Data and Donuts: Basic Analysis Using R

Slide 1: Hi, and welcome to Data and Donuts. I'm Tobin Magle, the data management specialist at the Morgan Library. Today we're going to be covering the basics of how to use R for data analysis.

Slide 2: To put this in the context of the research cycle, R is useful after data collection, when you are cleaning, processing, and analyzing your data.

Slide 3: In brief, we'll

1. introduce the R programming language and the R studio interface
2. Discuss how operators and functions work
3. Investigate the structure of data frames
4. And explore the plusses and pitfalls of using factors for categorical variables

Slide 4: for these exercises, you'll need to install both R and R Studio.

- R is the programming language that we'll be using in these exercises. It comes with software that will interpret and run the commands we write.
- RStudio is a "fancier" interface that we will use to write R scripts and interact with the R software.
- If you need help installing R and R studio, please see the installation instructions from Data Carpentry linked to on this slide. (http://www.datacarpentry.org/R-ecology-lesson/#setup_instructions)

Slide 5: Now that we know what we're talking about, let's discuss why we might want to learn to code in R:

- First, coding in any language will make the analysis that you do more reproducible. Without coding, researchers describe how they did their analysis in words, which are by nature imprecise. With R, you can share executable scripts to give colleagues and exact accounting of what you did.
- R specifically is widely used for scientific computing. In fact, the base R software has over 10,000 extensions called packages, many of which were created by researchers.
- These packages allow R to import a variety of data types, from tabular and geospatial data, to text and genomic sequences.
- Not only can you analyze data, but R also produces high quality, publication ready graphs. The code can be reused to make other graphs for similar data.
- Finally, R is free, open source and cross platform, allowing you to take it to any organization you may work at in the future.

Slide 6: Now that you're convinced that R is a good choice for your research, let's set up our workspace for this demo.

Demo 1: Setting Up a working Directory

- Start RStudio
- **File** > **New project** > **New directory** > **Empty project**

- Enter a name for this new folder and choose a convenient location for it (working directory)
- Click on **Create project**
- Create a data folder in your working directory
- Create a new R script (**File** > **New File** > **R script**) and save it in your working directory

Slide 7: Now your workspace should look roughly like this

Demo 2: Orientation to R-Studio interface

1. In the upper left, we have the script window. This is where you edit the text document that holds your code. The text turns red when the file is not saved.
2. In the lower left, in the lower left, we have the console. This is where the code gets executed and output, such as warnings and error messages are displayed.
3. In the upper right, we have the environment tab, where you can see data stored in the computer's memory. You can also look at what has been run in the console using the history tab.
4. In the lower right, you can see the directory you're working in, plots, what packages are installed/loaded, and help files.

Slide 8: It's important to emphasize the difference between the script window and the console

- Both of these windows will accept R commands
- The Console runs the commands, but doesn't save them permanently, though they appear in the history tab on the upper right
- The script is where you save commands that you want to save for later. Once you type them in the script, you still need to send them to the console to be run.
- You can do this by copying and pasting, or putting your cursor on the line that you want to run, and pressing control enter on PC and command enter on the mac.

Slide 9: Now that we're oriented to R studio, let's get into coding! First, we'll talk about operators. An operator is a symbol that tells R to perform a mathematical or logical operation. There are 5 types of operators, but we'll only be working with Arithmetic and Assignment operators in this lesson. For more information on the other types of operators, please see the link at the bottom of this slide (https://www.tutorialspoint.com/r/r_operators.htm).

Slide 10: Let's consider the assignment operator (<-) first. This operator, which is symbolized by the left carat (<) and a dash (-) with no spaces in between assigns a value to a variable. For example, we can assign the value 55 to the variable weight_kg). The assignment operator is so integral to R programming that they have created a short key: alt-dash (Alt-). Now let's look at an arithmetic operator, which allows R to do math like add, subtract, multiply, and divide on numbers or variables. These operators can be combined with the assignment operator. Let's look at operators in action.

Demo 3: Operators

Slide 11:
Exercise 1: Operators

What are the values after each statement in the following?

```
mass <- 47.5            # mass?
age  <- 122             # age?
mass <- mass * 2.0      # mass?
age  <- age - 20        # age?
mass_index <- mass/age  # mass_index?
```

Slide 12: For more complicated tasks, R also comes pre-installed with a variety of functions. Functions are a sequence of program functions that perform a task. You can add functions to R by installing packages or writing your own, but we'll be sticking to base R for this session. Each function accepts input in the form of arguments, and returns a value that can be assigned to a variable as output. Some examples of R functions are square root and round. Let's see how they work.

Demo 4**:** Functions
- Sqrt(4)
- Round(weight_lb)
- Args(round)

Slide 13: Typing numbers into script by hand is ok for practice, but what if you want to load a data table into R? R has useful built in functions to accommodate this:
- Download.file (downloads from a URL)
- Read.csv (loads a comma separated value file into a variable)

Slide 14: Before using these function, it's good to make sure your data format is machine readable. For all the details, see the end of the Data Organization session from Data and donuts. Briefly, spreadsheets in R are stored as a data type called a "data frame".
- In a data frame, each column is a variable, and each row is an observation of each variable.
- For example, if you're a field ecologist trapping animals and recording their species, weight and sex, you would have columns for species, weight and sex, and each row of the table would be an animal that you saw.
- Because a column contains values of one variable, all values must be the same data type (like numbers or text)
- Finally, the data must be "rectangular" or each column must have the same # rows

Demo 5**:**
- download.file(<URL>, <file name>)

- surveys<-read.csv(<file name>)

Slide 15: If you want some details about the data frame, you can use these functions

Demo 6: inspect the surveys data frame
- head(surveys) = look at first 6 rows (all columns)
- str(surveys) = structure # rows, cols, data types
- nrow(surveys) = number of columns
- ncol(surveys) = number of columns
- names(surveys) = column names
- summary(surveys) = does summary stats for each column

Slide 16:
Exercise 2: inspecting data frames

Based on the output of `str(surveys)`, can you answer the following questions?

- What is the class of the object `surveys`?
- How many rows and how many columns are in this object?
- How many species have been recorded during these surveys?

Slide 17: Often, it's useful to look at only part of a dataset when doing analysis. To do this, we can use brackets to subset the data.
- Use square brackets after the name of the data frame
- Specify the row, then the column separated by a comma
- If you want all the columns for a particular row, specify the row and leave the column blank and vice versa for all rows of one column.
- Specify a range of rows or columns using the : operator

Demo 7: Subset using brackets

Slide 18: In addition to bracket notation, you can subset by column name:
- Quote the column name in single square brackets: data frame with just that column
- Quote the col name after a comma in square brackets: vector with that col
- Quote the col name in double square brackets: same as above
- <data frame name>$<column name> = same as above

Demo 8: Subset using column names

Slide 19:
Exercise 3:
1. Create a data.frame (surveys_200) containing only the observations from row 200 of the surveys dataset.
2. Use nrow() to subset the last row in surveys_200.

3. Use nrow() to extract the row that is in the middle surveys_200. Store in a variable called surveys_mid
4. Use nrow() to get the same results as head(surveys) keeping just the first through 6th rows of the surveys dataset.

Slide 20: Sometimes you don't want a specific row, you want all rows that meet specific criteria like if you wanted the average weight for a specific species in the ecology example before. For this, we can use factors.

- Factors represent categorical data.
- This is very useful for comparing groups statistically and plotting
- They are stored as integers, but are assigned text labels. These unique labels are called levels
- They can be ordered (small, med, large) or unordered (M/F)

Slide 21: R has functions to create, modify and inspect factor variables.

- Create: factor
- See levels: levels
- See # levels: nlevels
- Specify order of levels: levels argument

Demo 9: factors

Slide 22: Because of their unique properties, it's easy to convert them to text or numbers:

- Instead of storing it as numbers with text labels, you can store each value as a text string, or characters using as.character()
- Numbers are a bit more complicated. Let's consider a factor variables with years
    - If you try as.numeric: you get the underlying numbers
    - Instead, you have to convert to character, then number
    - For some reason, this is the recommended way

Demo 10: Subset using brackets

Slide 23: Let's look at how factors can be used with the plot function by plotting

Demo 11: Plotting

- **plot(survey$sex)** – R knows it's a factor variable: makes a bar graph of how many are in each category.
- But what's with the missing label?
    - Look at the levels: **levels(survey$sex)**
    - Reveals a level with a blank label
    - Can change labels to solve
- renaming
    - **sex <- surveys$sex** # subset the column
    - **head**(sex) # look at first 6 records
    - **levels**(sex) # look at the factor levels

    - **levels**(sex)[1] <- "missing" # change the first label to “missing”
    - **levels**(sex) # look at factor levels again
    - **head**(sex) # see where missing values were
- re-plot

Slide 24: Here’s the code we used to rename the levels for reference

Slide 25:
Exercise 4:

1. Rename “F” and “M” to “female” and “male” respectively.
2. Now that we have renamed the factor level to “missing”, can you recreate the barplot such that “missing” is last (after “male”)?

Slide 26: Sometimes, using factors can cause more trouble than its worth. For example, I do a lot of data cleaning, and factors can cause errors when trying to combine tables with the same variables, but unique factor levels. In this case, we can use the argument **stringsAsFactors = FALSE** to tell R you want to use them as text only.

Demo 12: Not using levels: stringsAsFactors = FALSE

- In read.csv, it’s true by default. Look at the structure of the DF
- Add the argument stringsAsFactor = FALSE
- Recheck the str

Slide 27: Thanks for listening. If you need help, don’t hesitate to contact me at the address on the slide. Also see our data management services webpage for more information about what services we provide. If you want to see the source material for this lesson, see the data carpentry ecology lessons. Finally, the Base R cheat sheet will come in handy when you are coding on your own.