Lab 3

Kiana Fields

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`?

dats os a list of data.frames and a tibble. pdxTreesSmall is a dataset. ht is a num value.

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

```
view(ht)
```

The 10th value is 112, the 11th is 112, and the 12th is 48.

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

```
Species_df <- as.data.frame(pdxTrees$Species)
```

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

```
Species_df <- as.vector(pdxTrees$Species)
```

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

```
sets_2 <- dats$sets[[2]]
print(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.)

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

dougfir_4 <- dougfir_pdx[4, c("DBH", "Condition")]
print(dougfir_4)
```

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

print(thing1)
```

```
## [1] 25534
```

```r
library(pdxTrees)

calculateCI <- function(data, variable, conf_lvl) {

  if (is.null(data) || is.null(variable) || is.null(conf_lvl)) {
    stop("Please specify a dataset, a variable, AND a confidence level")
    }

  num_obs <- length(data[[variable]])
  average <- mean(data[[variable]])
  sd_se <- sd(data[[variable]])/sqrt(num_obs)
  quartile <- qt(p=conf_lvl, df=num_obs - 1)
  lower_ci <- average - quartile * sd_se
  upper_ci <- average + quartile * sd_se

  result <- list(
    num_obs = num_obs,
    average = average,
    sd_se = sd_se,
    quartile = quartile,
    lower_ci = lower_ci,
    upper_ci = upper_ci
  )

  return(result)
}

calculateCI(data = pdxTrees, variable = "DBH", conf_lvl = .975)
```

```
## $num_obs
## [1] 25534
##
## $average
## [1] 20.61408
##
## $sd_se
## [1] 0.08380945
##
## $quartile
## [1] 1.960057
```

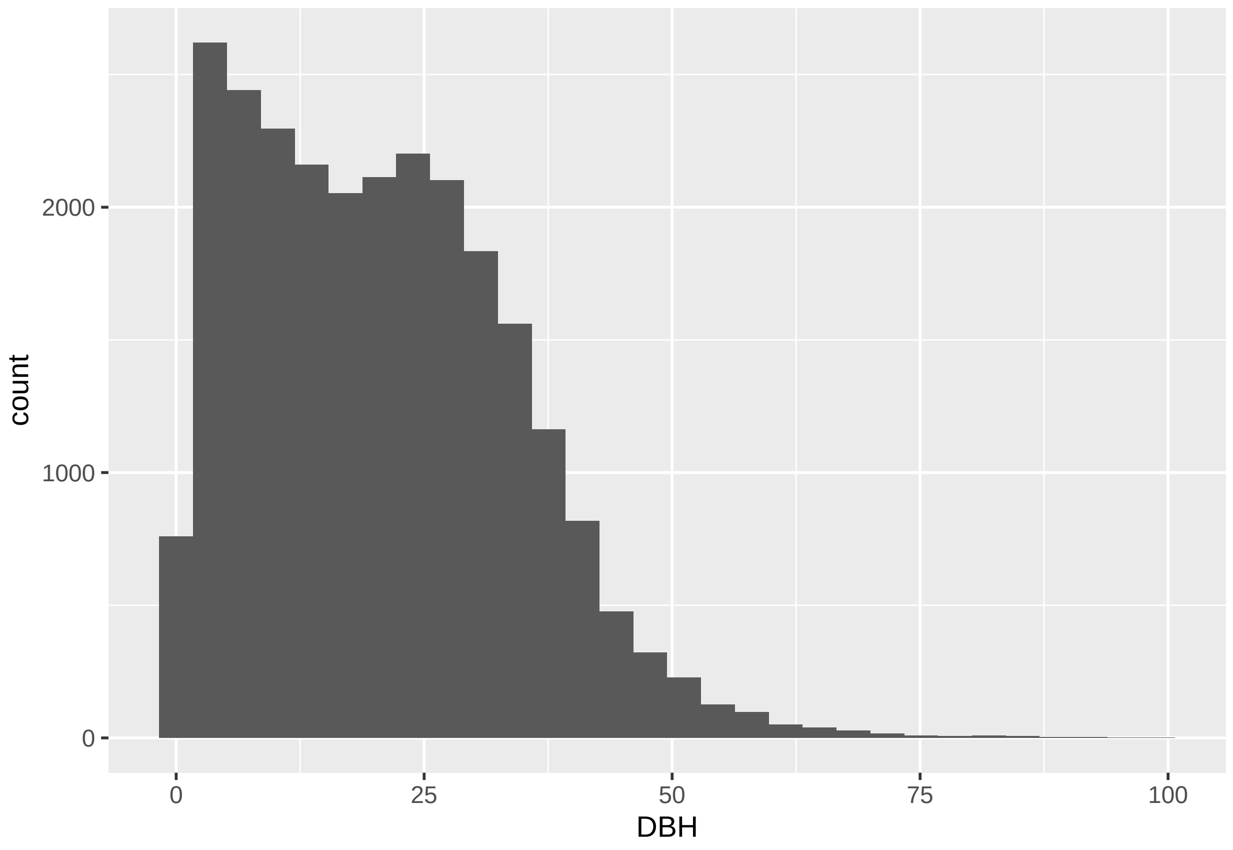
```
##
## $lower_ci
## [1] 20.44981
##
## $upper_ci
## [1] 20.77835
```

Problem 3: Wrapper Function for your ggplot

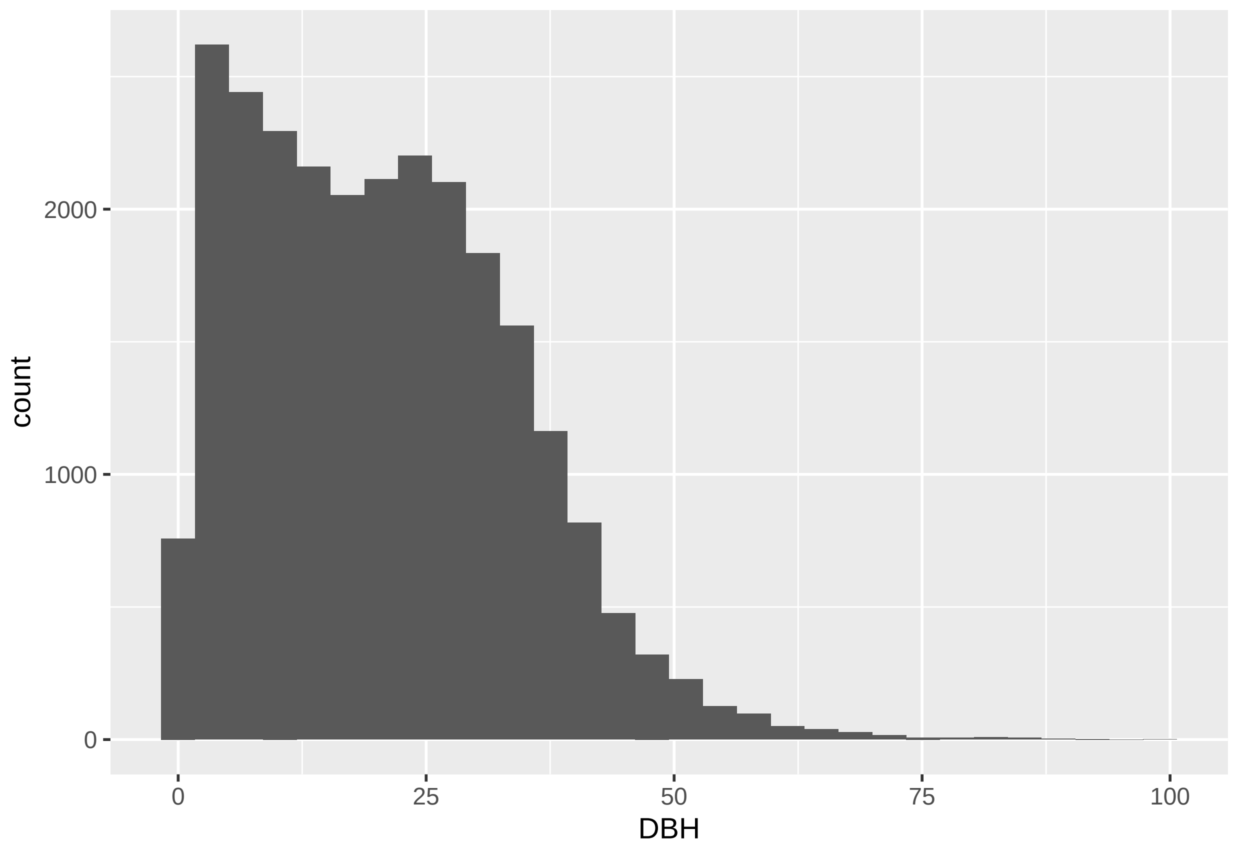
While we (i.e. Math 241 students) all love the grammar of graphics, not everyone else does. So for this problem, we are going to practice creating wrapper functions for `ggplot2`.

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.)

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



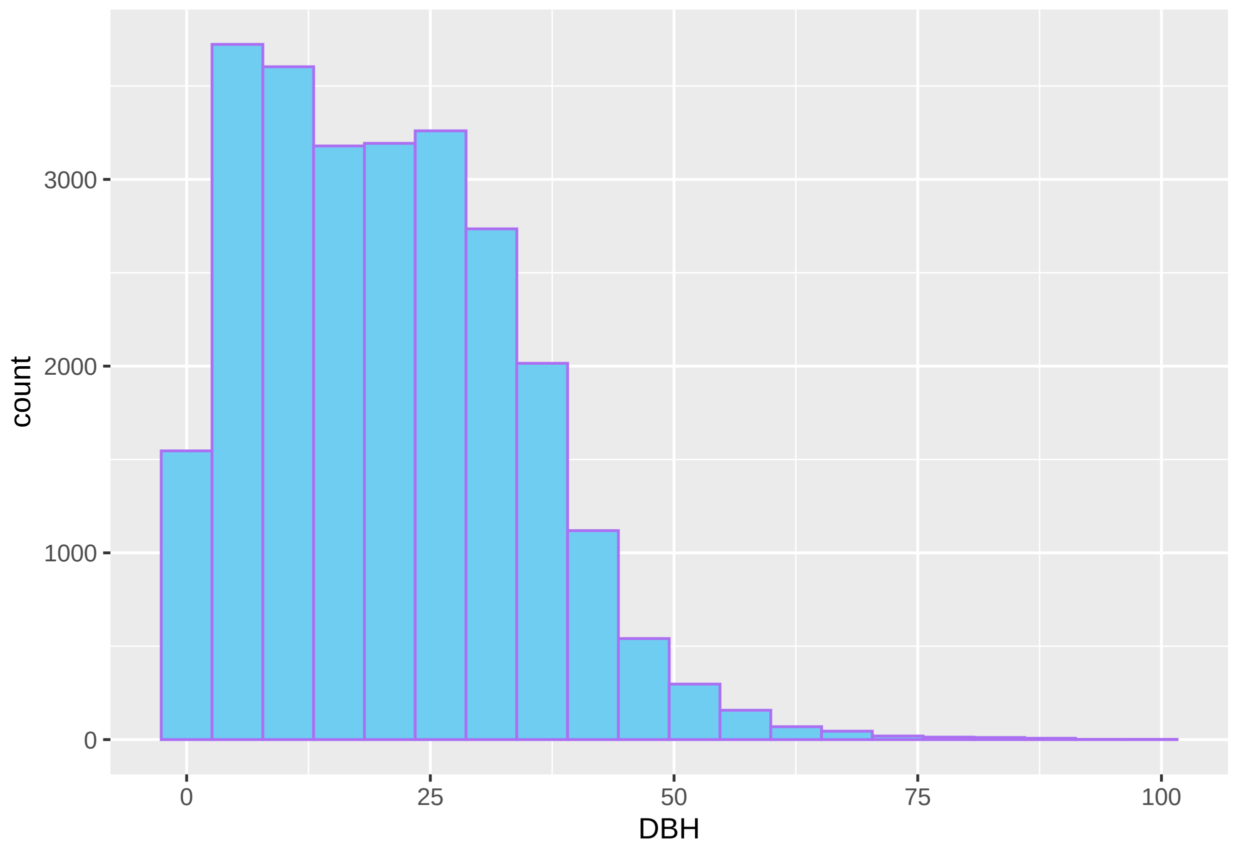
```r
# Shorthand histogram function
histo <- function(data, x){
   x <- enquo(x)
  ggplot(data = data, mapping = aes(x = !!x)) +
    geom_histogram()
}

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



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

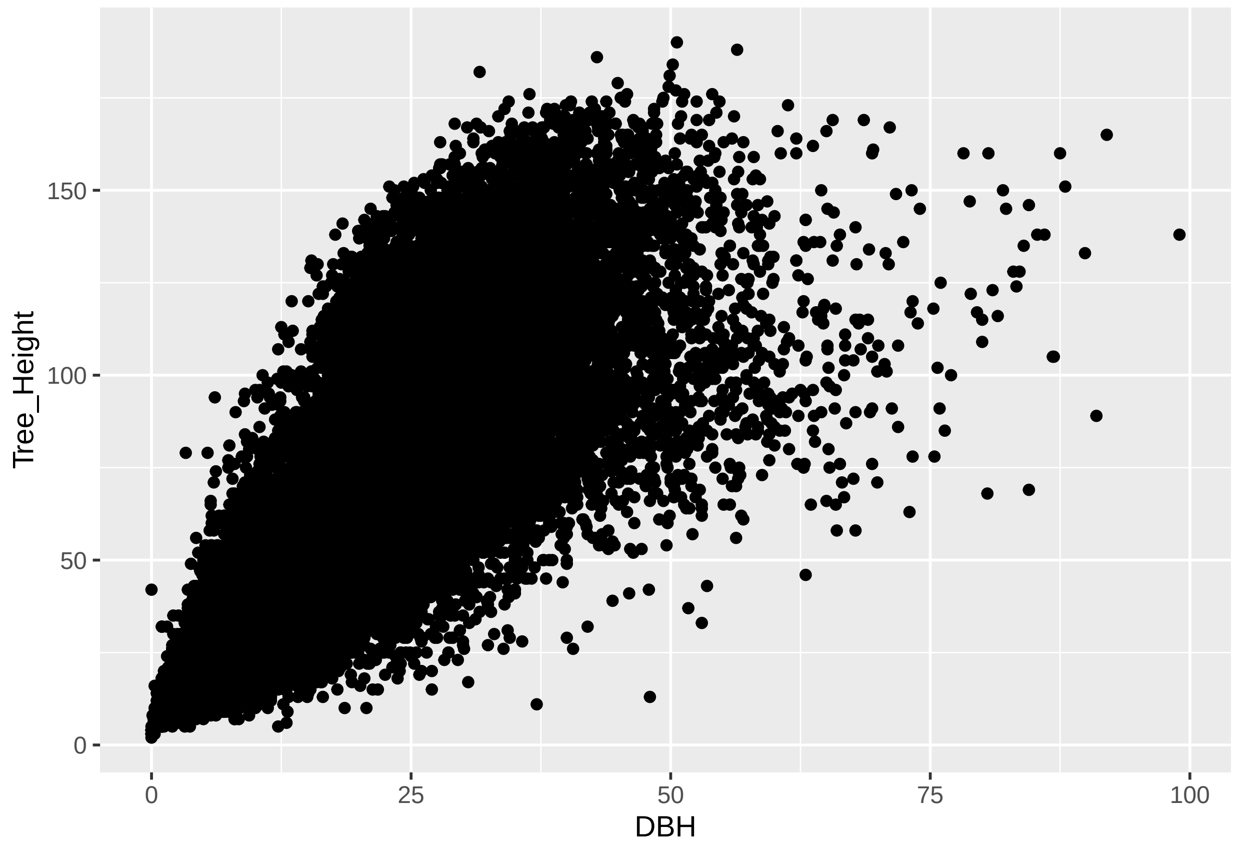
- The number of bins
- The fill color for the bars
- The color outlining the bars

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

# Test it
histo(pdxTrees, DBH, 20, "#70CDF2", "#AB70F2")
```

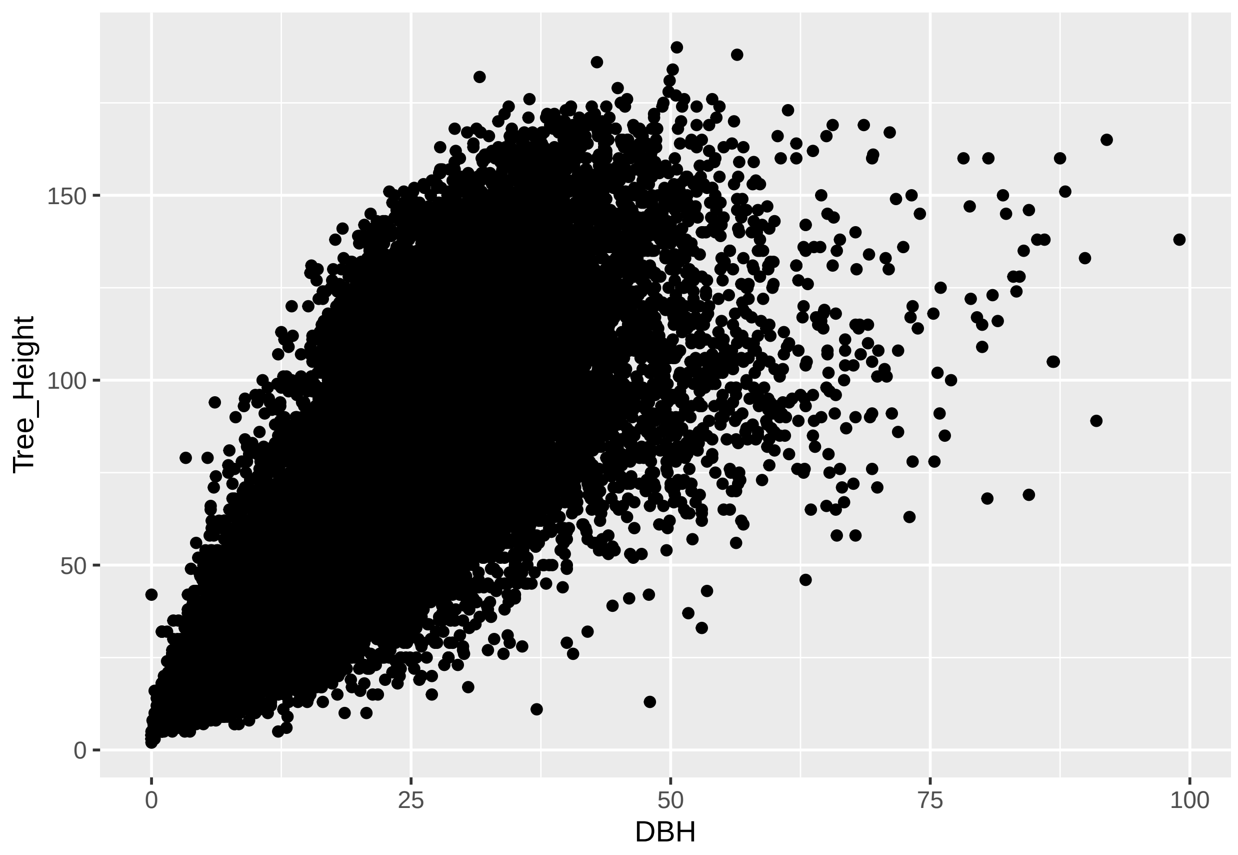


b. Write code to create a basic scatterplot with `ggplot2`. Then write and test a function to create a basic scatterplot.

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



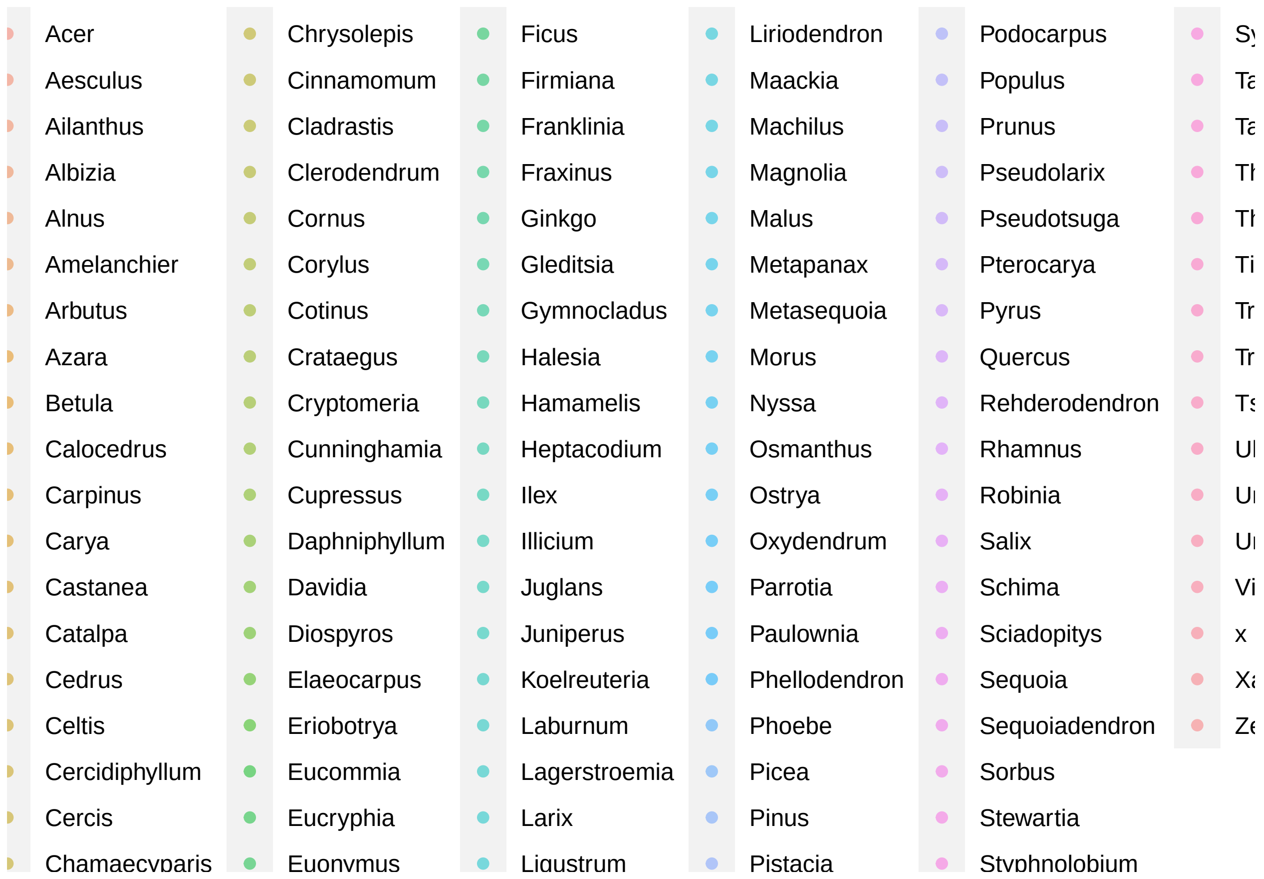
```
scatter <- function(data, x, y){
   x <- enquo(x)
   y <- enquo(y)
  ggplot(data = data, mapping = aes(x = !!x, y = !!y)) +
    geom_point()
}

# Test it
scatter(pdxTrees, DBH, Tree_Height)
```



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

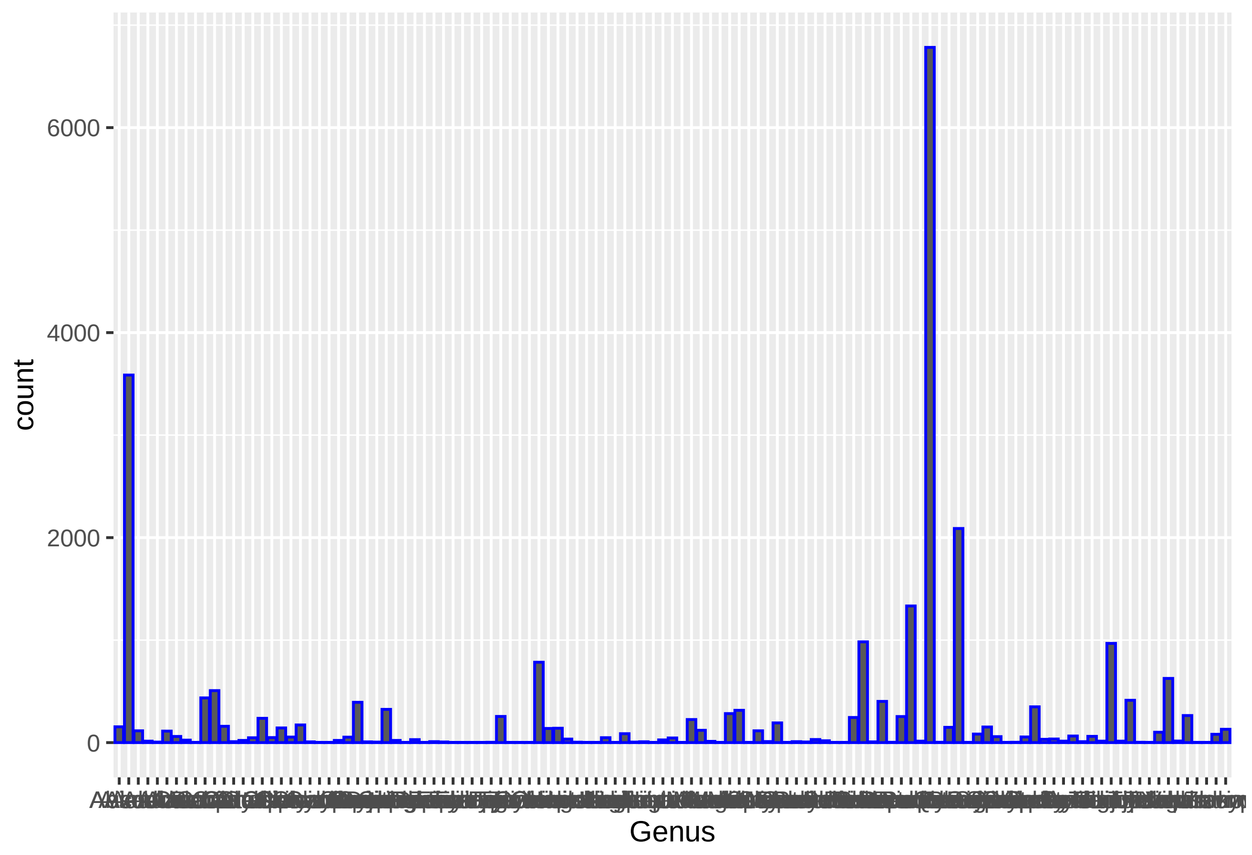
- Color the points by another variable.
- Set the transparency.

```r
scatter <- function(data, x, y, color, transparent){
   x <- enquo(x)
   y <- enquo(y)
  ggplot(data = data, mapping = aes(x = !!x, y = !!y)) +
    geom_point(aes_string(color = color), alpha=transparent)
}

# Test it
scatter(pdxTrees, DBH, Tree_Height, "Genus", 0.5)
```



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

```r
bars <- function(data, x, color){
  x <- enquo(x)
 ggplot(data = data, mapping = aes(x = !!x)) +
   geom_bar(color = color)
}

# Test it
bars(pdxTrees, Genus, "blue")
```



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
cond_prop <- function(data, var_a, var_b){
  var_a <- enquo(var_a)
  var_b <- enquo(var_b)
  result <- data %>%
    count(!!var_a, !!var_b) %>%
    group_by(!!var_a) %>%
    mutate(prop = n/sum(n)) %>%
    ungroup()
  print(result)
}

cond_prop(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.

```r
calculation <- function(data, var_a, var_b) {
  var_a <- enquo(var_a)
  var_b <- enquo(var_b)

  result <- data %>%
    group_by(!!var_a) %>%
    summarise(
      mean = mean(!!var_b, na.rm = TRUE),
      median = median(!!var_b, na.rm = TRUE),
      sd = sd(!!var_b, na.rm = TRUE),
      min = min(!!var_b, na.rm = TRUE),
      max = max(!!var_b, na.rm = TRUE),
      n = n(),
      n_missing = sum(is.na(!!var_b))
    ) %>%
    ungroup()

  print(result)
}


calculation(pdxTrees, Condition, DBH)
```

```
## # A tibble: 4 x 8
##   Condition  mean median    sd   min   max     n n_missing
##   <chr>     <dbl>  <dbl> <dbl> <dbl> <dbl> <int>     <int>
## 1 Dead       11.2    8.5  9.22   0.2  53.5   264         0
## 2 Fair       21.4   20.6 13.2    0    92   22279         0
## 3 Good       17.5   12.4 15.5    0    99    1646         0
## 4 Poor       13.9   11.4 10.4    0    80    1345         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
#' Make sure to switch eval = FALSE to eval = TRUE before knitting!!
library(babynames)
data("babynames")

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

  if(nrow(result) == 0) {
    stop("No rows match that combination")
  } 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
```

```r
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
```