Lab 3

Math 241, Week 3

```
libs <- c('tidyverse','knitr','viridis', 'mosaic','mosaicData','babynames', 'Lahman','nycflights13')
for(l in libs){
  if(!require(l,character.only = TRUE, quietly = TRUE)){
    message( sprintf('Did not have the required package << %s >> installed. Downloading now ... ',l))
    install.packages(l)
  }
  library(l, character.only = TRUE, quietly = TRUE)
}
```

Due: Friday, February 16th at 8:30am

Goals of this lab

1. Practice creating functions.
2. Practice refactoring your code to make it better! Therefore for each problem, make sure to test your functions.

Problem 1: Subset that R Object

Here are the R objects we will use in this problem (`dats`, `pdxTreesSmall` and `ht`).

```
library(pdxTrees)
library(mosaicData)

pdxTrees <- get_pdxTrees_parks()
# Creating the objects
dats <- list(pdxTrees  = head(pdxTrees),
             Births2015 = head(Births2015),
             HELPrct = head(HELPrct),
             sets = c("pdxTrees", "Births2015",
                      "HELPrct"))

pdxTreesSmall  <- head(pdxTrees)

ht <- head(pdxTrees$Tree_Height, n = 15)
```

a. What are the classes of `dats`, `pdxTreesSmall` and `ht`?

```
class(dats)
```

```
## [1] "list"
```

```
class(pdxTreesSmall)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
class(ht)
```

```
## [1] "numeric"
```

The class of `dats` is a list, the class of `pdxTreesSmall` is a dataframe, and the class of ht is a 'numeric.

b. Find the 10th, 11th, and 12th values of `ht`.

```
ht[10:12]
```

```
## [1] 112 112  48
```

c. Provide the `Species` column of `pdxTrees` as a data frame with one column.

```
species_df <- data.frame(Species = pdxTrees$Species)
head(species_df)
```

```
##   Species
## 1    PSME
## 2    PSME
## 3    CRLA
## 4    QURU
## 5    PSME
## 6    PSME
```

d. Provide the `Species` column of `pdxTrees` as a character vector.

```
species_vec <- pdxTrees$Species
head(species_vec)
```

```
## [1] "PSME" "PSME" "CRLA" "QURU" "PSME" "PSME"
```

e. Provide code that gives us the second entry in `sets` from `dats`.

```
second_entry <- dats$sets[2]
second_entry
```

```
## [1] "Births2015"
```

f. Subset `pdxTreesSmall` to only `Douglas-fir` and then provide the `DBH` and `Condition` of the 4th `Douglas-fir` in the dataset. (Feel free to mix in some `tidyverse` code if you would like to.)

```r
douglas_fir <- pdxTreesSmall %>%
  filter(Common_Name == "Douglas-Fir")

douglas_fir[4, c("DBH", "Condition")]
```

```
## # A tibble: 1 x 2
##     DBH Condition
##   <dbl> <chr>
## 1  32.1 Fair
```

Problem 2: Function Creation

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.
- Generalize it by allowing the user to specify a confidence level.
- Check the inputs and stop the function if the user provides inappropriate values.

```r
library(pdxTrees)
thing1 <- length(pdxTrees$DBH) # finds the length of the DBH vector in the pdxTrees dataset
thing2 <- mean(pdxTrees$DBH) # calculates the mean of the DBH vector in the pdxTrees dataset
thing3 <- sd(pdxTrees$DBH)/sqrt(thing1) #finds the of DBH and divides it by the sqrt of the length, pro
thing4 <- qt(p = .975, df = thing1 - 1) #Calculates the t-value for a two-tailed 95% confidence interva
thing5 <- thing2 - thing4*thing3 #FINAL OUTPUT - Calculates the lower bound of the 95% confidence inter
thing6 <- thing2 + thing4*thing3 #FINAL OUTPUT - Calculates the upper bound of the 95% confidence inter
```

```r
calculate_ci <- function(data_vec){
  n <- length(data_vec)
  mean <- mean(data_vec)
  sd <- sd(data_vec)
  se <- sd / sqrt(n)
  t_value <- qt(p = 0.975, df = n - 1)

  lower_bound <- mean - t_value * se
  upper_bound <- mean + t_value * se

  c(lower_bound = lower_bound, upper_bound = upper_bound)
}

calculate_ci(pdxTrees$DBH)
```

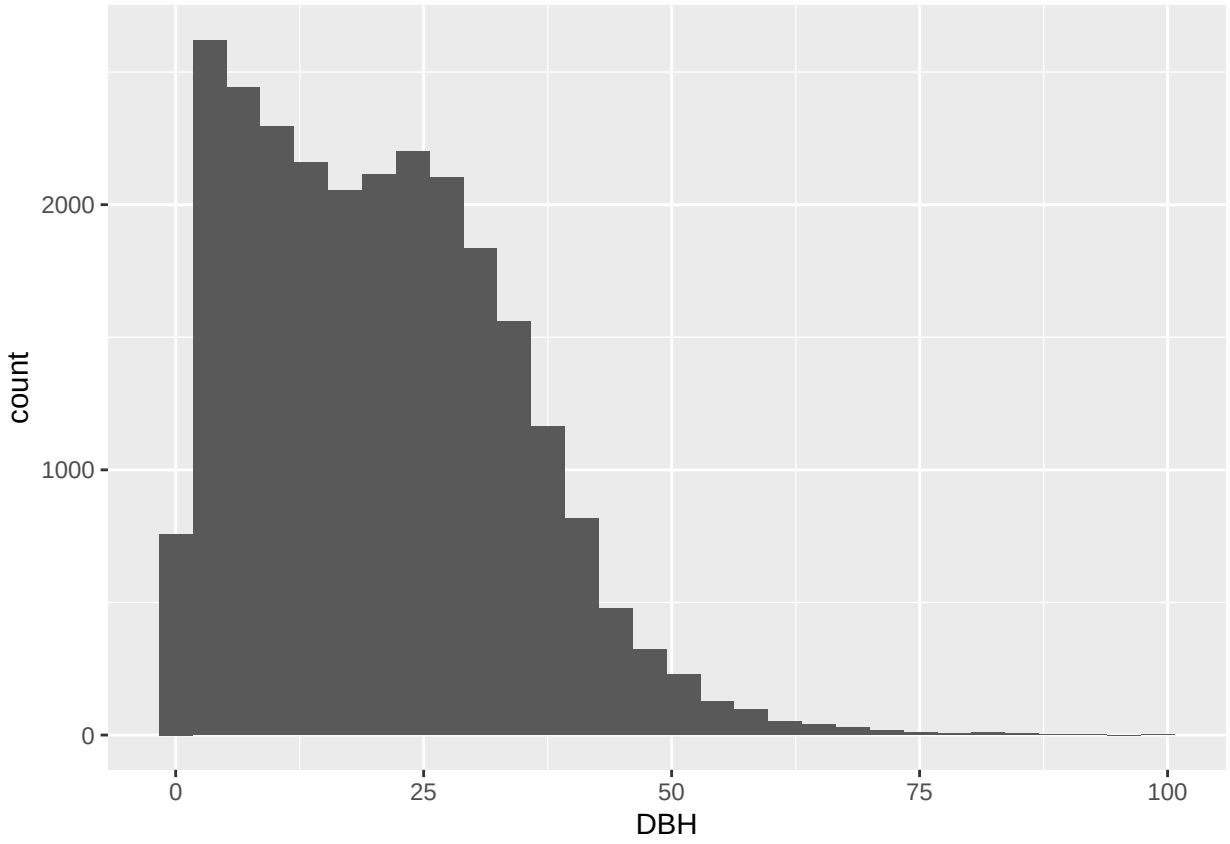
```
## lower_bound upper_bound
##    20.44981    20.77835
```

Problem 3: Wrapper Function for your `ggplot`

While we (i.e. Math 241 students) all love the grammar of graphics, not everyone else does. So for this problem, we are going to practice creating wrapper functions for `ggplot2`.

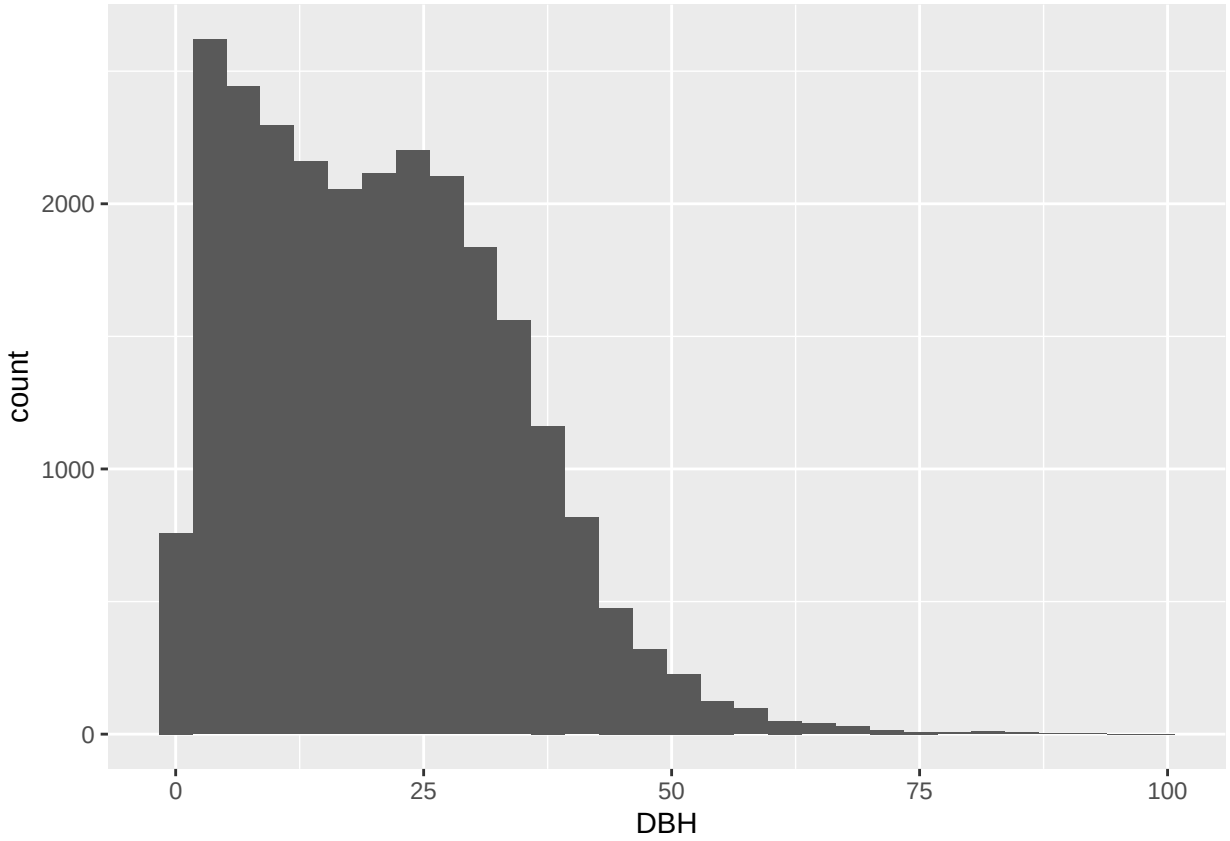
Here's an example of a wrapper for a histogram. Notice that I can't just list the variable name as an argument. The issue has to do with how many of the `tidyverse` functions evaluate the arguments. Therefore we have to quote (`enquo()`) and then unquote (`!!`) the arguments. (If you want to learn more, go here.)

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




```
# Shorthand histogram function
histo <- function(data, x){
   x <- enquo(x)
  ggplot(data = data, mapping = aes(x = !!x)) +
    geom_histogram()
}

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

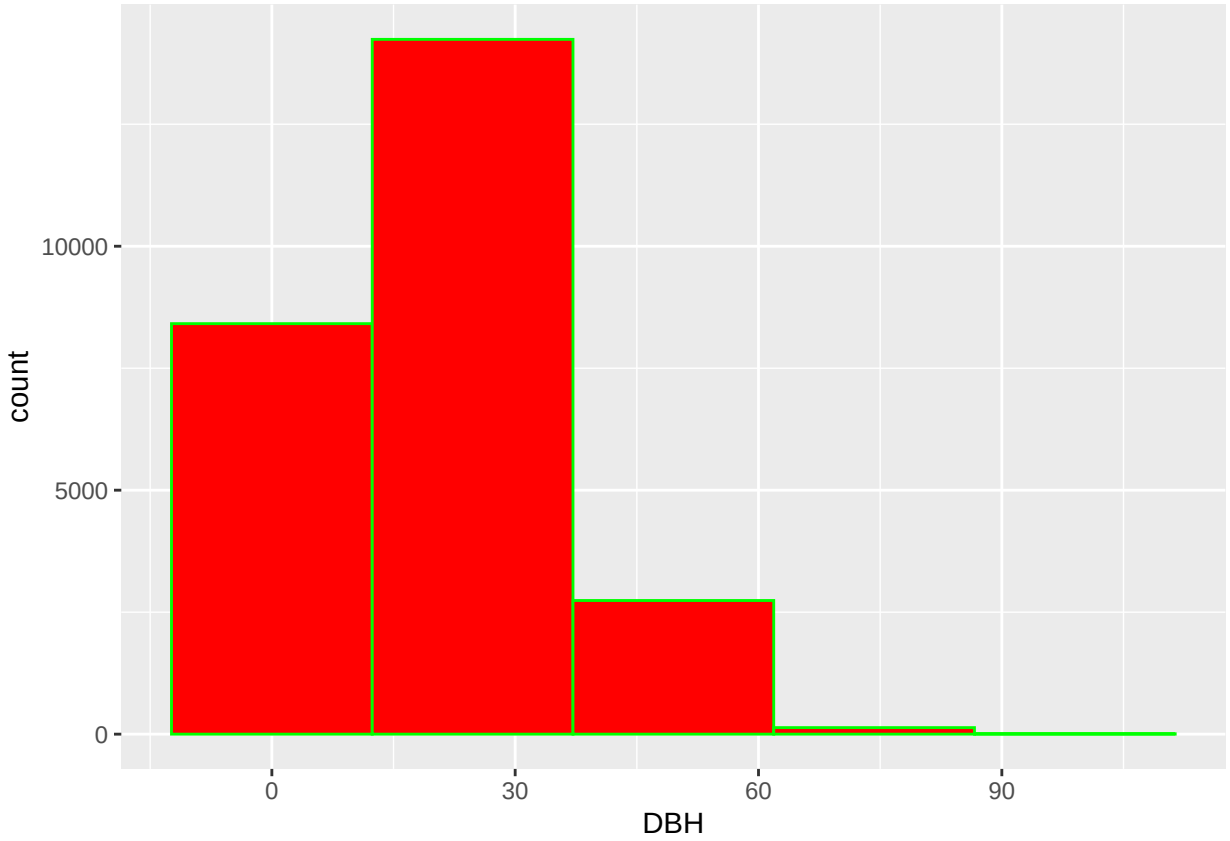


a. Edit `histo()` so that the user can set

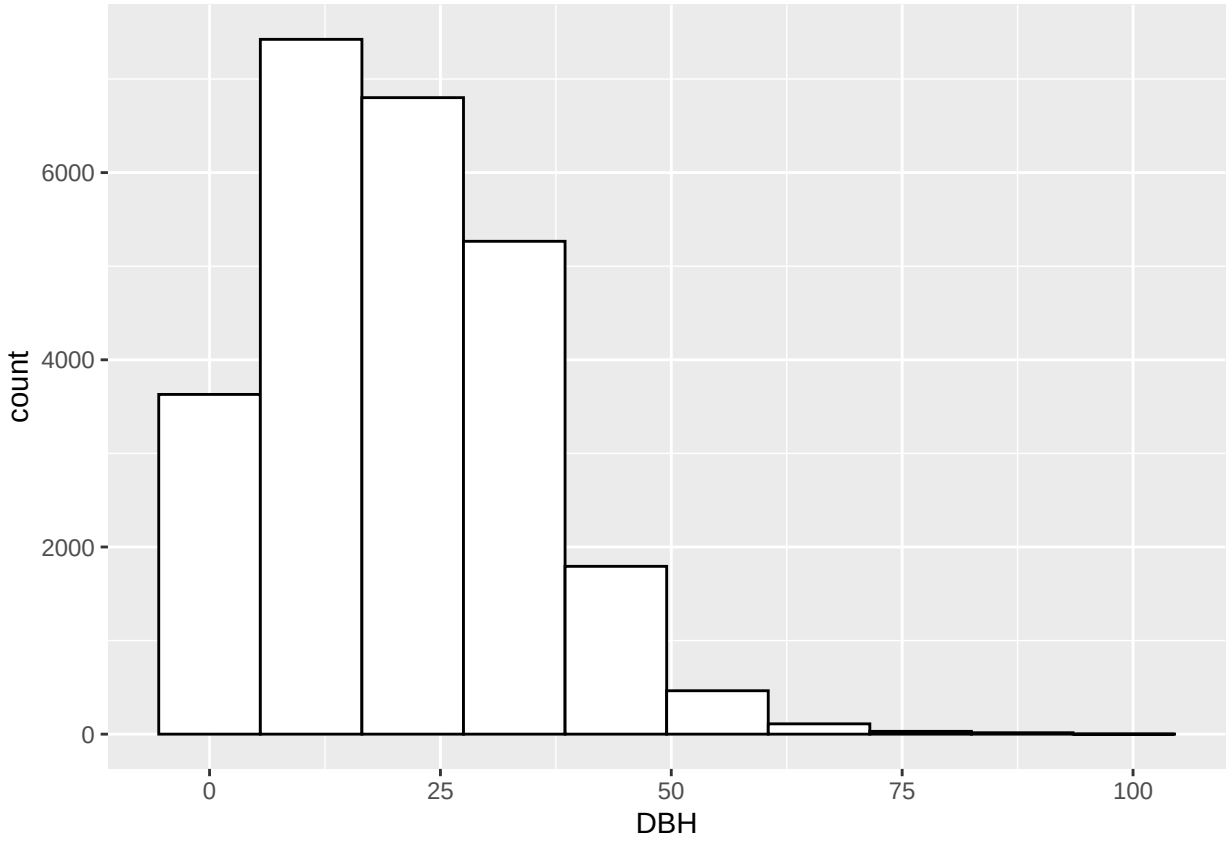
- The number of bins
- The fill color for the bars
- The color outlining the bars

```
histo <- function(data, x, bins, fill = "white", color = "black"){
   x <- enquo(x)
  ggplot(data = data, mapping = aes(x = !!x)) +
    geom_histogram(bins = bins, fill = fill, color = color)
}

histo(pdxTrees, DBH, 5, "red", "green")
```



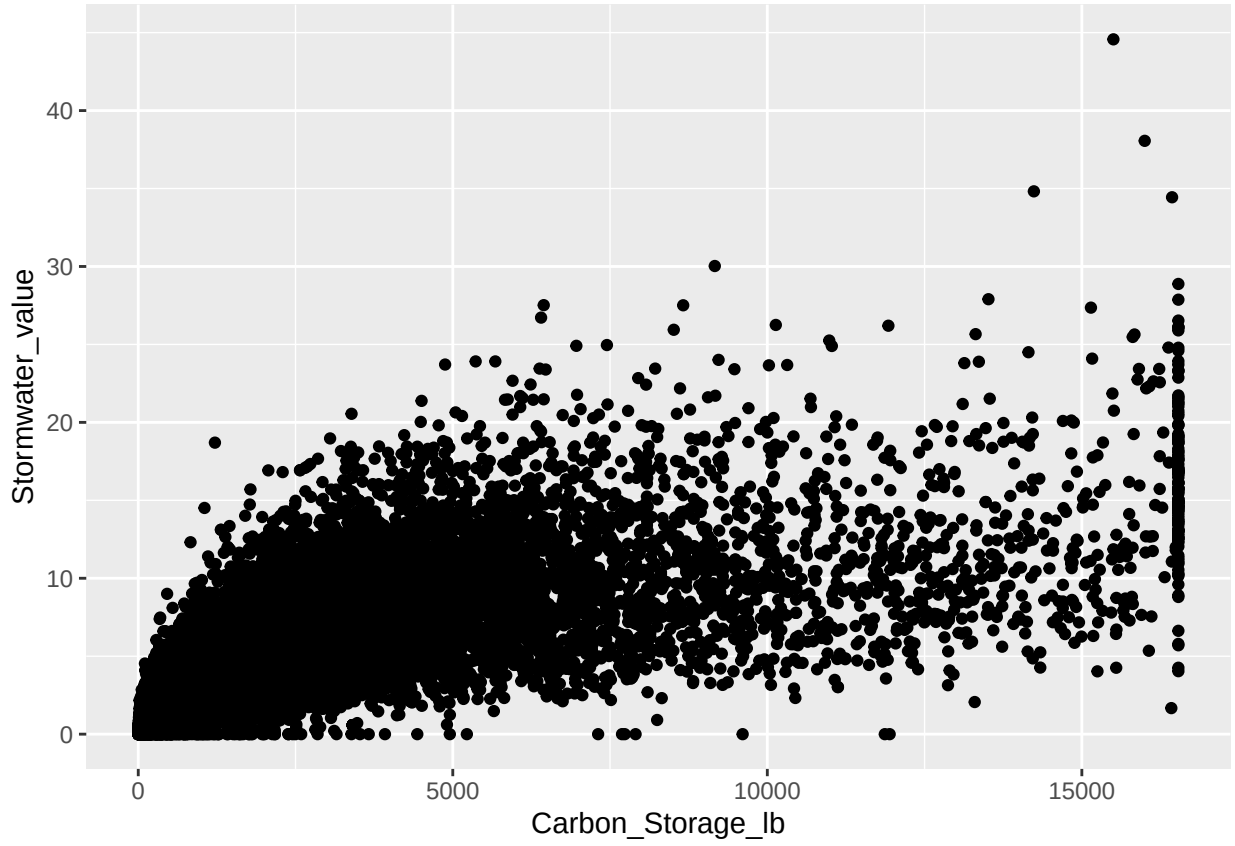

```
histo(pdxTrees, DBH, 10)
```



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

```
create_scatter <- function(data, x, y){
  x <- enquo(x)
  y <- enquo(y)
  ggplot(data = data, aes(x = !!x, y = !!y)) +
    geom_point()
}

create_scatter(pdxTrees, Carbon_Storage_lb, Stormwater_value)
```

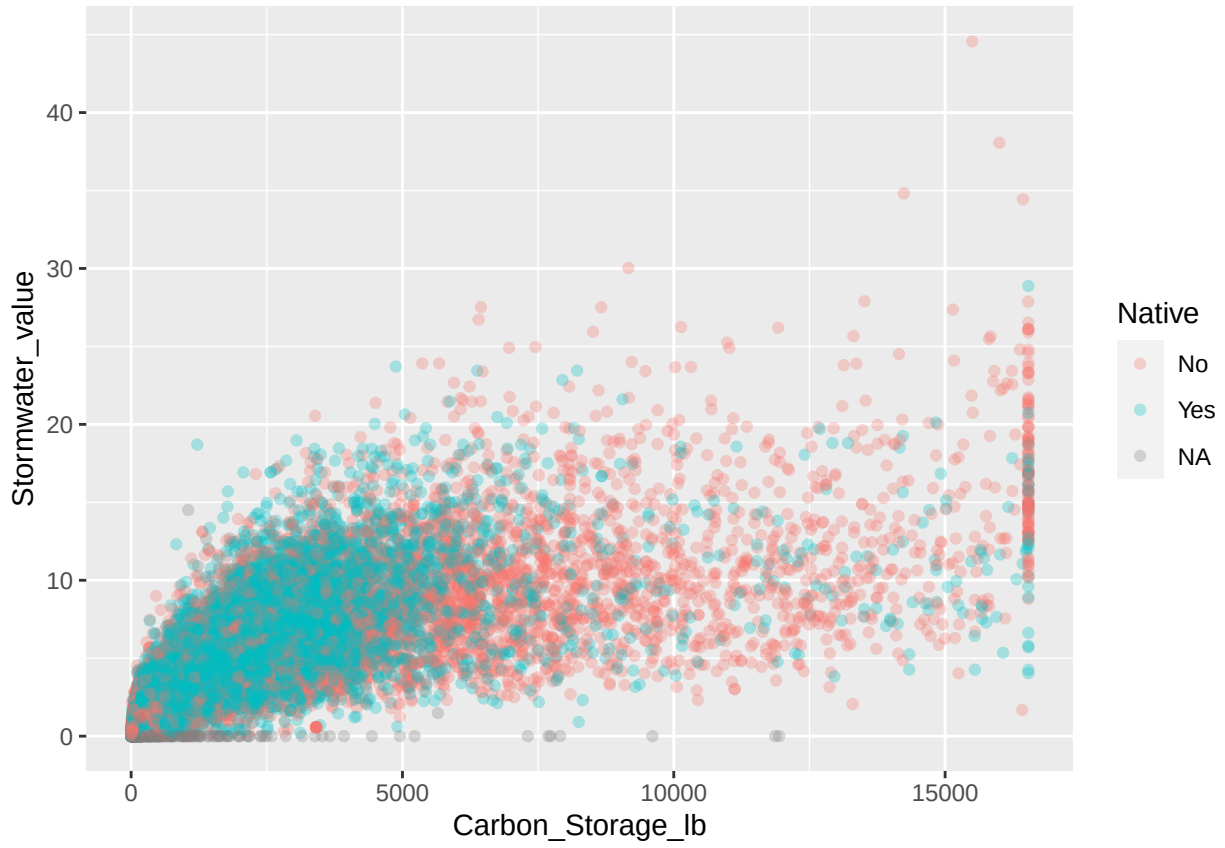


c. Modify your scatterplot function to allow the user to ...

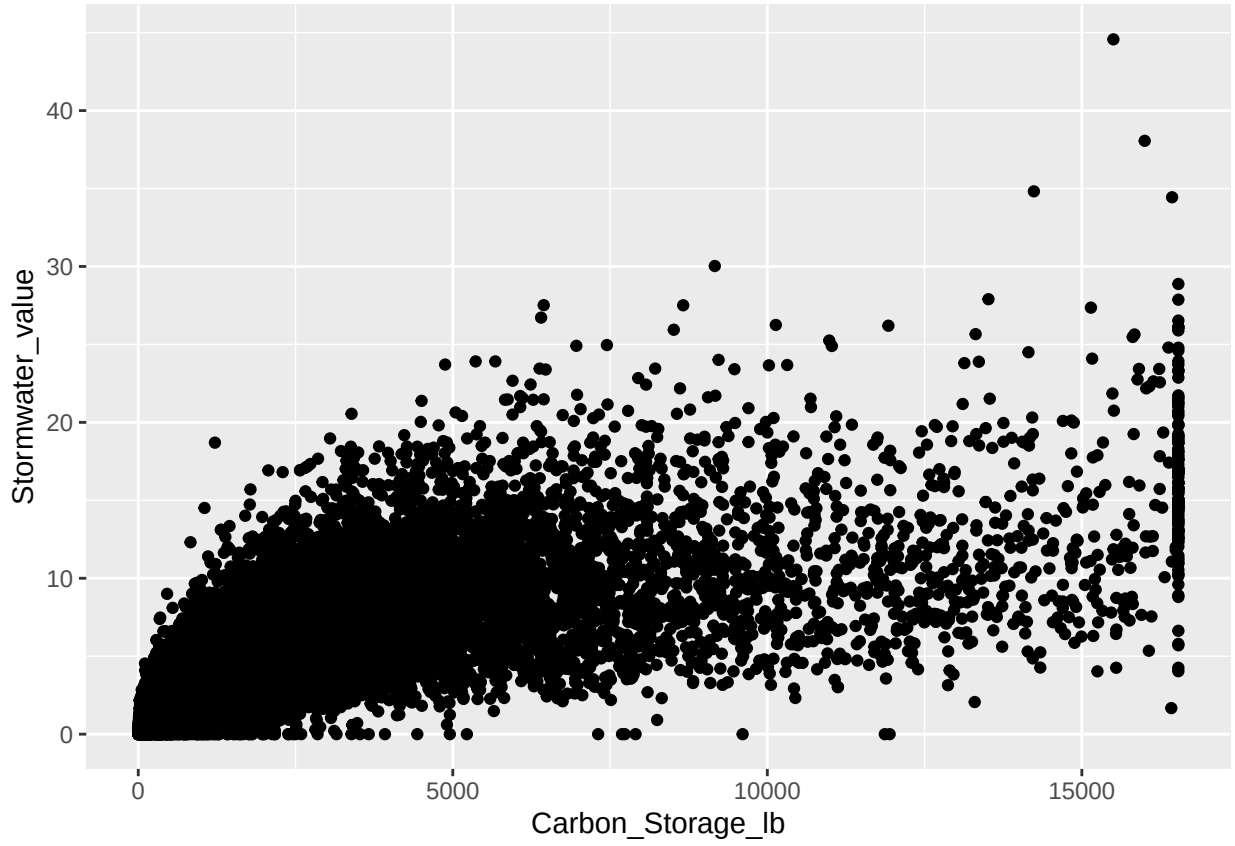
- Color the points by another variable.
- Set the transparency.

```
create_scatter <- function(data, x, y, color, alpha = 1){
  x <- enquo(x)
  y <- enquo(y)
  color <- enquo(color)
  ggplot(data = data, aes(x = !!x, y = !!y, color = !!color)) +
    geom_point(alpha = alpha)
}

create_scatter(pdxTrees, Carbon_Storage_lb, Stormwater_value, Native, 0.3)
```



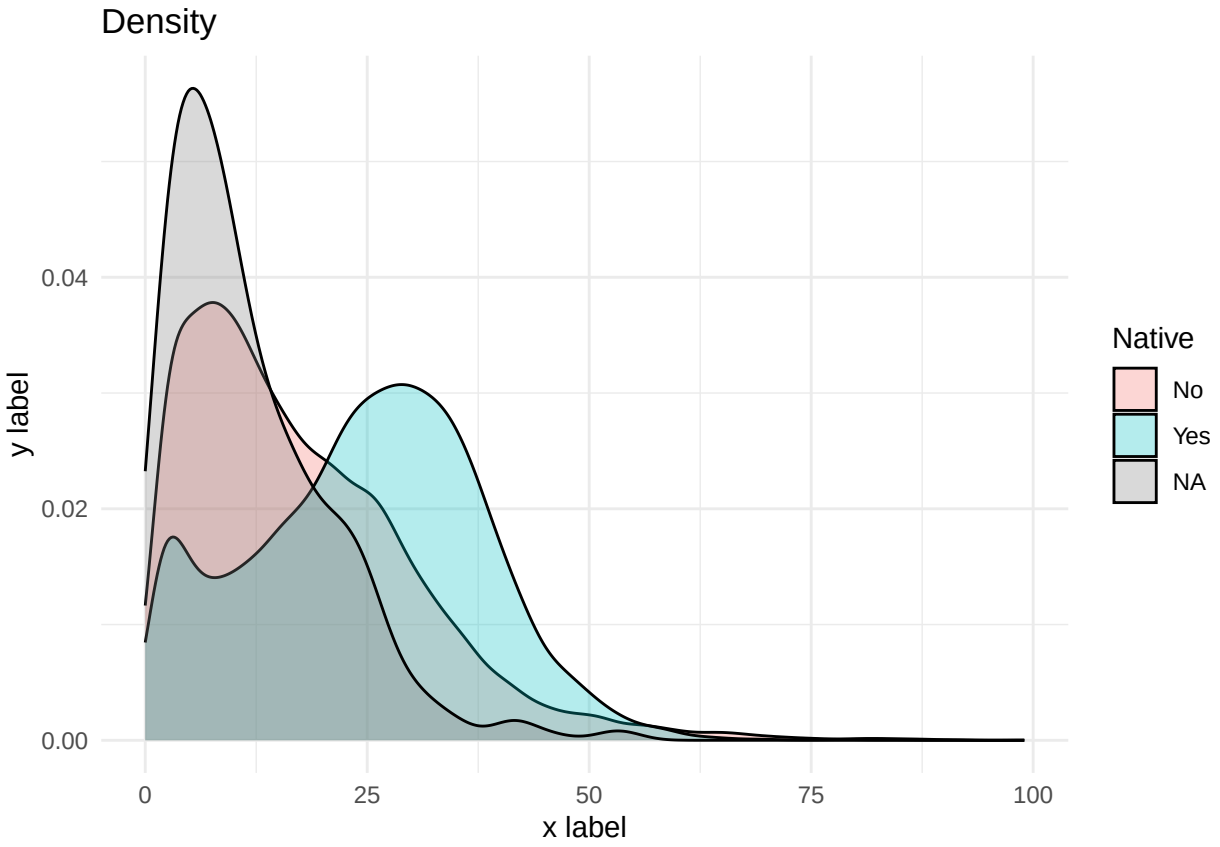

```
create_scatter(pdxTrees, Carbon_Storage_lb, Stormwater_value)
```



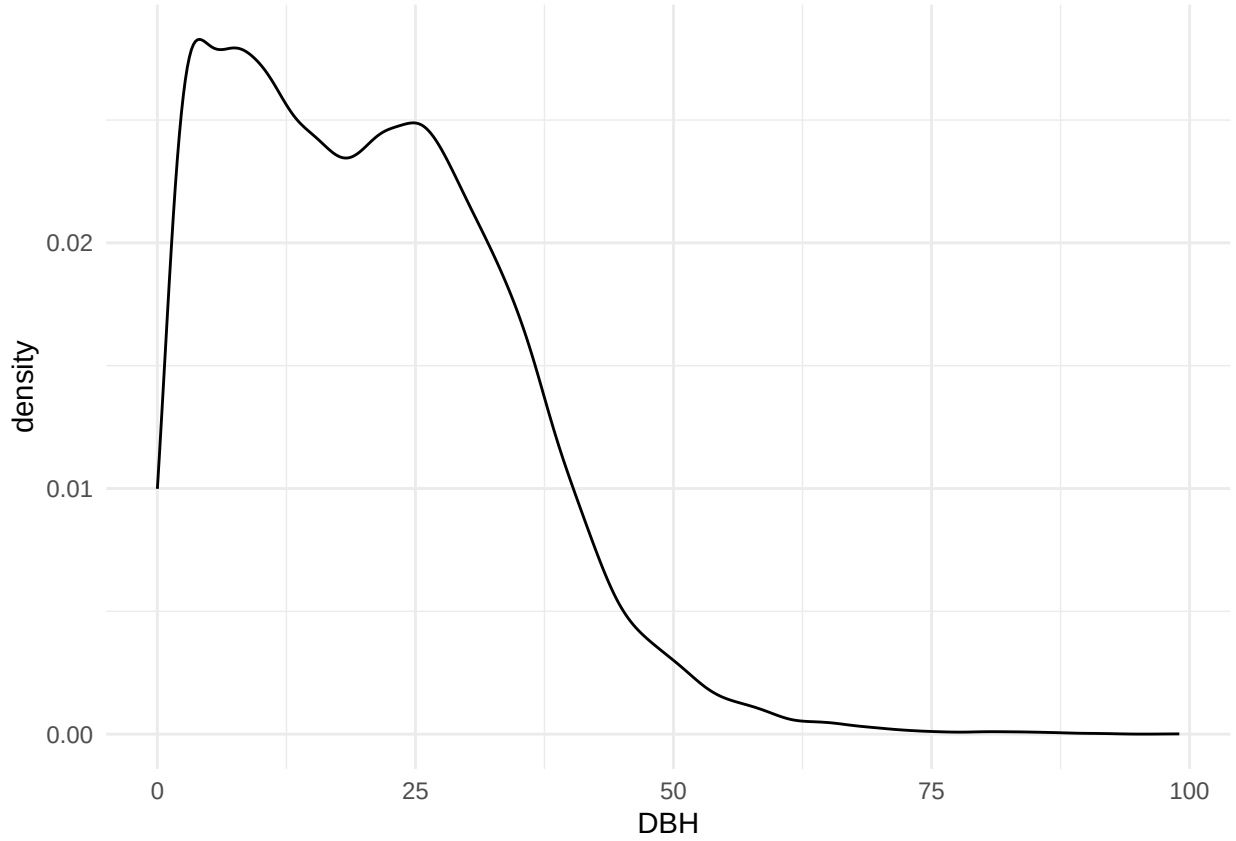
d. Write and test a function for your favorite `ggplot2` graph.

```
create_density <- function(data, x, fill, alpha = 1, title = waiver(), xlabel = waiver(), ylabel = waiv
  x <- enquo(x)
  fill <- enquo(fill)
  ggplot(data, aes(x=!!x, fill = !!fill)) +
    geom_density(alpha = alpha) +
    theme_minimal() +
    labs(title = title,
         x = xlabel,
         y = ylabel)
}

create_density(pdxTrees, DBH, Native, 0.3, "Density", "x label", "y label")
```




```
create_density(pdxTrees, DBH)
```



Problem 4: Functioning `dplyr`

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 quote and unquote the variable names. Make sure to test your function!

```
pdxTrees %>%
  count(Native, Condition) %>%
  group_by(Native) %>%
  mutate(prop = n/sum(n)) %>%
  ungroup() # creates a dataframe of with proportions of native and non native trees that are in each c
```

```
## # 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
```

```
create_prop_table <- function(data, group, type){
  group <- enquo(group)
  type <- enquo(type)
  data %>%
    count(!!group, !!type) %>%
    group_by(Native) %>%
    mutate(prop = n/sum(n)) %>%
    ungroup()
}

create_prop_table(pdxTrees, Native, Condition)
```

```
## # 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). Don't forget to test your function.

```
calculate_summary_stats <- function(data, var1, var2){
  var1 <- enquo(var1)
  var2 <- enquo(var2)
  data %>%
    group_by(!!var1) %>%
    summarise(mean = mean(!!var2),
              med = median(!!var2),
              sd = sd(!!var2),
              min = min(!!var2),
              size = nrow(!!var2),
              NAs = sum(is.na(!!var2)))
}

calculate_summary_stats(pdxTrees, Native, Edible)
```

```
## # A tibble: 3 x 6
##   Native  mean med      sd min     NAs
##   <chr>  <dbl> <chr> <dbl> <chr> <int>
## 1 No        NA <NA>     NA <NA>  13279
## 2 Yes       NA <NA>     NA <NA>  10589
## 3 <NA>      NA <NA>     NA <NA>    362
```

```
calculate_summary_stats(pdxTrees, Native, DBH)
```

```
## # A tibble: 3 x 6
##   Native  mean   med    sd   min   NAs
##   <chr>  <dbl> <dbl> <dbl> <dbl> <int>
## 1 No      17.6  14.8 12.9    0       0
## 2 Yes     24.9  25.8 12.9    0       0
## 3 <NA>    11.7   9.1  9.06   0.2     0
```

Problem 5: another `babynames` exercise

Write a function called grab_name that, when given a **name *and a year*** as an argument, returns the rows from the `babynames` data frame in the `babynames` package that match that name for that year (and returns an error if that name and year combination does not match any rows). Run the function once with the arguments **Ezekiel and 1883** and once with **Ezekiel and 1983**.

```
#' Make sure to switch eval = FALSE to eval = TRUE before knitting!!

grab_name <- function(myname, myyear) {
  result <- babynames %>%
    filter(year == myyear, name == myname)

  if (nrow(result) == 0) {
    stop("No matching rows found for the given name and year.")
  } else {
    return(result)
  }
}

grab_name("Ezekiel", 1883)
```

```
## # A tibble: 1 x 5
##    year sex   name        n     prop
##   <dbl> <chr> <chr>   <int>    <dbl>
## 1  1883 M     Ezekiel    14 0.000124
```

```
grab_name("Ezekiel", 1983)
```

```
## # A tibble: 1 x 5
##    year sex   name        n      prop
##   <dbl> <chr> <chr>   <int>     <dbl>
## 1  1983 M     Ezekiel   149 0.0000800
```