Data and Donuts: Data Wrangling Using dplyr
Based on the data carpentry ecology lessons:
http://www.datacarpentry.org/R-ecology-lesson/03-dplyr.html

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 data wrangling using the dplyr package in R based on the Data Carpentry curriculum.

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. Demonstrate the 6 verbs for data manipulation
2. Combine these verbs with an operator called the pipes
3. Create a clean dataset to export to a file

Slide 4: 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 bitly 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 5: In this lesson, we are going to move beyond the R base installation and install a package called dplyr.

- First you need to install the package using the install.packages function. You only need to do this once on each R installation
- Then you need to load the package using the library function. You need to do this every time you start RStudio to use the functions it contains.

Let's load the data and the packages before we learn more about dplyr.

Demo 1: Setting up

- Open the R project
- Point out the file structure:
    - Rproj file – save your place while you're working
    - complete R script – follow along if you don't like typing. But I recommend opening a new script and typing
    - Data folder with data file
- Open a new script file
- Read the data into a new variable called surveys
- Install package if you haven't already
- Load the package

Slide 6: Now that we have R studio up and running, let's talk about what dplyr does

- Dplyr provides tools for data manipulation in data frames
- It makes working with data easier than in base R by providing data manipulation "verbs"
- these verbs can be strung together using the pipe operator

To start, let's look at verbs that allow you to specify rows and columns: select and filter

Slide 7: Let's look at our first verb: select

- Select picks columns from a data frame
- and takes a data frame and column names as arguments
- Let's see an example of how select works

Demo 2: select

- Let's say we have a collaborator who only wants the plot, species id and weight data
- Function: Sslect
- arguments: the surveys data frame, the names of the columns you want to keep
- outputs a data frame that only has these columns

Slide 8: We've seen how to choose particular columns using select, now let's look at how to pick rows using filter

- Filter chooses rows based on specified criteria
- It takes the data frame as the first argument and relational expression as the second argument
- For example, you can specify that you only want rows where the year is equal to 1995
- Let's see select in action

Demo 3: filter

- Let's say we only want to look at records taken in 1995
- Function: filter
- Arguments: the surveys data frame, year = 1995
- Outputs a data frame that only has records from 1995

Slide 9: At this point, you might be thinking that you can do these things in R without using dplyr

- However, dplyr provides a convenient way to string verbs together
- using the pipe operator (%>%), (percent sign)-(greater than)-(percent sign)
- The operator goes at the end of each line that you want to string together.
- Then the output of the previous line becomes the data frame input for the next line
- This means that you don't have to explicitly provide the data frame argument in each verb function
- For example we could specify that we only want records that have weights of less than 5 g and only the species_id, sex and weight columns in one statement.
- Let's see how pipes work in a demo

Demo 4: pipes

- We can use the assignment operator to save the output in a data frame called surveys_sml
- First line is the data frame you want to start with (surveys) followed by the pipe
- Next line is the filter statement, which we know selects rows
    - In this case, we only want records where the weight is less than 5
    - Don't forget the pipe at the end of the line
- Then, we want a select statement to pick the columns
    - All we need here is the column names as arguments
    - Species_id, sex, weight
- And the output is a data frame that has 3 specified columns of records that have weight less than 5

Slide 10: Now that you know about select, filters an dpipes, let's do an exercise involving these commands

Exercise 1:

Using pipes, subset the survey data to include individuals collected before 1995 and retain only the columns year, sex, and weight.

Solution 1:

- Start with the surveys data frame, followed by the pipe operator
- Filter on year ==1995
- Select year, sex, and weight
- Could we reverse select and filter?
    - Yes, but only in cases where the select statement contains the variable being evaluated in the filter statement

Slide 11: dplyr also allows you to create new columns using the mutate function

- Like all the other dplyr functions, mutate takes the data frame as the first argument
- The second argument is the name of the new column, then an equation to calculate the values in this new column
- For example, the weight is currently in grams. We could create a new column called weight_kg that stores the weight in kilograms
- Let's see how this works in practice

Demo 5:

- Let's create a new column called weight_kg using mutate
- Function: mutate
- Arguments: surveys and weight_kg = weight/1000

- Just a note with pipes: this is the same as surveys %>% mutate(weight_kg = weight/1000)

Slide 12: Let's do another exercise combining the 3 verbs with pipes
Exercise 2:
Create a new data frame from the survey data that meets the following criteria:
1. contains only the species_id column and a new column called hindfoot_half
2. hindfood_half contains values that are half the hindfoot_length values.
3. Only include records from 1990 and after

Hint: think about how the commands should be ordered to produce this data frame!

Solution 2:
- Start with the surveys data frame
- First, let's filter the rows: filter(year>=1990)
- Then create hindfoot half: mutate(hindfoot_half = hindfoot_length/2)
- Finally, select the columns:
- Why did I do this in this order?
    - Have to do select last because the other 2 depend on year and hindfoot length, which aren't included in the output of the select statement
    - Mutate and filter could be switched, but reducing the number of rows first makes the computation a bit more efficient.

Slide 13: Next, we're going to look at the group by function.
- Group_by doesn't do a lot by itself, but it's very useful in the context of other dplyr verbs.
- Group_by groups data in a data frame by a factor variable.
- It takes a data frame and a factor variable as input,
- and outputs a new type of data structure called tbl_df.

Let's look at how this works

Demo 6: group_by
- Let's see how group_by works
- First, let's look at the structure of the input: surveys
    - Data frame with 13 variables and 30k obs
- Function: group by
- Arguments: surveys, sex
- Save it in a variable called grouped_surveys
- Now look at the structure of grouped survey:
    - Tbl_df, same obs/vars
    - Has attributes at the end that specify how the table is split up by sex

Slide 14: Now let's look at a verb that works well with group by: summarize.
- Summarize applies a summary statistic function to a variable
- It takes a grouped dataframe and a name and the definition of a summary statistic as arguments

- For example: we could use summarize to calculate the mean of weight.

Let's see how this works in practice.

Demo 7: summarize

- First, we have to convery surveys into a grouped table
  - Data<- tbl_df(surveys)
- Then we can send that tbl df into the summarize function, and tell it what statistic to calculate, in this case weight
  - Summary(data, mean_weight = mean(weight)
- And it outputs the mean weight for the whole dataset

Slide 15: Now let's see how group_by and summarize work together in a strategy called split-apply-combine

- Split-apply-combine calculates summary statistics based on a factor
- It takes a data frame, a factor variable and the definition of a summary statistic
- And the output is a table with a summary statistic for each level in the factor variable provided
- For example, we could generate a table of mean weight by sex
- Let's see how this works in practice

Demo 8:

- 1 factor, 1 summary
  - Start with surveys
  - group_by(sex)
  - Summarize with mean_weight = mean(weight, na.rm = TRUE) as argument
    - Get a real answer
- Group by multiple factors
  - Start with survey
  - Group by sex and species id
  - Summarize as above
  - Lots of NaN (not a number) – bc no value fits that sex/species combo
- Group by multiple factors (NA removed)
  - Start with survey
  - Filter(!is.na(weight))
  - Group by sex and species id
  - Summarize as above
  - Only includes combos that exist
  - Add print(n = 15) to control # records that get output
- Multiple factors, multiple summaries
  - Start with survey
  - Filter(!is.na(weight))
  - Group by sex and species id
  - Summarize
    - Mean_weight = mean(weight)
    - Min_weight = min(weight)

Slide 16: Let's look at another function that works well with group by: tally

- Tally counts the number of observations in a group
- It takes a data frame or grouped dataframe as its input

Demo 9: Tally

- Let's look at how tally works
- Start with surveys
- Group by sex
- Tally
- Output: a table with blank, F and M as rows and sex and count as columns

Slide 17: Let's pull this all together in an exercise

Exercise 3:

- How many individuals were caught in each plot_type surveyed?
    - Start with surveys
    - Group by plot type
    - tally

- Use group_by() and summarize() to find the mean, min, and max hindfoot length for each species (using species_id).
    - Start with surveys
    - Filter(!is.na(weight))
    - Group by species id
    - Summarize
        - Mean_hf = mean(hindfood_length)
        - Min_hf = mean(hindfood_length)
        - Min_hf = mean(hindfood_length)

- What was the heaviest animal measured in each year? Return the columns year, genus, species_id, and weight.
    - Start with surveys
    - Filter(!is.na(weight))
    - Group by year
    - Filter(weight == max(weight)) %>%
    - Select year, genus, species_id, weight
    - Arrange(year)

Slide 18: Since the next data and donuts session is about graphing with ggplot, we want to get ready by cleaning up this dataset. First we're going to remove all of the rows with missing values.

Demo 10: remove missing values

- Make a new df surveys_complete
- Start with surveys

- Filter statements
    - Remove missing species ids
    - Remoce missing weight values
    - Remove missing hindfoot length values
    - Remove missing sex values

Slide 19: We can also eliminate rare species from the dataset.

Demo 11: eliminate rare species

- **Create a list of common species (n>50 obs)**
    - Create new data frame called species_count
    - Start with surveys complete
    - Group by species id
    - Tally
    - Filter n>=50
- **Filter out the common species**
    - Start with Surveys_complete
    - Filter on species that are in the common species table you just created

Slide 20: Now that we have a clean dataset, we're going to write surveys complete to a file using the write.csv function

- write.csv takes a data frame, the name of an output file at minimum as arguments
- you can also specify other parameters, like whether or not to include row names in the file
- The content of a data frame is then output to the specified file

Demo 12: write data

- Function: write.csv
- Arguments:
    - Data frame: surveys complete
    - Output file: surveys_complete.csv
    - Row.names = FALSE – prevents row names from being written in the file
- Verify new file is there

Slide 21: Thanks for listening. As always, email me at the address on the slide if you need help with these or any other data management topics. See out web site for a list of the topics I can help with. Additionally, see the data carpentry lessons for the full source material for this lesson. Finally, the data wrangling cheat sheet is a good resource for dplyr as you're coding.