Problem Set 5

Insert Name

Stat 108, Week 8

Collaborators

I collaborated with... (list names of collaborators here).

```
# Put all necessary libraries here
library(tidyverse)
```

Due: Wednesday, March 29th at 10:00pm

Goals of this problem set

1. Practice controlling the flow of your `R` code.
2. Practice creating and modifying functions.

Note

In some of your chunks, you will be testing your functions and want to determine how your functions behave when they error out. For those chunks, include `error = TRUE` in the chunk options so that your document still knits (but also displays the error message).

Problem 1

For this problem you will practice writing conditional statements. Make sure to test out your conditionals using data from `trees`, the 8 trees around Portland's Woodstock Community Center.

```
library(pdxTrees)
trees <- get_pdxTrees_parks(park = "Woodstock Community Center") %>%
  select(Common_Name, DBH, Condition, Tree_Height,
         Native, Edible)
```

a. Write a set of conditional(s) that satisfies the following requirements: If any of the trees in `trees` is a "Sweetgum" and taller than 60 feet, print out "Tall Sweetgums found".

b. Write a set of conditional(s) that satisfies the following requirements: If any of the trees in `trees` do not have missing information for whether they are edible, print out "Some values are not missing."

c. Write a set of conditional(s) that satisfies the following requirements: If all of the Sweetgum trees in `trees` are taller than 80 feet, print out "All Sweetgums found are very tall." If only some of the Sweetgum trees in 'trees are taller than 80 feet, print out "Some Sweetgums found are very tall."

For Problems 2-4, feel free to keep using `trees` for testing but you might also want a larger dataset for testing:

```
pdxTrees <- get_pdxTrees_parks()
```

Problem 2

Figure out what the following code does and then turn it into a function. For your new function, do the following:

- Test it.
- Provide default values (when appropriate).
- Use clear names for the function and arguments.
- Make sure to appropriately handle missingness.
  - Hint: `length()` provides the number of entries (including `NA`s) of a vector.

- Check that any data inputs are the appropriate classes.
  - And, provide a helpful error message if they aren't.
- Generalize it by allowing the user to specify a confidence level.
- Provide a warning message if the sample size is less than 30 using a conditional statement and `message()`.
  - In your message, report the issue and their sample size.

```
thing1 <- length(trees$DBH)
thing2 <- mean(trees$DBH)
thing3 <- sd(trees$DBH)/sqrt(thing1)
thing4 <- qt(p = .975, df = thing1 - 1)
thing5 <- thing2 - thing4*thing3
thing6 <- thing2 + thing4*thing3
```

Problem 3

While we (i.e. Stat 108 students) all love the grammar of graphics, not everyone does. For this problem, we are going to practice creating wrapper functions for `ggplot2`.

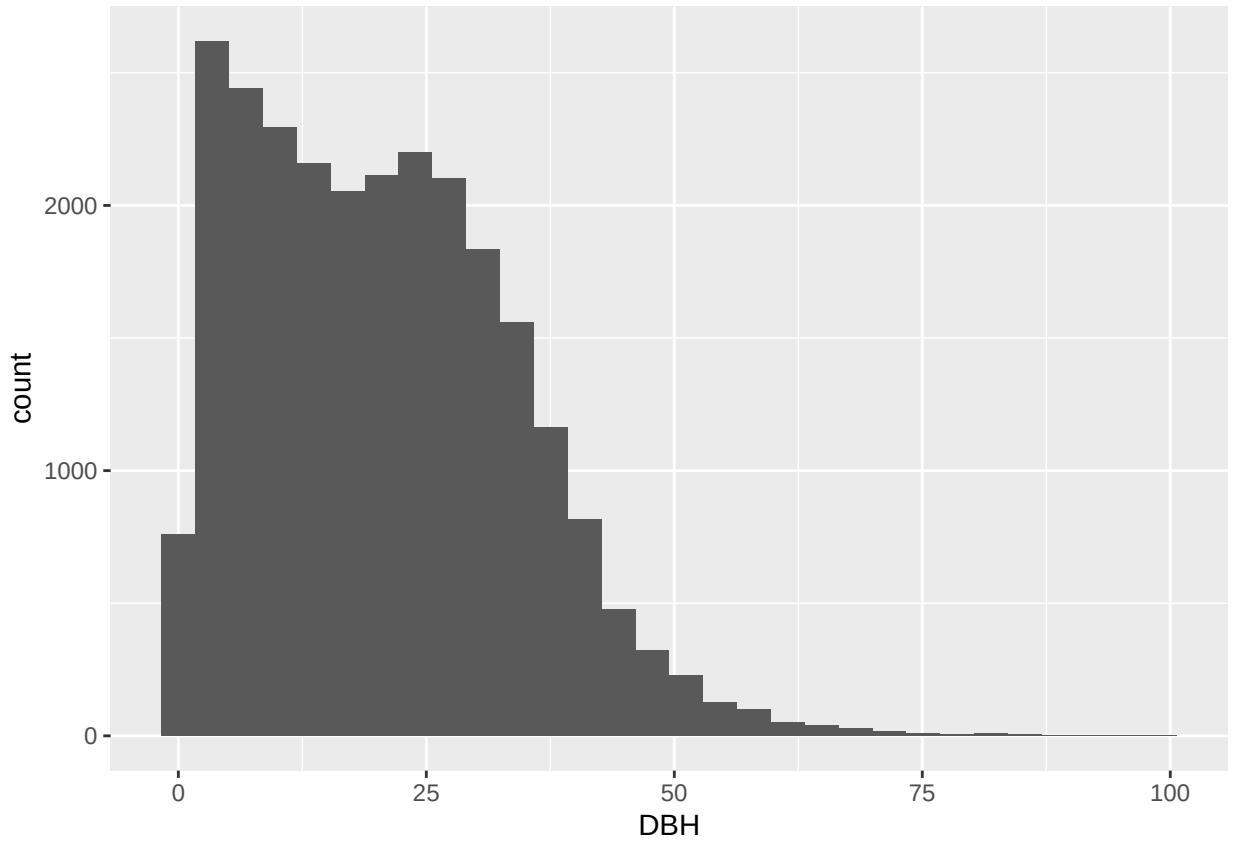
Recall our discussion from class on tidy evaluation. If you want to learn more, check out these pages:

- For using `ggplot2` functions in your own functions
- For using `dplyr` functions in your own functions

Here's our example of a wrapper for a histogram.

```
# Minimal viable product working code
ggplot(data = pdxTrees, mapping = aes(x = DBH)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

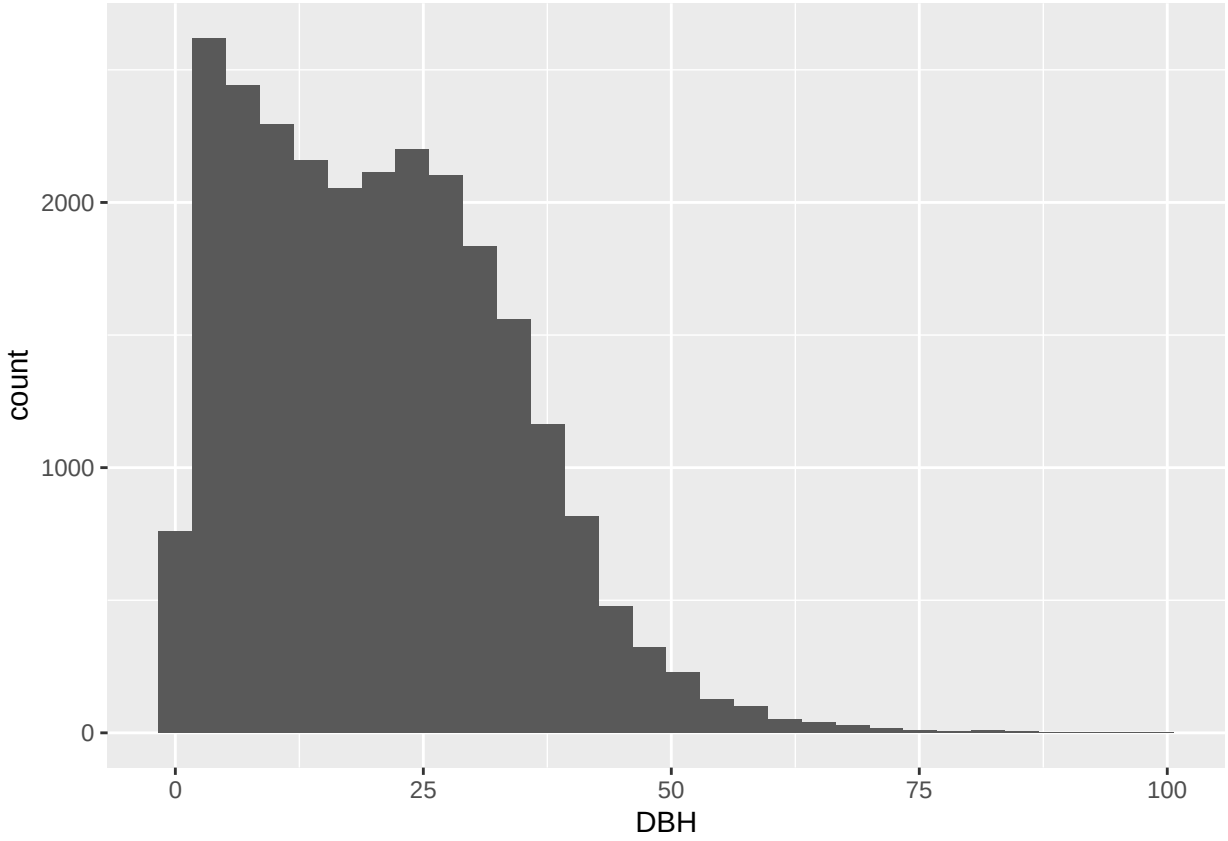



```
# Function
histo <- function(data, x, ...){
  #Determine if x is numeric
  x_class <- data %>%
    pull({{ x }}) %>%
    is.numeric()
  stopifnot(x_class)

  ggplot(data = data, mapping = aes(x = {{ x }})) +
    geom_histogram()
}

# Test it
histo(pdxTrees, DBH)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



a. Edit `histo()` so that the user can set

- The number of bins
- The fill color for the bars
- The color outlining the bars

b. Write code to create a basic scatterplot with `ggplot2`. Then write and test a function to create a basic scatterplot.

c. Modify your scatterplot function to allow the user to:

- Color the points by another variable.
- Set the transparency.
  - And include a check that the transparency input is within the appropriate range.

Also check the inputs.

d. Write and test a function for your favorite `ggplot2` graph. Make sure to give the user at least 3 optional inputs that they can change to customize the plot.

Problem 4:

Who thinks it is a bit clunky to get conditional proportions using `dplyr`? Let's practice writing functions for common data wrangling operations.

a. Take the following code and turn it into an R function to create a conditional proportions table. Similar to `ggplot2`, you will need to handle the tidy evaluation. And, make sure to test your function!

```
pdxTrees %>%
  count(Native, Condition) %>%
  group_by(Native) %>%
```

```
  mutate(prop = n/sum(n)) %>%
  ungroup()
```

```
## # A tibble: 10 x 4
##    Native Condition     n    prop
##    <chr>  <chr>     <int>   <dbl>
##  1 No     Fair      12284 0.865
##  2 No     Good       1043 0.0734
##  3 No     Poor        875 0.0616
##  4 Yes    Fair       9877 0.904
##  5 Yes    Good        600 0.0549
##  6 Yes    Poor        454 0.0415
##  7 <NA>   Dead        264 0.658
##  8 <NA>   Fair        118 0.294
##  9 <NA>   Good          3 0.00748
## 10 <NA>   Poor         16 0.0399
```

b. Write a function to compute the mean, median, sd, min, max, sample size, and number of missing values of a quantitative variable by the categories of another variable. Make sure the output is a data frame (or tibble).