining Clean Data

What exactly is clean data? Clean data is accurate, complete, and in a format that is ready to analyze. Characteristics of clean data include data that are:

* Free of duplicate rows/values
* Error-free (e.g. free of misspellings)
* Relevant (e.g. free of special characters)
* The appropriate data type for analysis
* Free of outliers (or only contain outliers have been identified/understood), and
* Follows a “tidy data” structure

Common symptoms of messy data include data that contain:

* Special characters (e.g. commas in numeric values)
* Numeric values stored as text/character data types
* Duplicate rows
* Misspellings
* Inaccuracies
* White space
* Missing data
* Zeros instead of null values

So the next step, is to ensure our data follows those protocols so we can properly work with it in RStudio.

### Finding Null Values

One way to help clean our data is to determine how many null values (or missing data points) are lurking in our data set. To check the number of null values, you can use this code:

sum(is.na(books))

The resulting data shows that we have a lot of missing values, 14509 null values. But that does not provide the detail we might want in order to further explore where our missing data is. To see how many missing values we have by column, we can use the following code.

colSums(is.na(books))

### Checking What is in the Data Set

There are a few ways we can view the data within ‘books’. One option is to use glimpse(), which will show us some of the column values within the data set.

glimpse(books) # print names of the books data frame to the console

#### D.1 Renaming

From there, we can see that there are some results from our column ‘title’ that are not particularly helpful for our inquiry. We can use the rename() function to rename some of those confusing data variables. Here we rename the X245.ab variable. Make sure you assign the output to your books value, otherwise it will just print it to the console. In other words, we are overwriting the previous books value with the new one, with X245.ab renamed to title.

books <- rename(books,

title = X245.ab)

We can also rename multiple variables at once using the following code:

books <- rename(books,

author = X245.c,

callnumber = CALL...BIBLIO.,

isbn = ISN,

pubyear = X008.Date.One,

subCollection = BCODE1,

format = BCODE2,

location = LOCATION,

tot\_chkout = TOT.CHKOUT,

loutdate = LOUTDATE,

subject = SUBJECT)

books

You can use the help() function to learn more about what we can do using rename().

#### D.2 Filtering and Sorting Data

In the last lesson we learned how to subset a data frame using brackets. As with other R functions, the dplyr package makes it much more straightforward, using the filter() function.

Here we will create a subset of books called booksOnly, which includes only those items where the format is books. Notice that we use two equal signs == as the logical operator:

booksOnly <- filter(books, format == "book")

We can also filter for multiple elements at once:

bookCheckouts <- filter(books,

format == "book",

tot\_chkout > 0)

We can also reorder our data within our data set to make it more clear exactly what types of information we are reading. The arrange() function in the dplyr package allows you to sort your data by alphabetical or numerical order.

booksTitleArrange <- arrange(books, title)

To sort in descending order use desc()

booksHighestChkout <- arrange(books, desc(tot\_chkout))

booksHighestChkout

#### D.3 Split, Analyze, and Combine

Many data analysis tasks can be approached using the split-apply-combine paradigm: split the data into groups, apply some analysis to each group, and then combine the results. dplyr makes this very easy through the use of the group\_by() function.

The function group\_by() is often used together with summarize(), which collapses each group into a single-row summary of that group. group\_by() takes as arguments the column names that contain the categorical variables for which you want to calculate the summary statistics.

So to compute the average checkouts by format:

books %>%

group\_by(format) %>%

summarize(mean\_checkouts = mean(tot\_chkout))

Or

books %>%

filter(format == "book") %>%

mutate(call\_class = str\_sub(callnumber, 1, 1)) %>%

group\_by(call\_class) %>%

summarize(count = n(),

sum\_tot\_chkout = sum(tot\_chkout)) %>%

arrange(desc(sum\_tot\_chkout))

## Exercise 3: Adding Other data

For step two we will need to get our test data set into RStudio. This is necessary, as without any data uploaded into the frame, we will not be able to create any visualizations. The visualization portion of our lesson will focus on using two different data sets.

First, we need to download the data set from the internet. The data set we will be using comes from the Data Carpentries workshop ‘Data Analysis and Visualization in R for Ecologists’ by Lesson Maintainers François Michonneau & Auriel Fournier.

To download the files we need from the internet, type the following phrase into your console:

download.file(url="https://ndownloader.figshare.com/files/2292169",

destfile = "portal\_data\_joined.csv")

You have now added the file to your ‘data’ folder. To access the data and to create visualizations we will need to load the data into our session. To do this, type:

surveys <- read.csv("portal\_data\_joined.csv")

Although no data will appear automatically in the console, we can test to see if the data loaded by typing:

View(surveys)

You can now scroll through the table to explore what variables we are dealing with, and if there are any stories we might want to tell with this data set.

## Exercise 4: Data Visualisations in R

### Making a bar chart

Let’s start our data visualization by creating the most prolific and basic of visualization types, the bar chart. To do this we can compare the number of females and males found that were catalogued during the experiment.

To create this, we use the very basic and straightforward phrase plot().

plot(surveys$sex)

Plot() can also be used to make scatter plots, by adapting it to plot.default.

We can now see a bar chart with two columns showing the total number of ‘Female’ and ‘Male’ rodents in our data set. However, there is a third column, showing the number of rodents for which sex is unknown, that is unnamed. We should give the bar a title in our x axis so that our readers will have all the context necessary to fully interpret our graph.

First we must pull out the data on ‘sex’ to create a copied set of data that will allow us to make changes without modifying the working copy of the data frame.

To do this, we can use the code:

sex <- surveys$sex

We have now created a separate category ‘sex’, which we can use as a variable in future code phrases as a shortcut.

To get clearer information on these levels we can use the phrase:

levels(sex)

This should return in quotation marks the titles of all the headers for our data on the rodents ‘sex’, shown as ‘ [1] "" "F" "M" ‘.

We can see that one of our categories shows the data for rodent sex was left blank, so this level can be renamed as unknown.

levels(sex)[1] <- "Unknown"

Using the level(sex) phrase again, we can now see that our levels data expressing rodent sex are now named ‘ "unknown" "F" "M" ‘.

If we run the plot function again, we can see this change is reflected in our bar chart as well.

plot(sex)

We may want to add more context to this visualization. We can do this by adding a title to our graph, as well as labeling the x and y axes.

title(main="Rodent Gender", sub=" ",

xlab="Number of Rodents", ylab="Number of Rodents")

We can even go further by coloring our labels:

title(main="Rodent Gender", col.main="red",

sub="Rodents recorded gender during experiments", col.sub="blue", xlab="Number of Rodents", ylab="Number of Rodents", col.lab="black", cex.lab=0.75)

Now let's practice by creating a second bar chart for the variable ‘species\_id’ within our surveys data set. When you have successfully made a second graph with labels move onto the next exercise.

### Making a Histogram

Building off what we just did to make a bar chart, we can now try to make a histogram.

We can start by testing out the hist() function using some sample data already included in RStudio downloads AirPassengers

hist(AirPassengers)

You should now see a graph of a histogram that plots out the various values for air passenger frequency.

However, if you want to select only a specific column of a data frame, surveys for example, to make a histogram, you will have to use the hist() function with the dataset name in combination with the $ sign, followed by the column name. It is important to note that you can ONLY use NUMERIC data with the hist() function. As such, instead of using sex, we ill use our year collumn data:

hist(surveys$year)

We can also use this method to see the distribution of hindfoot\_lenght.

hist(surveys$hindfoot\_length)

Adapting a basic histogram in R using the hist() function is simple. You only need to follow the below guide.

hist(surveys$hindfoot\_length,

main="Histogram for Hindfoot Length",

xlab="Hindfoot Length",

border="blue",

col="green",

las=1,

breaks=10)

You can see in the above code, that we have added some design elements and titles to the map in situ, much like we did with the more extensive code above. It is always helpful to use the help() function to learn more about individual actions you can do as part of function types in R, such as adding a header or color.

The hist() function shows you by default the frequency of a certain bin on the y-axis. However, if you want to see how likely it is that an interval of values of the x-axis occurs, you will need a probability density rather than frequency. You thus want to ask for a histogram of proportions. You can change this by setting the freq argument to false or set the prob argument to TRUE:

hist(surveys$hindfoot\_length,

main="Histogram for Hindfoot Length",

xlab="Hindfoot Length",

border="blue",

col="green",

las=1,

breaks=10,

prob=TRUE)

The resulting graph should have the density shown on the y axis.

While you can showcase density, overlaying a line to show that changing probability is possible, you need to have data that doesn't have any missing values. We can then try this moving back to AirPassengers. After you’ve called the hist() function to create the above probability density plot, you can subsequently add a density curve to your dataset by using the lines() function:

hist(AirPassengers,

main="Histogram for Air Passengers",

xlab="Passengers",

border="blue",

col="green",

xlim=c(100,700),

las=1,

breaks=5,

prob = TRUE)

lines(density(AirPassengers))

We will come back to histograms after we look through ggplot2!

### Installing tidyverse

For the following exercises, you will need to add a R package, ‘tidyverse’.

To add this package, you may need to install it first. To do so, type the following into your console: install.packages(tidyverse).

To enable us to work with the tidyverse package in this project, use the following:

library(tidyverse)

You can check to see if this package has been added by either looking under the packages tab for the bottom right window of RStudio. A check mark should be next to the tidyverse name to show it is enabled. For our purposes, we are most interested in using ‘ggplot2’, which is included in the tidyverse package.

As described by its authors, ggplot2 is “ A system for 'declaratively' creating graphics, based on "The Grammar of Graphics". You provide the data, tell 'ggplot2' how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details” (H. Wickham, see ?ggplot2).

ggplot2 allows you to specify variables to plot, how these variables should be displayed, as well as other visual properties (such as graph layout, color, title....). ggplot2 reads the data set as if every dimension has a column and every row is an observation (see von Hardenberg et al. 2018).

### D. Histograms using ggplot2

On the other hand, you can also use the ggplot() function to make the same histogram. In this case, you take the dataset chol and pass it to the data argument. Next, pass the hindfoot\_length column from the dataset as values on the x-axis and compute a histogram of this:

ggplot(data=surveys, aes(surveys$hindfoot\_length)) +

geom\_histogram()

As you saw before, ggplot2 is an implementation of the grammar of graphics, which means that there is a basic grammar to producing graphics: you need data and graphical elements to make your plots, just like you need a personal pronoun and a conjugated verb to make sentences. This means that you feed data to a plot as x and y elements and you need to manipulate some details, such as colors, markers, etc. as graphical elements, which are added as layers.

We can also add other layers of interest / information to this histogram as well by modifying our ggplot inputs. This includes adding color as well as adding context through a title and cleaning up the wording of our x axis.

ggplot(data=surveys, aes(x=surveys$hindfoot\_length)) +

geom\_histogram(col="red",

fill="green",

alpha = 0.2)+

labs(title="Histogram for hindfoot\_length", x="hindfoot\_length", y="Count")

You can also change the bins in your histogram using the ggplot code as well. This is done by using breaks.

ggplot(data=surveys, aes(surveys$hindfoot\_length)) +

geom\_histogram(breaks=seq(10, 30, by=2))

You can also fill the bins with colors according to the count numbers that are presented in the y-axis by passing ..count..

ggplot(data=surveys, aes(surveys$hindfoot\_length)) +

geom\_histogram(breaks=seq(10, 30, by=2),

col="red",

aes(fill=..count..))

You can take this one step further and create a gradient color scheme to showcase information within the histogram.

ggplot(data=surveys, aes(surveys$hindfoot\_length)) +

geom\_histogram(breaks=seq(10, 30, by=2),

col="red",

aes(fill=..count..)) +

scale\_fill\_gradient("Count", low="green", high="blue")

You can also add a trend line over this histogram as well, adding a final layer of information to your graph using the geom\_denisty() function.

### E. Basic Scatter Plot using ggplot2

Now that we have made two bar charts and loaded up tidyverse into our project, we can try making other types of graphs. For example, a scatter plot.

Let's start by assigning our surveys data set as our primary point of focus in ggplot.

ggplot(data = surveys\_complete, mapping = aes(x = weight, y = hindfoot\_length))

With this action, we have also designated what information will fill our x and y axis. You can see in our view plots window that we now have an unpopulated graph with labeled x-axis and y-axis.

We must now add graphical representations of the data as points, bars, or lines. These representations can be called ‘geoms’ (see von Hardenberg A et al. 2018). There are many different types of geoms that can be added to our graphs using ggplot2, which allows us to make detailed and complex graphs within RStudio. (Note: It is important to know that the + in ggplot2 enables the modification of existing ggplot objects (see see von Hardenberg A et al. 2018))

To add geoms to our existing graph, we need to rework our previous line of code.

plot\_1 <- ggplot(data = surveys, mapping = aes(x = weight, y = hindfoot\_length))

plot\_1 + geom\_point()

Your graph should now be filled with multiple points of data from the surveys data set.

Before we move further, we should also title our graph. This way, when we export it, we will know what analysis we were performing with our visualization. Let's call this graph Weight vs. Hoof Length.

Type this into the console:

print((plot\_1 +geom\_point()) + ggtitle("Weight vs. Hoof Length"))

If you had a long title, one that might work better split onto two lines, use the \n to separate the lines of text. For example:

print((plot\_1 + geom\_point()) + ggtitle("Weight vs. Hoof Length \nA comparison of rodents"))

We now have a labeled graph comparing all of our rodent populations eight and hind foot length.

### F. Creating a more complex scatter plot

#### F.1 Hexplots

As R is an open source and free language, many people have adapted frameworks for creating interesting data visualization components that can be accessed using RStudio.

This includes changing out the traditional circle point for points of different shapes. One of these shapes available to us is a hexagon.

To change the points in our previous graph to hexagons, we need to install the following package and load it for use:

install.packages("hexbin") library(hexbin)

We can now use the geom\_hex function to transform our points.

print((plot\_1 + geom\_hex()) + ggtitle("Weight vs. Hoof Length with hexagons") )

#### F.2 Building plots in stages

Often ggplot2 can be used to build plots in stages, or iteratively. We can start by building out the basic type of plot we have been working with for exercises 5 and 6.A.

ggplot(data = surveys, mapping = aes(x = weight, y = hindfoot\_length)) + geom\_point()

With this basic graph in place, we can add color to our data points to add another layer of readability to our graph.

ggplot(data = surveys, mapping = aes(x = weight, y = hindfoot\_length)) + geom\_point(alpha = 0.1, color = "blue")

However, we can add even more meaning or information to our graph if we tie the colors of our points to information from our data set. This may help us to see clustering of values based around a third variable.

ggplot2 will automatically assign color variations to the different variable values within the graph. Let’s add color to our graph, and a third layer of information, by using species\_id.

ggplot(data = surveys, mapping = aes(x = weight, y = hindfoot\_length)) + geom\_point(alpha = 1, aes(color = species\_id))

You can now see that we have a graph that is showing clustering of species\_id by the weight and hindfoot length of our tested rodents. The ggplot2 function also automatically created a key for all of our variables to the right of our graph. Based on the steps we just learned to graph a scatter plot on the variables of ‘weight’ vs. species\_id’, with the variable ‘plot\_type’ used to color the graph.

#### F.3 Cleaning out our data within a visual frame

Ideally, when you start visualizing your data you should already have cleaned out your data set of all extemporaneous information or anything that might stand in the way of a clean visualization. We can see from the last visualization we made, that we may need to clean our data in order for us to achieve the desired visualization.

When we look at the above graph it becomes clear that there are some species included in the data set that do not have corresponding weight information. In order to clean up this graph, we need to remove some of the variables in our data set that have no values (or NA) for either ‘weight’, ‘hindfoot\_length’, or ‘species\_id’.

To do this, use the following code:

surveys\_complete <- surveys %>%

filter(!is.na(weight),

!is.na(hindfoot\_length), !is.na(sex))

We can now remake the graph from **D.2.** using the following code:

ggplot(data = surveys\_complete, mapping = aes(x = species\_id, y = weight)) +

geom\_point(aes(color = plot\_type))

We can also remake the graph showing the comparison between weight, hindfoot\_lenght, and species\_id, to see if there are any changes to the colors or clustering.

ggplot(data = surveys\_complete, mapping = aes(x = weight, y = hindfoot\_length)) +

geom\_point(alpha = 1, aes(color = species\_id))

Compare both sets of graphs and decide which is the better visualization.

### 

### G: Box Plot using ggplot2

#### G.1. Basic Box Plot

Box plots (or box and whisker plots) are a method of showing the distribution of a data set based on the minimum, first quartile, median, third quartile, and maximum (a.k.a. the five number summary).Box plots are also easy to make using ggplot2.

For example, we can quickly show the distribution of weight by species with the following line of code:

ggplot(data = surveys\_complete, mapping = aes(x = species\_id, y = weight)) +

geom\_boxplot()

We can change the basic makeup of our boxplot to be represented by points as well as the shape of the box. This may help you to analyze specific data points while still seeing the mean and quartile relationships of the data.

To create that type of box plot use the code:

ggplot(data = surveys\_complete, mapping = aes(x = species\_id, y = weight)) + geom\_boxplot(alpha = 0) + geom\_jitter(alpha = 0.3, color = "pink")

However, we now have the issue that the boxes are hidden by the points.... How might we change that?

The answer is to switch the order in which the commands appear:

ggplot(data = surveys\_complete, mapping = aes(x = species\_id, y = weight)) + geom\_jitter(alpha = 0.3, color = "pink") + geom\_boxplot(alpha = 0)

As each command is enacted in order, whatever you want to come out on top of all the other information should be the last line of the code.

#### G.2 Basic Violin Plot

Another interesting type of plot, that plays off the idea and shape of the box plot, is the violin plot. A violin plot shows the density of the points by altering the shape of the “box”

To create a violin plot:

ggplot(data = surveys\_complete, mapping = aes(x = species\_id, y = weight)) + geom\_violin (alpha = 0)

### 

### H: Basic line graphs (as time series) in ggplot2

Another useful type of visualization to show change over time, or across several discrete variables, is the line graph.

Luckily, our data set does have time information, in the form of day, month, and year columns in our surveys\_complete table.

Before we create the line graph, we need to first count the total number of records in each group.

yearly\_counts <- surveys\_complete %>% count(year, species\_id)

Let’s start by visualizing a line graph where years are represented on the x axis and the counts from our species\_id field on the y axis.

ggplot(data = yearly\_counts, mapping = aes(x = year, y = n)) + geom\_line()

The graph above shows the results of all counts for all species\_ids broken down by year. While this may be interesting to note general trends over time, we can get further into our analysis by creating a line graph that showcases the count for each separate species in every year as a set of multiple line graphs.

To do so, use this script:

ggplot(data = yearly\_counts, mapping = aes(x = year, y = n, group = species\_id)) + geom\_line()

While we can now see that there has been a breakdown in the species counts, both in the increased number of lines as well as the smaller value at the top of our y-axis, this graph does not help our analysis very much as we cannot tell what line is associated with what species.

We can use color to help further contextualize our line graph. Like before, the colors will be assigned an order by the ggplot2 script which will automatically group the data. As such, we can remove the group function we had included in the script to create the previous graph.

ggplot(data = yearly\_counts, mapping = aes(x = year, y = n, color = species\_id)) + geom\_line()

### I: Series of Small Multiples

One of the most effective types of data visualizations for larger data sets is the use of small multiples. Small multiples are a way to break down data into several visualizations, so that otherwise crowded graphs become easy to compare over multiple plots.

In the language of ggplot2 small multiples are called facets.

To create an example of a faceted (small multiple) graph, we can split up the time series graph we made in Exercise F.

Use the following:

ggplot(data = yearly\_counts, mapping = aes(x = year, y = n)) +

geom\_line() + facet\_wrap(~ species\_id)

Again, we can add color to these graphs so that if you were to show the two small multiple graphs we have made so far together you could link the species by color, using this script:

We could also decide that instead of coloring by species, we could use color within these small multiple graphs to show the difference in count per rodent sex as well. This would create within each small graph a line showing counts for ‘female’ and ‘male’ rodents.

To do so, we first need to create a field holding the counts for ‘female’ and ‘male rodents per year:

yearly\_sex\_counts <- surveys\_complete %>% count(year, species\_id, sex)

Then, using the following code, turn that into a series of graphs:

ggplot(data = yearly\_sex\_counts, mapping = aes(x = year, y = n, color = sex)) + geom\_line() + facet\_wrap(~ species\_id)

Or

ggplot(data = yearly\_counts, mapping = aes(x = year, y = n, color = species\_id)) + geom\_line() + facet\_wrap(~ species\_id)

### J: Changing the background of plots

One thing that can help make graphs successful is to try and unclutter graphs of any visual stimuli that does not represent information or help to make that information more readable.

So far, the graphs we have made using the ggplot2 library have all had grey gridded backgrounds underneath them. This does not necessarily make for the cleanest or easiest to read graph, especially for our small multiple comparisons.

To remove the grey background behind the last plot from exercise G, use:

ggplot(data = yearly\_sex\_counts, mapping = aes(x = year, y = n, color = sex)) +

geom\_line() + facet\_wrap(~ species\_id) + theme\_bw() + theme(panel.grid = element\_blank())

### J: Saving your plots

While you can use the button to export your graphs under the plot tab in the bottom right hand window of RStudio, you can also save your graphs using code. This will allow you to control the resolution of your graphs to a greater extent.

The ggsave function allows you to set the width, height, and dpi of your export.

To do so for our last plot use the following code:

ggsave("rodentsYear.pdf", width = 4, height = 4, dpi = 300)

## Additional Information

If you have any additional questions or wish to attend office hours to receive additional support with visualization you made today or for a project, please email: eslayton@andrew.cmu.edu

### References used to create this workshop:

von Hardenberg A, Obeng A, Pawlik A, Pletzer A, Shiklomanov A, Fouilloux A, Wright A, Fournier A, Marwick B, Brown CT, Johnson C, Voter C, Hulshof C, Bahlai C, Shaw C, Li D, Bouquin D, Stubbs D, Quinn D, Vanichkina D, Fishman D, Wilson E, Hart E, Hannon E, Sügis E, Strauss E, Gan E, Becker E, White E, Rodriguez-Sanchez F, Michonneau F, Boehm F, GMoncrieff, Ye H, Dashnow H, Lapp H, JSurman, Ashander J, Byrnes J, Hollister JW, Chen J, Dunic J, Jon, Keane J, Stachelek J, Herr J, Mislan KAS, Woo K, Cranston K, Jordan KL, Ram K, Hertweck K, Todd-Brown K, Lotterhos K, Peck K, Direk K, Hall K, Tylén K, Chatzidimitriou K, Deer L, Gatto L, Wasser L, Tarkowski L, Breckels L, Foos M, Chiapello M, Robinson M, Akenbrand MJ, Kuzak M, Grenié M, Grenié M, Salmon M, Duffy M, Koontz M, Johnston M, Marino N, Carchedi N, Burge O, Lijnzaad P, Lijnzaad P, Peek R, Supp S, Taylor S, Labou S, Pederson S, Webster T, Reiter T, Sandmann T, Teal T, Furnass W, Pearse W, Li Y, Lapp Z, ab604, ashander, cengel, Seok B, sfn\_brt, suparee (2018). “Data Carpentry: R for data analysis and visualization of Ecological Data.” doi: 10.5281/zenodo.569338 , http://datacarpentry.org/R-ecology-lesson/ .

“[R for Social Scientists”: https://datacarpentry.org/r-socialsci/](https://datacarpentry.org/r-socialsci/)

Karlijn Willems (2019) How to Make a Histogram with Basic R <https://www.datacamp.com/community/tutorials/make-histogram-basic-r>

Karlijn Willems (2019) How to Make a Histogram with ggplot2: <https://www.datacamp.com/community/tutorials/make-histogram-ggplot2>