Data and Donuts: Data Visualization in R using ggplot2

Slide 1: Hi, and welcome to Data and Donuts. I'm Tobin Magle, the data management specialist at the Morgan Library at Colorado State University. Today we're going to be covering data visualization in R using ggplot2

Slide 2: In brief, we'll

1. Discuss why you'd want to use ggplot to made graphs
2. Go over the basic elements of ggplot graphs
3. Identify modifications you can apply to these graphs
4. Learn how to create and apply themes to your graphs
5. And export plots

Slide 3: So why use ggplot2?

- For one, using a scripting language to make graphs makes your research reproducible. You can hand your code and your data to anyone and they can reproduce your results if they can install R. But this is true of all graphics packages in R
- Also, it's part of the tidyverse, which is a set of tools specifically designed for working with data. You can even pipe together commands from dplyr to ggplot to create a seamless workflow.
- Another reason is that the graphs look good by default, where that would take a lot of work in the base R graphics package. And if you don't like the default, it's easy to create and apply custom themes.

Slide 4: The data set that we'll be working with contains data about various characteristics of small animals from an ecological study.

- Each row is data about an individual observed animal.
- Each column is one variable the describes each animal, like species, sex and weight.

Slide 5: for these exercises, we're assuming that you have a basic working knowledge of R and R studio. You'll need to

- Install both R and R Studio. See the setup instructions from Data Carpentry Linked on this slide if you need help.
- Download and unzip the quickstart files from the link on the slide. This file provides a premade working directory and file structure for this lesson.
- If you want to know how to set up a directory for yourself or are unfamiliar with R and R studio, see the Basic data analysis in R lesson linked on this slide.

Slide 6: These data will be loaded into a data frame using the read_csv function. If you're not familiar with read_csv, please go back and see the data wrangling with dplyr video for more information.

Demo 1: Setup

- Open R Studio project via rproj file:

    - Show data in the data folder
    - Show script
- Load ggplot2 package
- Load data using read.csv

Slide 7: Once the data are loaded, we'll be using the ggplot2 package to plot these data. All ggplots contain 3 basic elements:

1. Data in the form of a tibble
2. Aesthetics that determine how the plot looks and
3. Geoms, which specify how the data should be plotted. For example, will the data be represented by points, lines, bars, etc.

Slide 8: The ggplot2 package contains a set of functions to implement these concepts:

1. The ggplot() function initializes the ggplot object.
2. The aes() function draws the axes and other visual features. It used as an argument to the ggplot function
3. The set of geom functions like geom_points() draws the data on the plot
4. Finally, the + operator allows you to add components to the plot in a modular fashion.

Slide 9: The simplest ggplot uses all of the described function to make a plot. The example code on this slide

1. Specifies that we're plotting data from the surveys_complete dataframe
2. Weight is plotted on the x axis, hindfoot length will be plotted on the y axis
3. The data will be represented by points.

Let's break down these pieces of code to see what they do.

Slide 10: First, let's look at the ggplot function. If data is the only specified argument, the function will still initialize the plot window, but it won't plot anything because you haven't specified what to plot.

Slide 11: If you add and the aesthetics argument which specifies what should go on the x and y axes, it draws the axes. In this case, we have weight on the x axis and hindfoot length on the y axis.

Slide 12: However, the data are not plotted on the graph until you specify the geom. Make sure to add the + operator at the end of the ggplot statement. In this case, we'll be plotting the data as points using geom_point()

Let's see how this works

Demo 2:

- gray plot area

```
ggplot(data = surveys_complete)
```

- Axes drawn

```
ggplot(data = surveys_complete,
       aes(x = weight,
         y = hindfoot_length))
```

- scatter plot of hindfoot length vs weight

```
ggplot(data = surveys_complete,
       aes(x = weight,
           y = hindfoot_length)) +
 geom_point()
```

Slide 13: Now that we've drawn basic scatterplot, let's customize it.

- One common customization is altering the transparency of the points
- This modification allows you to see where the points are the most dense
- We can do this by using the alpha argument in the geom_point function

Demo 3:

- Copy and paste Demo 1
- Add alpha = 0.1 as an argument to geom_point

Slide 14: Now let's add some color. We can turn the points blue by adding the color = "blue" argument to geom_point. For a full list of colors, see the color reference chart linked on this slide.

Demo 4:

- Copy and Paste Demo 2
- Add color = blue as an argument to geom_point

Slide 15: We can also use color to tell us something about the data. For example, we can colorize the points based on a factor variable, in this case species id. To do this, we can alter the aesthetics within geom_point with the argument color = species_id.

Demo 5:

- Copy and Paste Demo 3
- Change color = blue to aes(color = species_id)
- Need to use aes because it's referencing the data frame instead of one color

Slide 16: At this point, you might be wondering why we are using the aes function in two different parts of the ggplot code.

- The aes statement in the ggplot function affects the aesthetics of the whole plot
- The aes statement in the geom_point function affects only the points.

- In this case, you could move color = species id into the first aes statement and it wouldn't make a difference, but if you are layering multiple geoms on the plot, it becomes important.

Slide 17:
Exercise 1:
- Use the previous example as a starting point.
- Add color to the data points according to the plot from which the sample was taken (plot_id).

Hint: Check the class for plot_id. Consider changing the class of plot_id from integer to factor. Why does this change how R makes the graph?

Solution:
- Copy and paste previous example
- Change species_id to as.factor(plot_id)

Slide 18: Now let's look at some other geoms we can use starting with box plots.
- Box plots work best with a factor variable on the x axis and a numeric variable on the y axis.
- The syntax for using a boxplot geom is geom_boxplot()
- Let's see how this works

Demo 6:
- Ggplot = surveys complete
- Aes = x = species id, y = hindfoot length
- Geom = geom_boxplot()
- Creates box plots with median, quartiles and outlier points

Slide 19: We said before that you can overlay multiple geoms on the same plot. Let's add some jittered points to the box plot we just made.

Demo 7:
- Copy/paste the previous box plot graph
- Add alpha = 0 as an argument of geom_boxplot so we can see the points
- Add geom_jitter with the arguments alpha = 0.3 and color = "tomato"
    - Jitter plots the points so that they are spread out horizontally on the x axis
- See that the points are densest in the box plots, as you'd expect.
- You can also get a sense of the number of observations for each species

Slide 20:
Exercise 2:

Plot the same data as in the previous example, but as a Violin plot
- Hint: see geom_violin().
- What information does this give you about the data that a box plot does

Solution:

- Copy and paste boxplot w/o the jitter geom
- Replace geom_boxplot with geom_voilin
- Shows you how many observations are at each y value by the width of the violin

Slide 21: Now we're going to plot time series data using geom_line().

- But first, we need to reshape the surveys_complete data frame
- The code on this slide illustrates how to do this using the dplyr package. For more information, see the coding and cookies session on data wrangling
- The output of this code generates a new data frame with year and species id as columns, plus a new column called n that represents the number of observations of a given species in a given year.

Slide 22: To plot these data as a line graph,

- We'll specify that data = yearly counts
- Specify we want year on the x axis and n on the y axis
- And that we should draw a line between the points

Demo 8:

```
ggplot(data = yearly_counts,
       aes(x = year,
           y = n)) +
geom_line()
```

doesn't tell you much, because it's not separated by species

Slide 23: To make a line for each species, we can add a new argument, group, to the aes function.

Demo 9:

- Copy/paste demo 6
- Add group = species_id to aes
- Now there's a line for each species_id
- But we can't tell which line represents which species

Slide 24: To fix this, we can color the lines based on species_id

Demo 10:

- Copy/paste demo 7
- Add color = species id
- See there's a legend showing which line is what species

Slide 25:
Exercise 3:

- Use what you just learned to create a plot that depicts how the average weight of each species changes through the years.
- Hint: reshape the data using the following code

```
yearly_weight <- surveys_complete %>%
    group_by(year, species_id) %>%
    summarize(avg_weight = mean(weight))
```

Solution:

- Copy paste code for counts
- Replace df with yearly_weight
- Replace y = n with y = avg_weight

Slide 26: Now that we've got an informative graph of species observations over the years, let's get it publication ready.

- First, I'd remove the gray background and
- Make the axis labels more descriptive
- As well as increasing the font size
- We can do this using premade themes and the theme function

Slide 27: First, let's apply the premade black and white theme. If you want to see what other themes are available, see ?theme_bw.

Demo 11:

- Copy paste the time series graph code
- Add theme_bw()
- ?theme_bw

Slide 28: This theme is nicer, but we can tweak it to make it exactly what we want. For example, you might want to change the wording on the axis labels. You can do this using the labs() function.

Demo 12:

- Copy paste Demo 9
- Add labs function with title, x and y as arguments
    - Title is the plot title (Observed species in time)
    - X is x axis label (year of observation)
    - Y is y axis label (count)

Slide 29: The new labels are an improvement, but they are a little hard to read. We can change the font size and type with the theme() and element_text() functions

Demo 13:

- Copy paste demo 10
- Add theme
    - Text = element text (size = 16, family = Arial)

Slide 30: Once you get the graph the way you want it, you can save all the theme elements into a custom theme which is stored in a variable. In this case, we're starting with the black and white theme, but changing the font to 16 point Arial.

Slide 31: You can apply this theme to any plot of your choice. For example, a box plot

Demo 14:

- Copy and paste box plot example
- Add arial theme

Slide 32: Finally, you can export your plots to a file using the ggsave function.

- First, save the plot to a variable
- Then run the ggsave function
    - First artgument is the name of the file
    - Second is the variable that holds the plot
    - Then the width and height in inches

Demo 15:

- Save the box plot to a variable
- Ggsave("boxplot.png", box_plot, width = 15, height = 10)
- Open new image file to see.

Slide 33: Thanks for listening. As always, if you have any questions about this material or any other data management topic, please don't hesitate to contact me at the email address on this slide. Other data management topics we cover can be found on our Data management services web site. If you would like to view the full material that this lesson was based on, see the data carpentry R ecology ggplot lesson. Finally, the ggplot2 cheat sheet is a great resource when you're coding on your own.