Lab 1D - Zooming Through Data

Directions: Follow along with the slides and answer the questions in **bold** font in your journal.

## Data with Clarity

* Previously, we've looked at graphs of entire variables (By looking at all of their values).
  + Doing this is helpful to get a *big picture* idea of our data.
* In this lab, we'll learn how to *zoom in* on our data by learning how to subset.
  + We'll also learn a few ways to manipulate the plots we've been making to make them easier to use for analyses.
* Import the data from your class' *Food Habits* campaign and name it food.

## Splitting data sets

* In lab 1B, we learned that we can *facet* (or split) our data based on a categorical variable.
* **Use the dotPlot() function to create a dotPlot of the amount of sugar in our food data.**
  + The code to create a dotPlot is exactly like you'd use to make a histogram.
  + Make sure to use a capital *P* in dotPlot.
* **Split the dotPlot in two by faceting on our observations' salty/sweet variable.**
  + **Describe how R decides which observations go into the left or right plot.**
  + **What does each *dot* in the plot represent?**

## Altering the layout

* It would be much easier to compare the sugar levels of salty and sweet snacks if the dotPlots were stacked on top of one another.
* We can change the **layout** of our separated plots by including the layout option in our dotPlot function.
  + Add the following option to the code you used create the dotPlot split by salty\_sweet

layout = c(1,2)

* *Hint*: Use your history pane to see how we handled options with the bargraph function. Use a similar syntax to add the layout option to the dotPlot function.

## Subsetting

* Subsetting is a term we use to describe the process of looking at only the data that conforms to some set of rules:
  + Geologists may subset earthquake data by looking at only large earthquakes.
  + Stock market traders may subset their trading data by looking only at the previous day's trades.
* There's *many* ways to subset data using RStudio, we'll focus on learning the most common methods.

## The filter function

* Creating two plots, one for salty and one for sweet is useful for comparing salty and sweet But what if we want to examine only one group by itself?
* Start by creating a subset of the data:
  + Fill in the blanks below with the data and variable names needed to filter our food data based on people who ate Salty snacks:

food\_salty <- filter(\_\_\_\_ , \_\_\_\_ == "Salty")

* **View food\_salty and write down the number of observations in it. Then use the subset data to make a dotPlot of the sodium in our Salty snacks.**

## So what's really going on?

* Coding in R is really just about supplying directions in a way that R understands.
  + We'll start by focusing on everything to the right of the "<-" symbol

food\_salty <- filter(\_\_\_\_ , \_\_\_\_ == "Salty")

* filter() tells R that we're going to look at only the values in our data that follow a *rule*.
* The first blank should be the data we're going to filter down into a smaller set (Based on our rule).
* salty\_sweet == "Salty" is the rule to follow.

## 3 parts of defining rules

* We can decompose our rule, salty\_sweet == "Salty", into 3 parts:
  + 1. salty\_sweet, is the particular *variable* we want to use to select our subset.
    2. "Salty", is the *value* of the variable that we want to select. We only want to see data with the value Salty for the variable salty\_sweet.
    3. == describes how we want to relate our variable (salty\_sweet) to our value ("Salty"). In this case, we want values of salty\_sweet that are *exactly equal* to "Salty".
* Notice: *Values* (that are also words) have quotation marks around them. *Variables* do not.

## More on ==

* We can use the head() function to help us see what's happening when we write salty\_sweet == "Salty".
  + head() returns the values of the first 6 observations.
  + The tail() function returns the last 6 observations.
* Run the following code and answer the question below:

head(~salty\_sweet == "Salty", data = food)

* **What do the values TRUE and FALSE tell us about how our *rule* applies to the first six snacks in our data? Which of the first six observations were Salty?**

## Saving values

* To use our subset data we need to save it first.
  + When we *save* something in R what we are really doing is giving a value, or set of values, a specific name for us to use later.
* The arrow <- is called the "assignment" operator. It assigns names (on the left) to values (on the right)
  + We now focus on everything to the left of, and including, the "<-" symbol

food\_salty <- filter(\_\_\_\_ , \_\_\_\_ == "Salty")

## Saving our subset

food\_salty <- filter(\_\_\_\_ , \_\_\_\_ == "Salty")

* This code then:
  + takes our subset data, (everything to the right of "<-") ...
  + and assigns the subset data, by using the arrow "<-" ...
  + the name food\_salty.
* We can now use food\_salty to do anything we could do with the regular food data ...
  + but only including those snacks who reported being Salty.

## Including more filters

* We often want to filter our data based on multiple rules.
  + For instance, we might want to filter our food data based on the food being salty AND costing less than 2 dollars.
* We can include multiple filters to our subsets by separating each rule with a comma like so:

my\_sub <- filter(food , salty\_sweet == "Salty", cost <= 2)

* View the my\_sub data we filtered in the above line of code and verify that it only includes salty snacks that cost less than 2 dollars.

## Put it all together

* **Use an appropriate dotPlot to answer each of the following questions:**
  + **About how much fat does the typical sweet snack have?**
  + **How does the typical amount of fat compare when healthy\_level < 3 and when healthy\_level > 3?**
* It can sometime be helpful to change the number of *intervals*, or *bins*, used in dotPlots and histograms.
  + To change the number of *intervals* in your plots, include the nint option. For instance, to have a plot with 3 bins, use nint = 3. To have a plot with 30 bins, use nint = 30.