Lab 3

Ashwin Dev

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"
```

dats: list
pdxTreesSmall: data frame, tibble
ht: 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.

```
library(dplyr)

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

---

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

```
species_vector <- as.character(pdxTrees$Species)
```

---

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

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

```
## [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.)

```
douglas_fir_4th <- pdxTreesSmall %>%
  filter(Common_Name == "Douglas-Fir") %>%
  slice(4) %>%
  select(DBH, Condition)

douglas_fir_4th
```

```
## # 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.

```
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
```

New function

```r
calculate_confidence_interval <- function(data, confidence_level = 0.95) {
  if(!is.numeric(data)) {
    stop("Data should be numeric.")
  }

  if(any(is.na(data))) {
    warning("Missing values found in the data. Excluding missing values from the calculations.")
    data <- na.omit(data)
  }

  if(confidence_level <= 0 || confidence_level >= 1) {
    stop("Confidence level should be greater than 0 and less than 1.")
  }

  n <- length(data)
  mean_data <- mean(data)
  std_error <- sd(data) / sqrt(n)
  t_critical <- qt(p = 1 - (1 - confidence_level) / 2, df = n - 1)
  lower_bound <- mean_data - t_critical * std_error
  upper_bound <- mean_data + t_critical * std_error

  return(list(
    lower_bound = lower_bound,
    upper_bound = upper_bound,
    mean = mean_data,
    confidence_level = confidence_level
  ))
}

#test
ci_result <- calculate_confidence_interval(pdxTrees$DBH, confidence_level = 0.90)
print(ci_result)
```

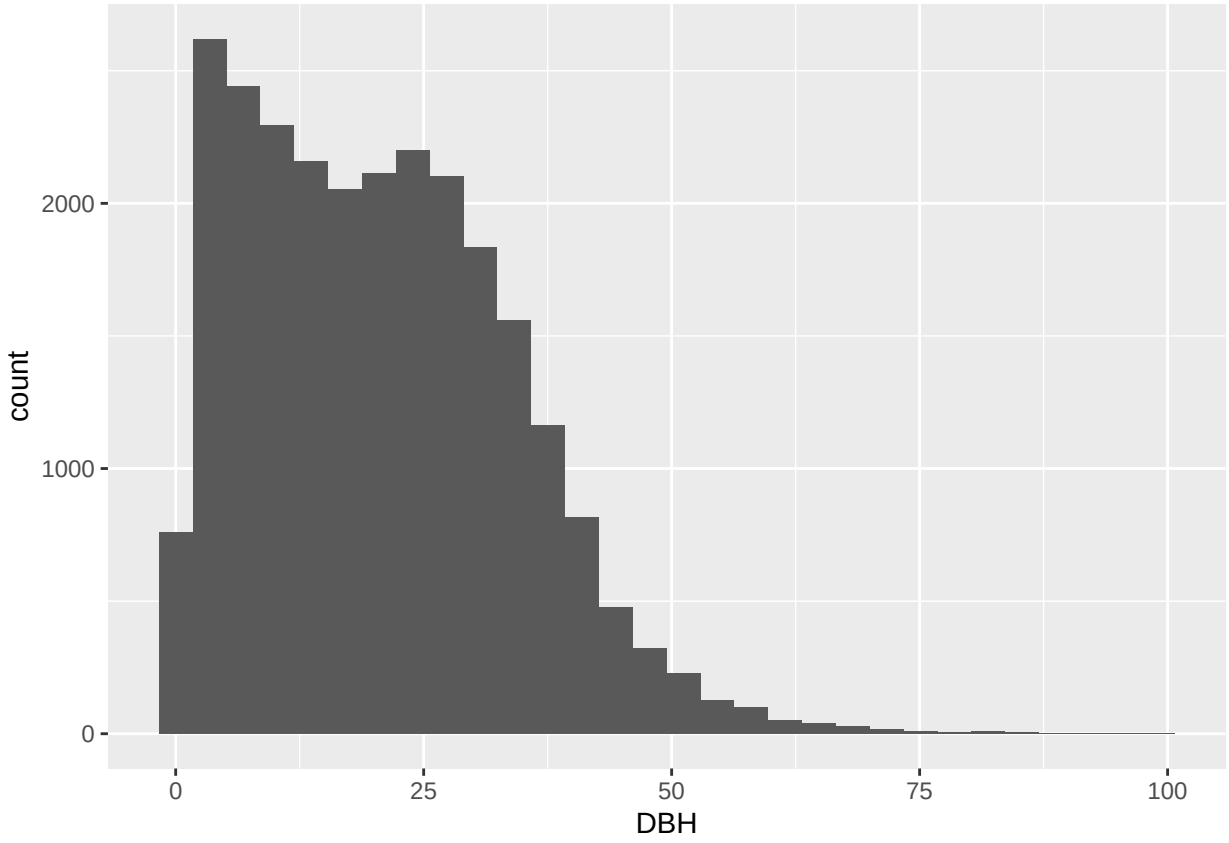
```
## $lower_bound
## [1] 20.47622
##
## $upper_bound
## [1] 20.75194
##
## $mean
## [1] 20.61408
##
## $confidence_level
## [1] 0.9
```

Problem 3: Wrapper Function for your `ggplot`

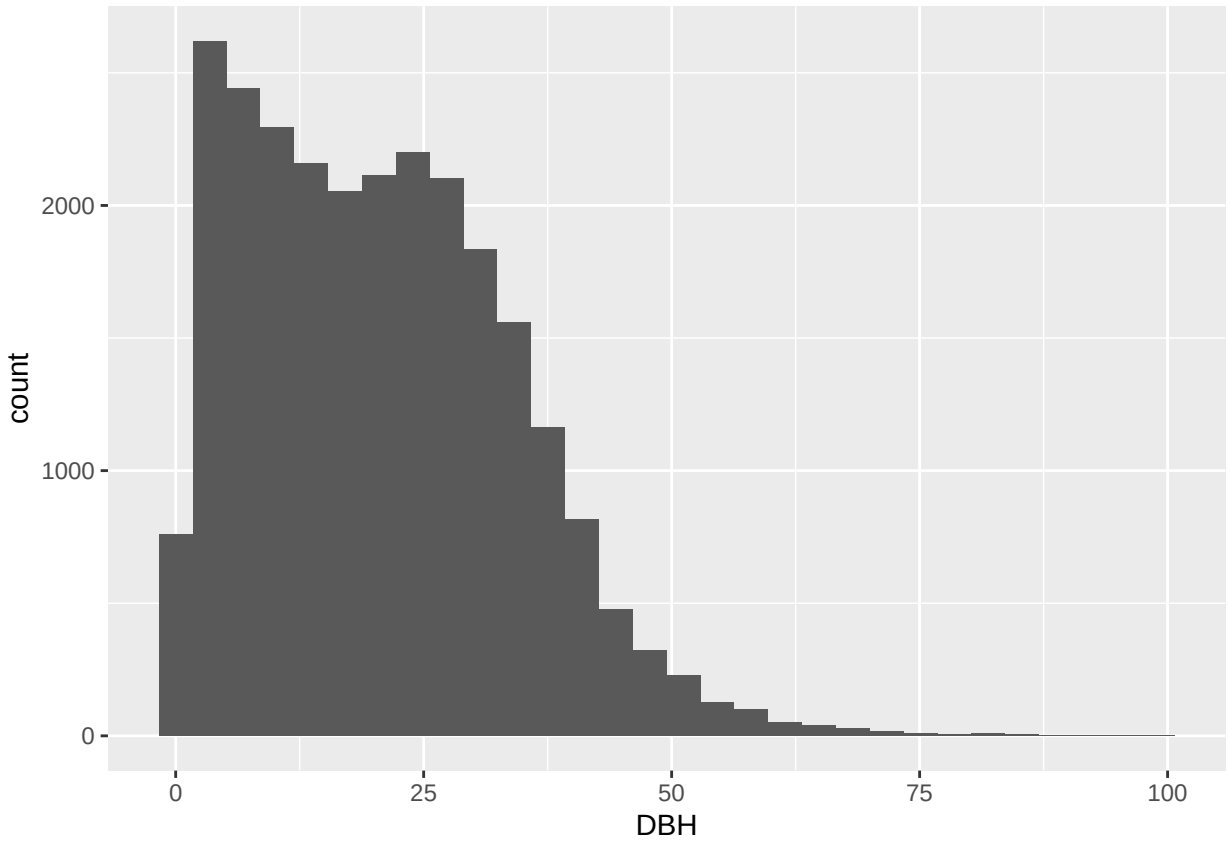
While we (i.e. Math 241 students) all love the grammar of graphics, not everyone else does. So for this problem, we are going to practice creating wrapper functions for `ggplot2`.

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()
```



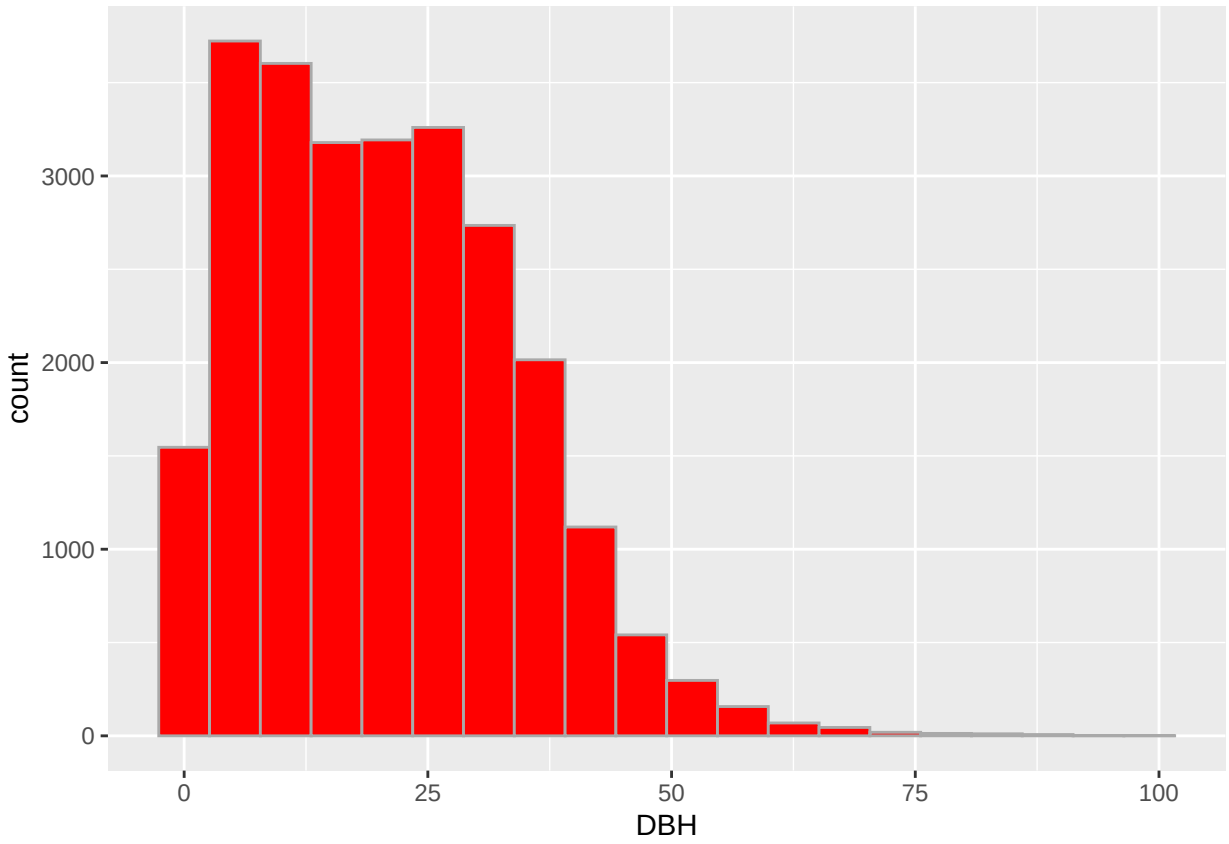

```
# Shorthand histogram function
histo <- function(data, x){
   x <- enquo(x)
  ggplot(data = data, mapping = aes(x = !!x)) +
    geom_histogram()
}

# 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

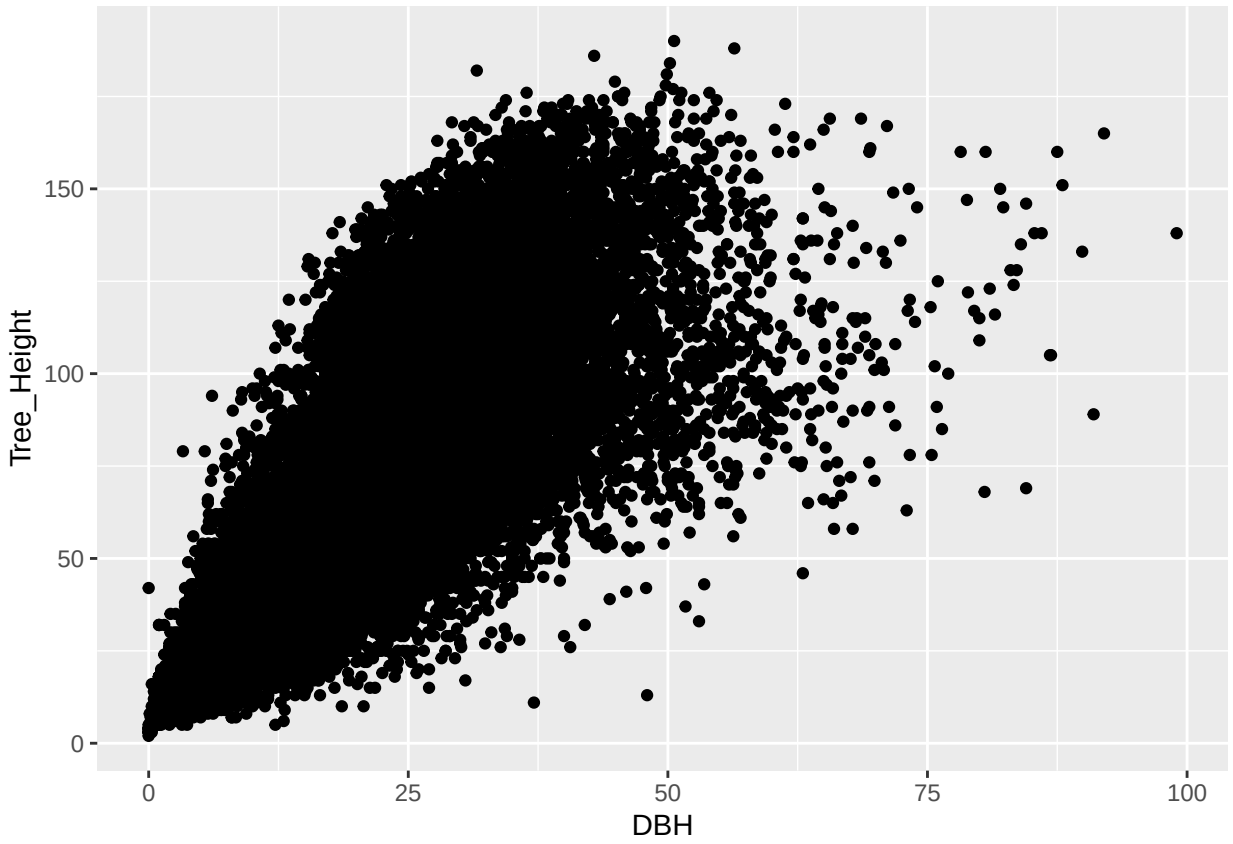
```
library(ggplot2)

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
histo(pdxTrees, DBH, bins = 20, fill_color = "red", outline_color = "darkgray")
```



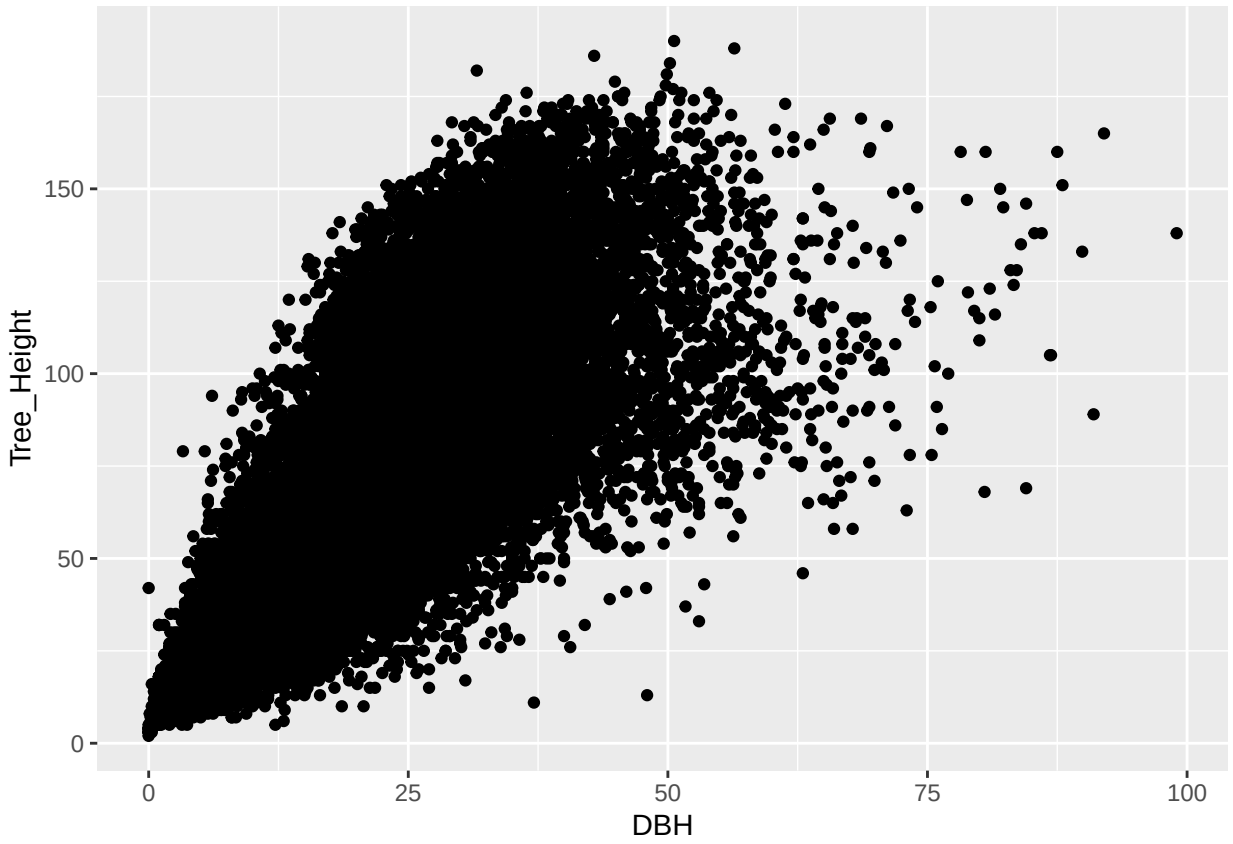
---

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

```
ggplot(data = pdxTrees, aes(x = DBH, y = Tree_Height)) +
  geom_point()
```




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

#test
scatter_plot(pdxTrees, DBH, Tree_Height)
```

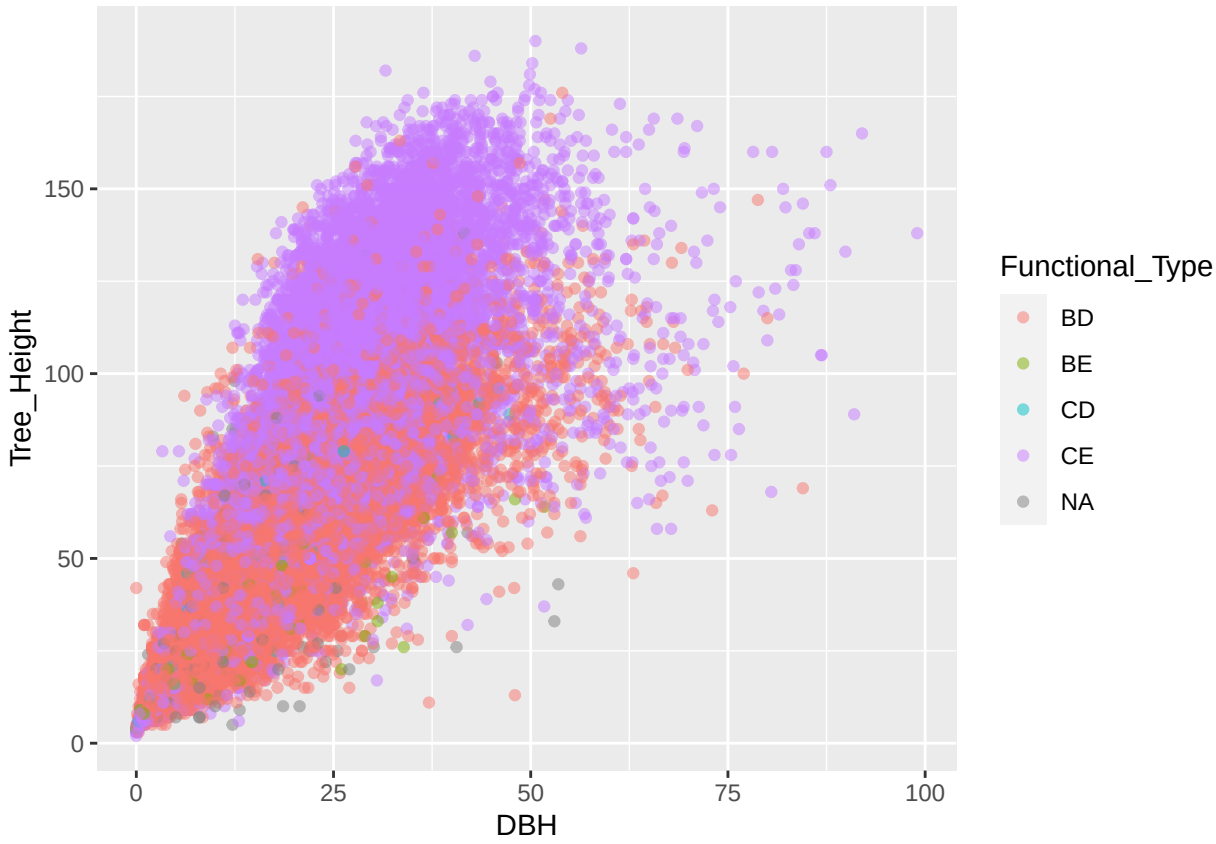


---

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

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

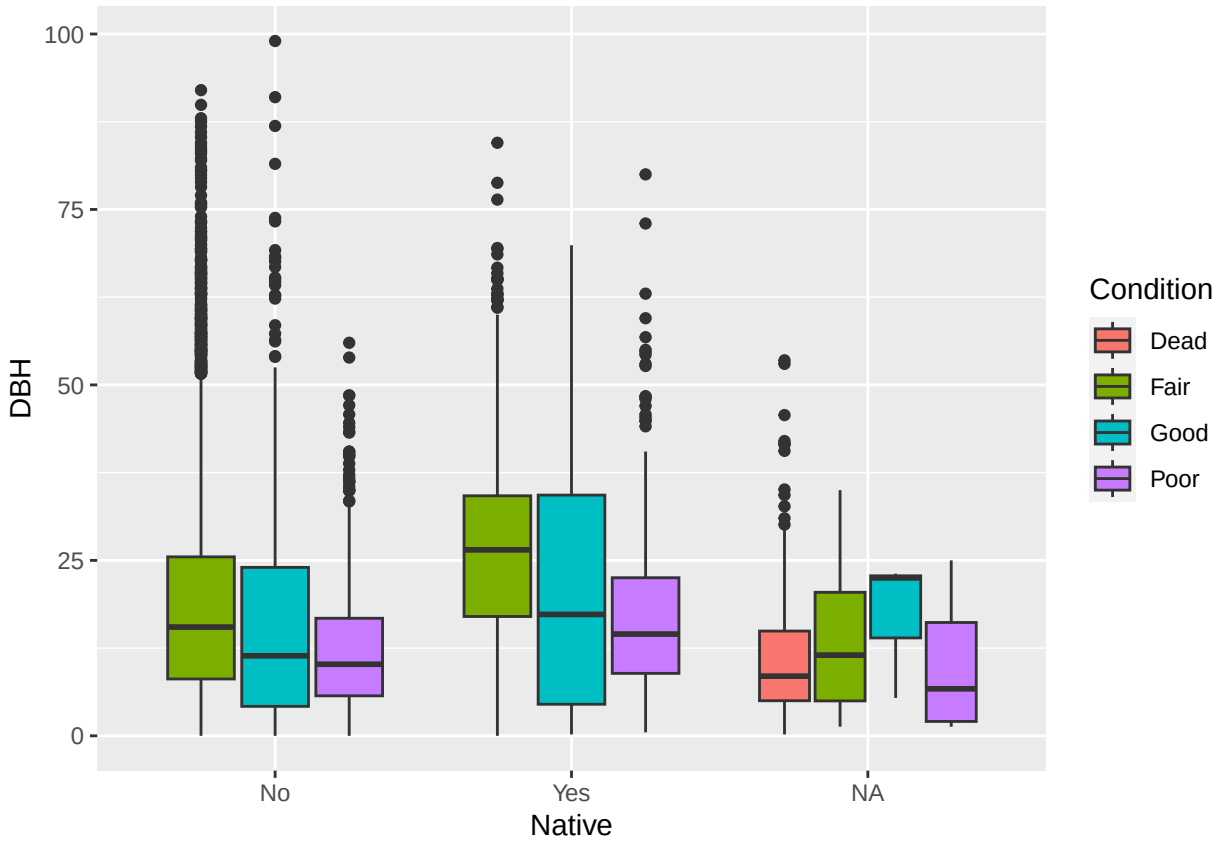
```
scatter_plot <- function(data, x_var, y_var, color_var = NULL, transparency = 1){
  plot <- ggplot(data, aes_string(x = x_var, y = y_var)) +
    geom_point(aes_string(color = color_var), alpha = transparency)

  if(is.null(color_var)){
    plot <- plot + geom_point(alpha = transparency)
  }

  plot
}

#test
scatter_plot(pdxTrees, "DBH", "Tree_Height", color_var = "Functional_Type", transparency = 0.5)
```



---

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

```
boxplot_gg <- function(data, x_var, y_var, fill_var = NULL) {
  plot <- ggplot(data, aes_string(x = x_var, y = y_var, fill = fill_var)) +
    geom_boxplot()

  if(is.null(fill_var)) {
    plot <- plot + geom_boxplot()
  }

  plot
}

#test
boxplot_gg(pdxTrees, "Native", "DBH", fill_var = "Condition")
```



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
```

```
conditional_proportions <- function(data, group_var, prop_var){
  # quote the variable names
  group_var <- enquo(group_var)
  prop_var <- enquo(prop_var)

  # create conditional proportions table
  data %>%
    count(!!group_var, !!prop_var) %>%
    group_by(!!group_var) %>%
    mutate(prop = n/sum(n)) %>%
    ungroup()
}

# test
conditional_proportions_table <- conditional_proportions(pdxTrees, Native, Condition)
print(conditional_proportions_table)
```

```
## # 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.

```
summarize_by_group <- function(data, group_var, quant_var){
  group_var_enquo <- enquo(group_var)
  quant_var_enquo <- enquo(quant_var)

  summary_df <- data %>%
    group_by(!!group_var_enquo) %>%
    summarise(
      Mean = mean(!!quant_var_enquo, na.rm = TRUE),
```

```
      Median = median(!!quant_var_enquo, na.rm = TRUE),
      SD = sd(!!quant_var_enquo, na.rm = TRUE),
      Min = min(!!quant_var_enquo, na.rm = TRUE),
      Max = max(!!quant_var_enquo, na.rm = TRUE),
      Sample_Size = n(),
      Missing = sum(is.na(!!quant_var_enquo))
    ) %>%
    ungroup() # Ensure the grouping is removed after summarisation

  return(summary_df)
}

# test
summary_stats <- summarize_by_group(pdxTrees, Native, DBH)
print(summary_stats)
```

```
## # A tibble: 3 x 8
##   Native  Mean Median    SD   Min   Max Sample_Size Missing
##   <chr>  <dbl>  <dbl> <dbl> <dbl> <dbl>       <int>   <int>
## 1 No      17.6   14.8 12.9    0    99         14202       0
## 2 Yes     24.9   25.8 12.9    0    84.5       10931       0
## 3 <NA>    11.7    9.1  9.06   0.2  53.5         401       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) {
  data("babynames")

  filtered_data <- babynames %>%
    dplyr::filter(name == myname, year == myyear)

  # check if the  dataset is empty
  if(nrow(filtered_data) == 0) {
    stop("No matches found for the name and year combination.")
  }

  return(filtered_data)
}

# Run the function with the specified arguments
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
```

```
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
```