Lab 3

Math 241, Week 3

```
libs <- c('tidyverse','knitr','viridis', 'mosaic','mosaicData','babynames', 'Lahman','nycflights13','rn
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"
```

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.

```
pdxTrees_species_df <- select(pdxTrees, Species)

pdxTrees_species_df
```

```
## # A tibble: 25,534 x 1
##    Species
##    <chr>
##  1 PSME
##  2 PSME
##  3 CRLA
##  4 QURU
##  5 PSME
##  6 PSME
##  7 PSME
##  8 PSME
##  9 PSME
## 10 PSME
## # i 25,524 more rows
```

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

```
pdxTrees_species_chr <- head(pdxTrees$Species)
as.character(pdxTrees_species_chr)
```

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

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

```
dats$sets[2]
```

```
## [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
library(dplyr)

pdxTreesSmall %>%
  filter(Species == "Douglas-fir") %>%
  slice(4) %>%
  select(DBH, Condition)
```

```
## # A tibble: 0 x 2
## # i 2 variables: DBH <dbl>, Condition <chr>
```

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

```r
calculate_CI <- function(data, confidence_level = 0.95) {

  if (missing(data) || is.null(data)) {
    stop("Please provide a valid data set.")
  }


  if (confidence_level <= 0 || confidence_level >= 1) {
    stop("Confidence level must be between 0 and 1.")
  }


  n <- length(data)
  mean_val <- mean(data)
  std_err <- sd(data) / sqrt(n)
  t_value <- qt(p = (1 + confidence_level) / 2, df = n - 1)

  lower_bound <- mean_val - t_value * std_err
  upper_bound <- mean_val + t_value * std_err
```

```r
  result <- list(
    mean = mean_val,
    lower_bound = lower_bound,
    upper_bound = upper_bound,
    confidence_level = confidence_level
  )

  return(result)
}

result <- calculate_CI(pdxTrees$DBH, confidence_level = 0.95)
print(result)
```

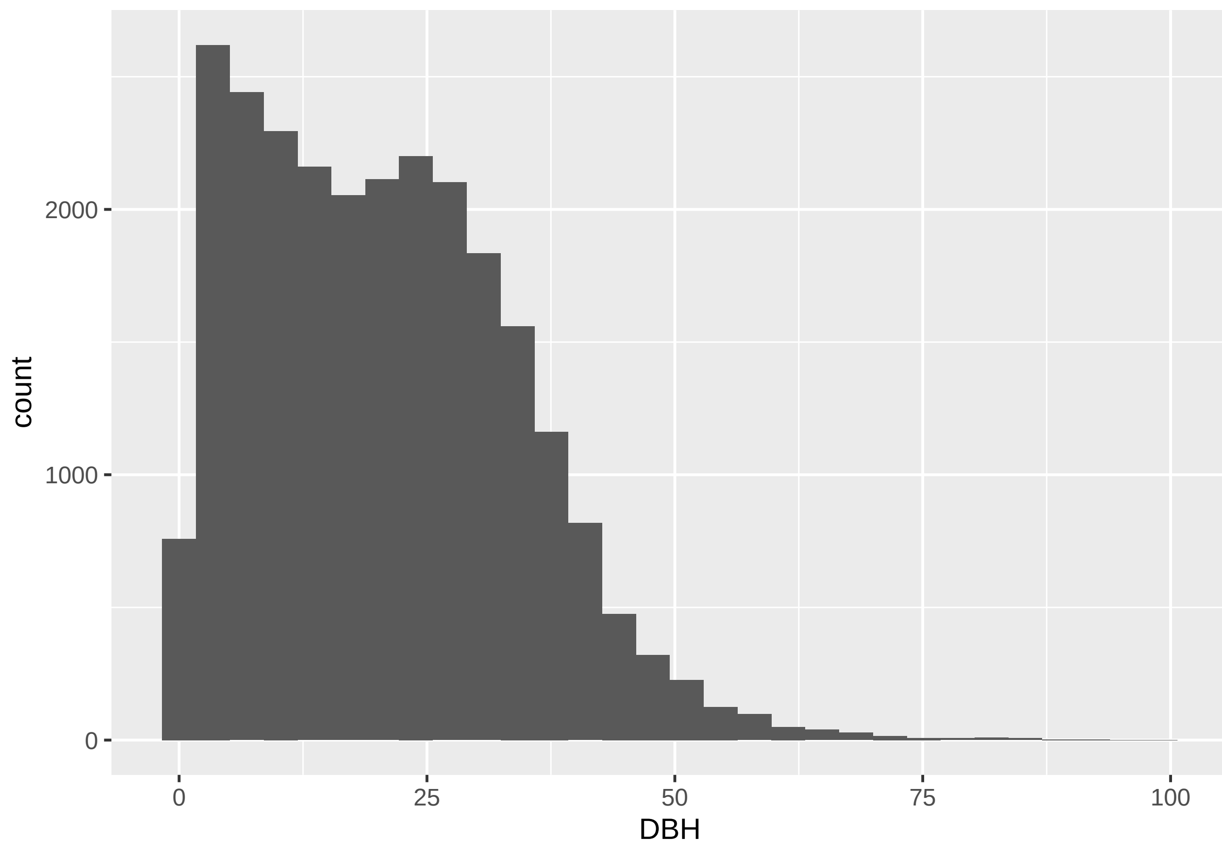
```
## $mean
## [1] 20.61408
##
## $lower_bound
## [1] 20.44981
##
## $upper_bound
## [1] 20.77835
##
## $confidence_level
## [1] 0.95
```

Problem 3: Wrapper Function for your `ggplot`

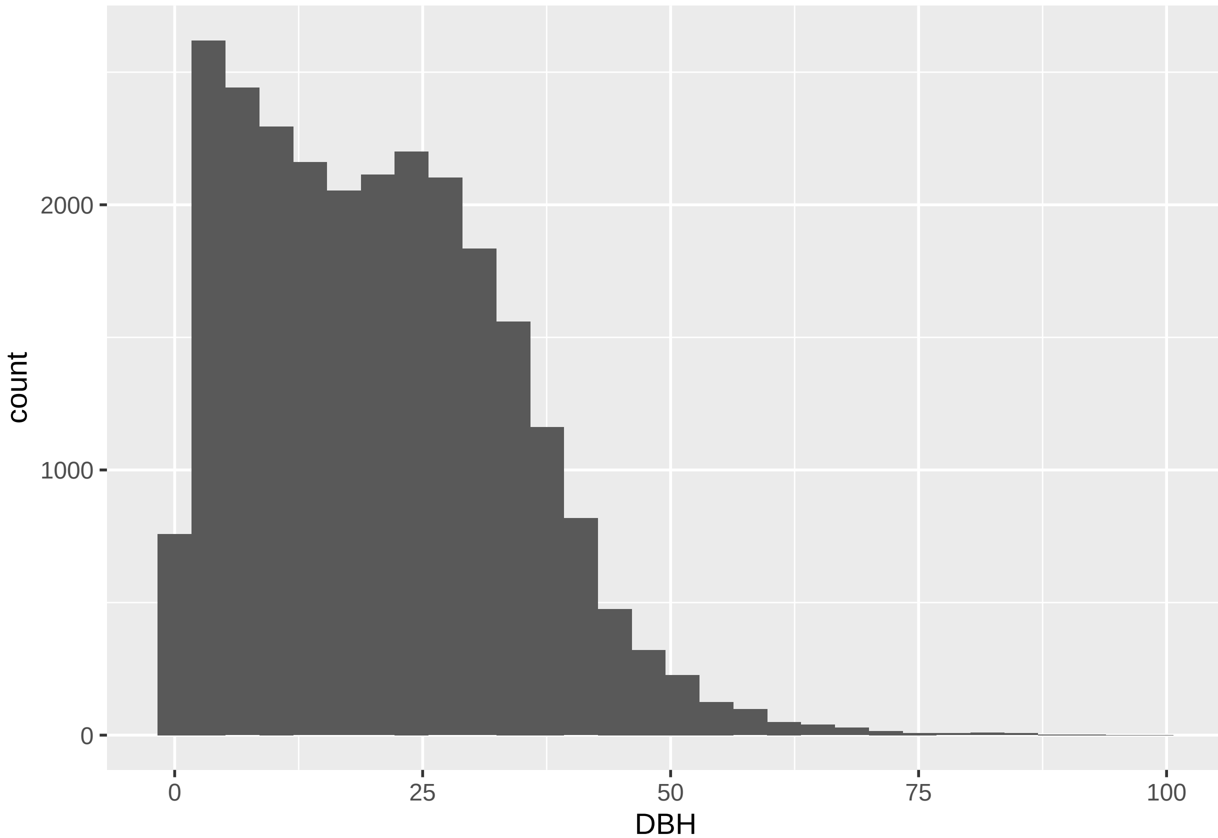
While we (i.e. Math 241 students) all love the grammar of graphics, not everyone else does. So for this problem, we are going to practice creating wrapper functions for `ggplot2`.

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.)

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



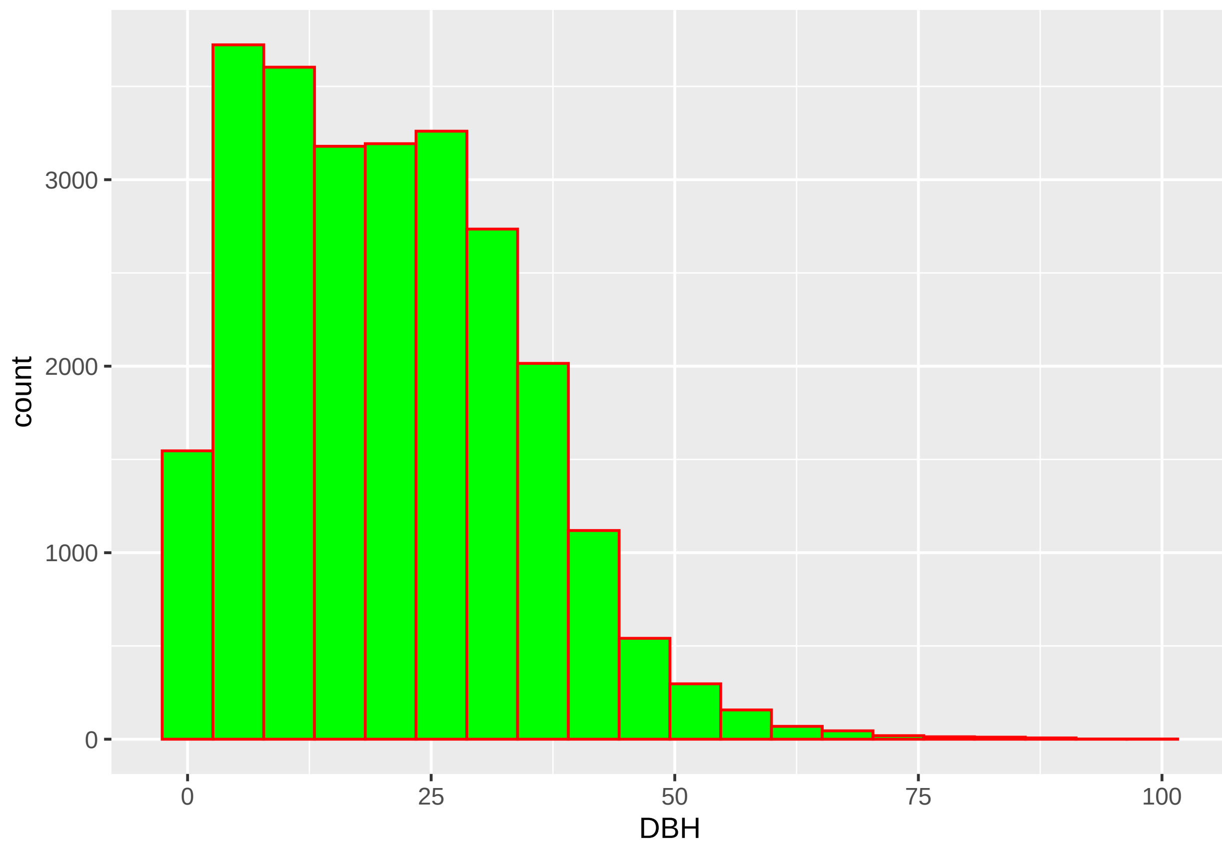

```r
# Shorthand histogram function
histo <- function(data, x){
   x <- enquo(x)
  ggplot(data = data, mapping = aes(x = !!x)) +
    geom_histogram()
}

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



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

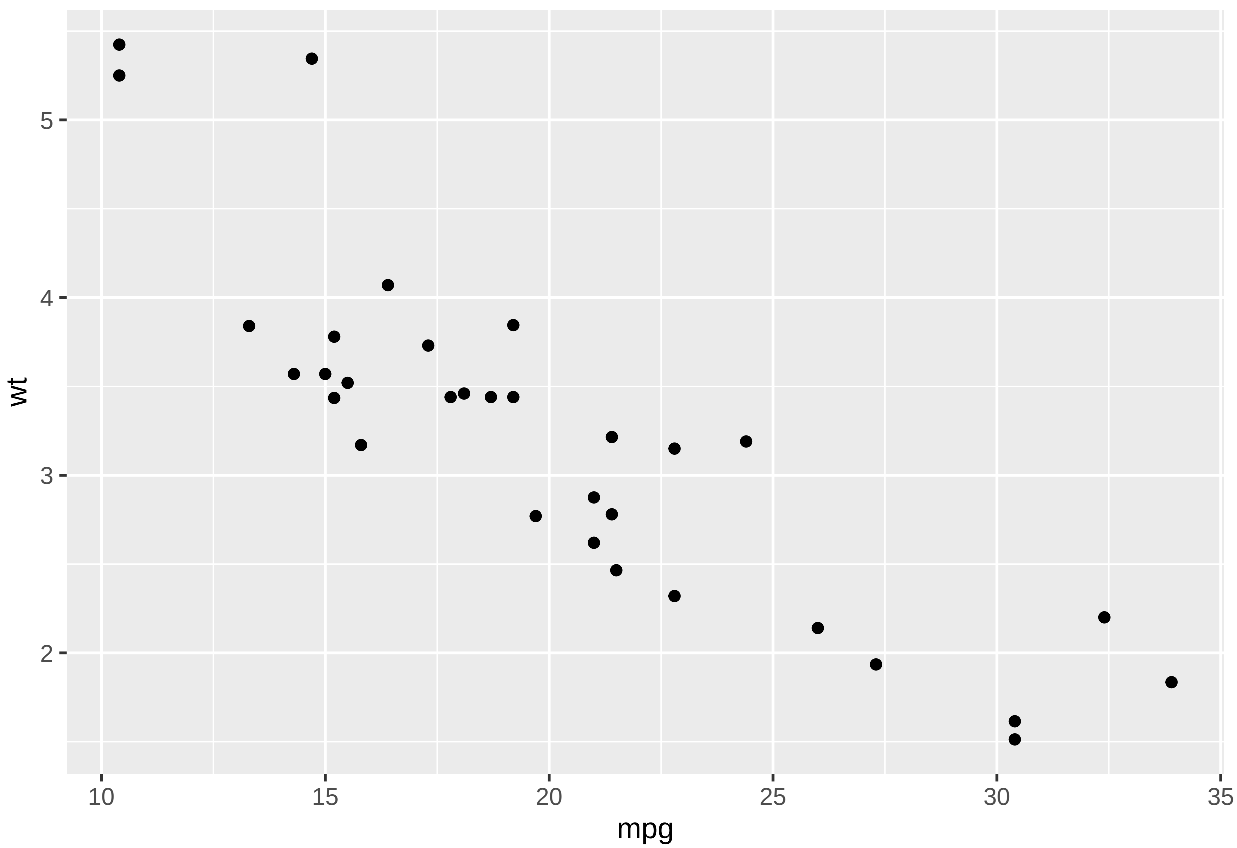
- The number of bins
- The fill color for the bars
- The color outlining the bars

```
histo <- function(data, x, bins = 30, fill_color = "blue", outline_color = "black") {
  x <- enquo(x)
  ggplot(data = data, mapping = aes(x = !!x)) +
    geom_histogram(bins = bins, fill = fill_color, color = outline_color)
}

# Test it with custom parameters
histo(pdxTrees, DBH, bins = 20, fill_color = "green", outline_color = "red")
```

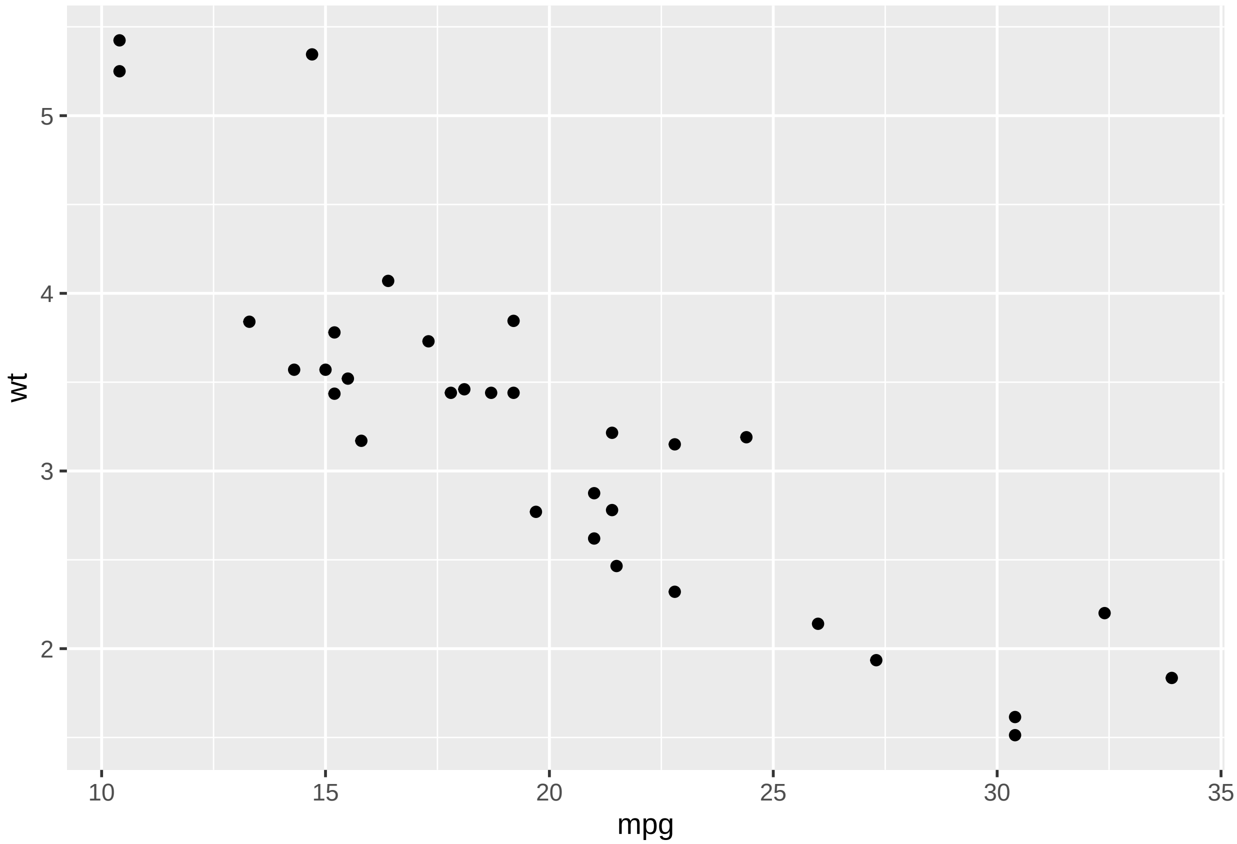


b. Write code to create a basic scatterplot with `ggplot2`. Then write and test a function to create a basic scatterplot.

```
library(ggplot2)


basic_scatterplot <- ggplot(data = mtcars, aes(x = mpg, y = wt)) +
  geom_point()


print(basic_scatterplot)
```



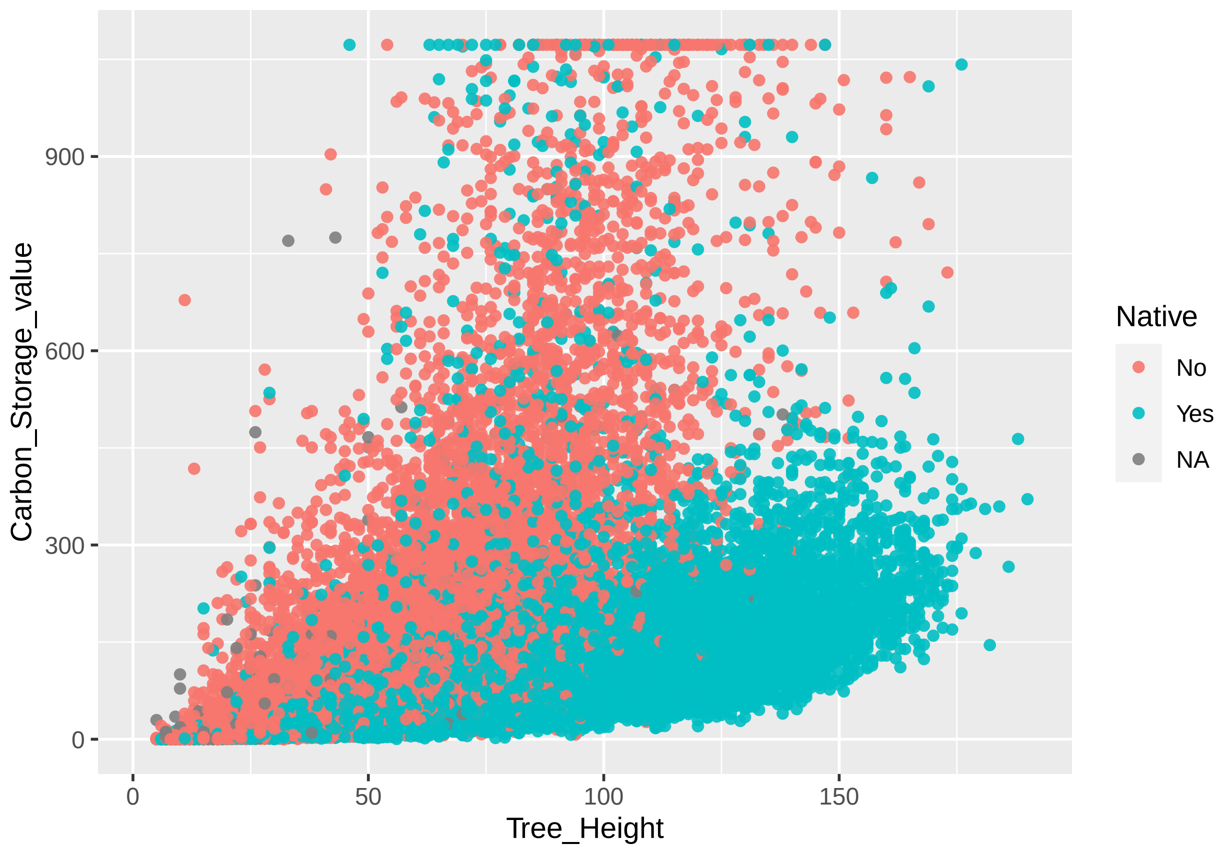

```
scatterplot_basic <- function(data, x, y) {
  ggplot(data = data, aes(x = {{ x }}, y = {{ y }})) +
    geom_point()
}


scatterplot_basic(mtcars, mpg, wt)
```



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

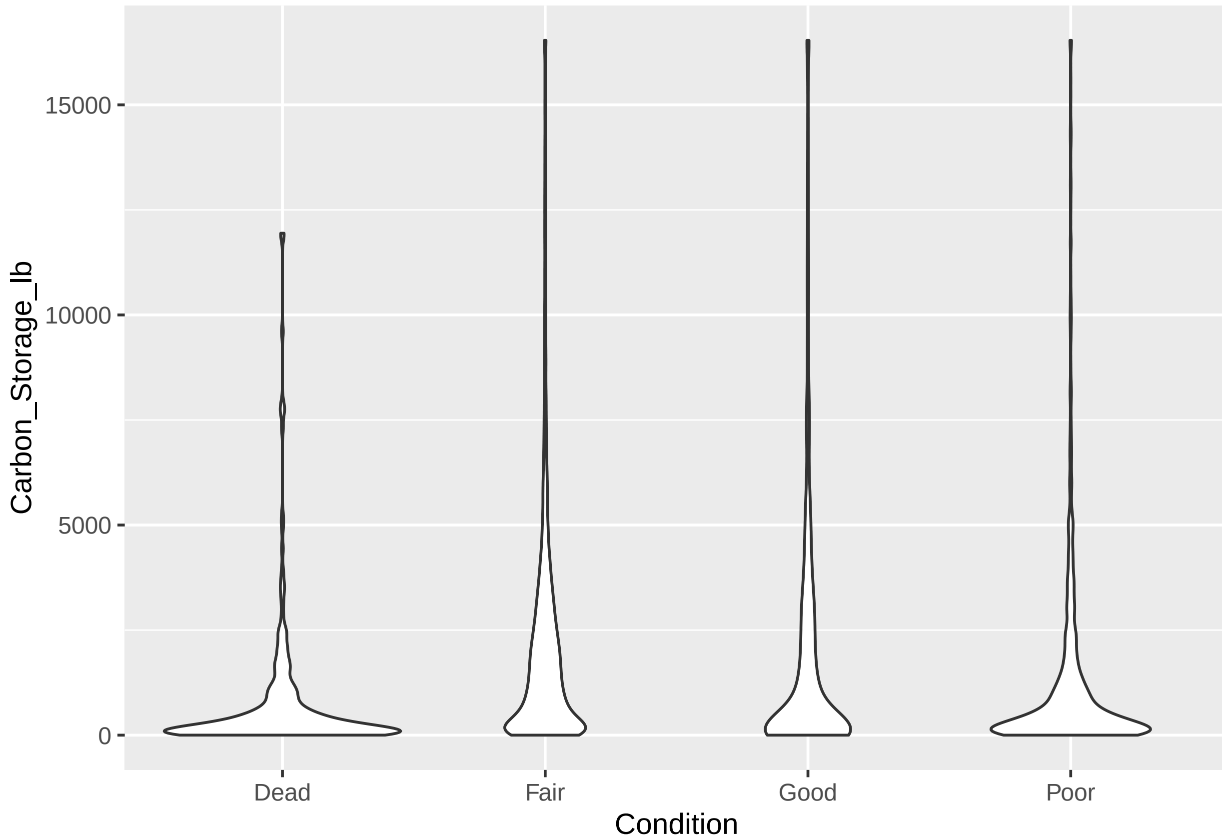
- Color the points by another variable.
- Set the transparency.

```
scatter_function <- function(data, x, y, alpha = 0.5, color = Condition){
   x <- enquo(x)
   y <- enquo(y)
   color <- enquo(color)
  ggplot(data = data, mapping = aes(x = !!x, y = !!y, color = !!color)) +
    geom_point(alpha = alpha)
}

# Testing scatter function
scatter_function(pdxTrees, Tree_Height, Carbon_Storage_value, alpha = 0.9, color = Native)
```

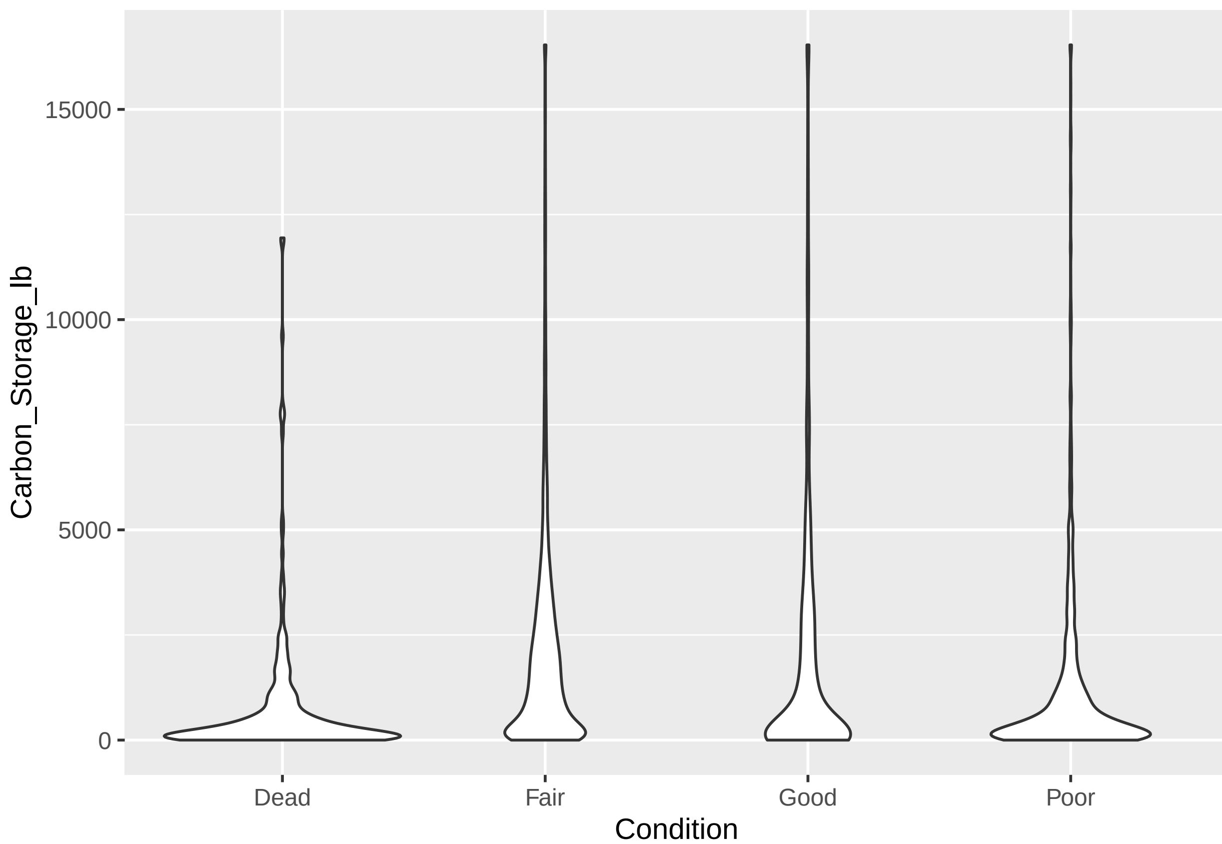


d. Write and test a function for your favorite `ggplot2` graph.

```
library(ggplot2)

#Basic violin plot code
ggplot(data = pdxTrees, aes(x = Condition, y = Carbon_Storage_lb)) +
  geom_violin()
```




```
#Function for basic violin plot
violin_function <- function(data, x, y){
   x <- enquo(x)
   y <- enquo(y)
  ggplot(data = data, mapping = aes(x = !!x, y = !!y)) +
    geom_violin()
}

#Testing violin function
violin_function(pdxTrees, Condition, Carbon_Storage_lb)
```



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()
```

```
## # 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
```

```r
conditional_proportions_table <- function(data, var1, var2) {
  data %>%
    count(!!enquo(var1), !!enquo(var2)) %>%
    group_by(!!enquo(var1)) %>%
    mutate(prop = n/sum(n)) %>%
    ungroup()
}

result <- conditional_proportions_table(pdxTrees, Native, Condition)
print(result)
```

```
## # 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.

```r
library(dplyr)

quant_summary_by_category <- function(data, category_var, quant_var) {
  data %>%
    group_by(!!enquo(category_var)) %>%
    summarize(
      mean = mean(!!enquo(quant_var), na.rm = TRUE),
      median = median(!!enquo(quant_var), na.rm = TRUE),
      sd = sd(!!enquo(quant_var), na.rm = TRUE),
      min = min(!!enquo(quant_var), na.rm = TRUE),
      max = max(!!enquo(quant_var), na.rm = TRUE),
      n = n(),
      missing_values = sum(is.na(!!enquo(quant_var)))
    ) %>%
    ungroup()
}


example_data <- data.frame(
  Category = rep(c("A", "B", "C"), each = 5),
  Value = c(1, 2, 3, 4, NA, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)
)
```

```r
result <- quant_summary_by_category(example_data, Category, Value)
print(result)
```

```
## # A tibble: 3 x 8
##   Category  mean median    sd   min   max     n missing_values
##   <chr>    <dbl>  <dbl> <dbl> <dbl> <dbl> <int>          <int>
## 1 A          2.5    2.5  1.29     1     4     5              1
## 2 B          7      7    1.58     5     9     5              0
## 3 C         12     12    1.58    10    14     5              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**.

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

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

  return(result)
}

# Test with Ezekiel and 1883
result_1883 <- grab_name("Ezekiel", 1883)
print(result_1883)
```

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

```r
# Test with Ezekiel and 1983
result_1983 <- grab_name("Ezekiel", 1983)
print(result_1983)
```

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