# 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(tidyverse)
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`?

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 <- pdxTrees %>%
  select(Species)
```

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

```
vector <- pdxTrees$Species
```

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.)

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

pdxTreeNew$DBH[4]
```

```
## [1] 32.1
```

```
pdxTreeNew$Condition[4]
```

```
## [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)

create_confidence_interval <- function(x, alpha = 0.05){
  if (!is.numeric(x)){
    stop('Unfortunately this function only works for numeric input.\n',
         'You have provided an object of class:', class(x))
  }

  if (!is.numeric(alpha)){
    stop('Unfortunately this function only works for numeric input.\n',
         'You have provided an object of class:', class(x))
  }

  x<- x[!is.na(x)]

  length <- length(x)
  mean <- mean(x)
  se <- sd(x)/sqrt(length)
  t_star <- qt(p = 1-(alpha/2), df = length - 1)
  lower_interval <- mean - t_star*se
  upper_interval <- mean + t_star*se

  return(c(lower_interval,upper_interval))
}

create_confidence_interval(pdxTrees$DBH, alpha = 0.01)
```

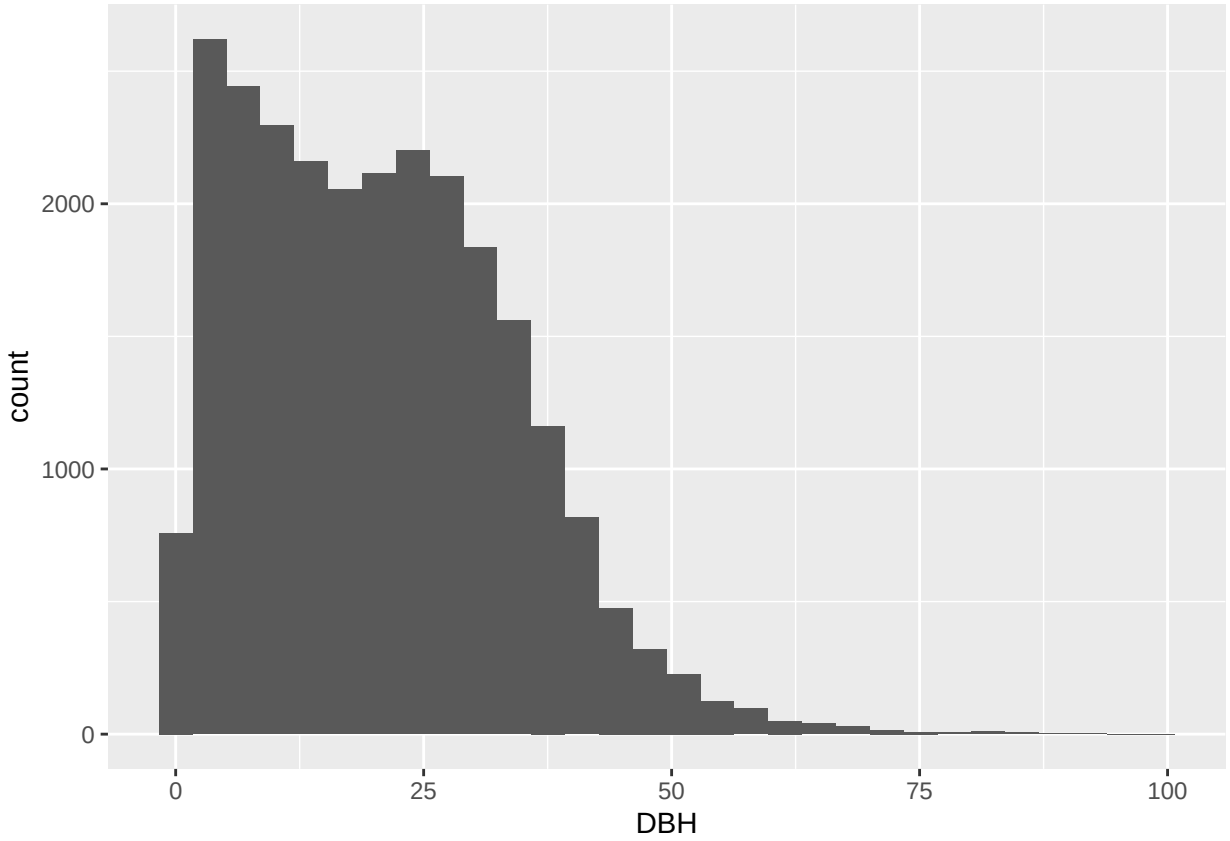
```
## [1] 20.39819 20.82998
```

### Problem 3: Wrapper Function for your `ggplot`

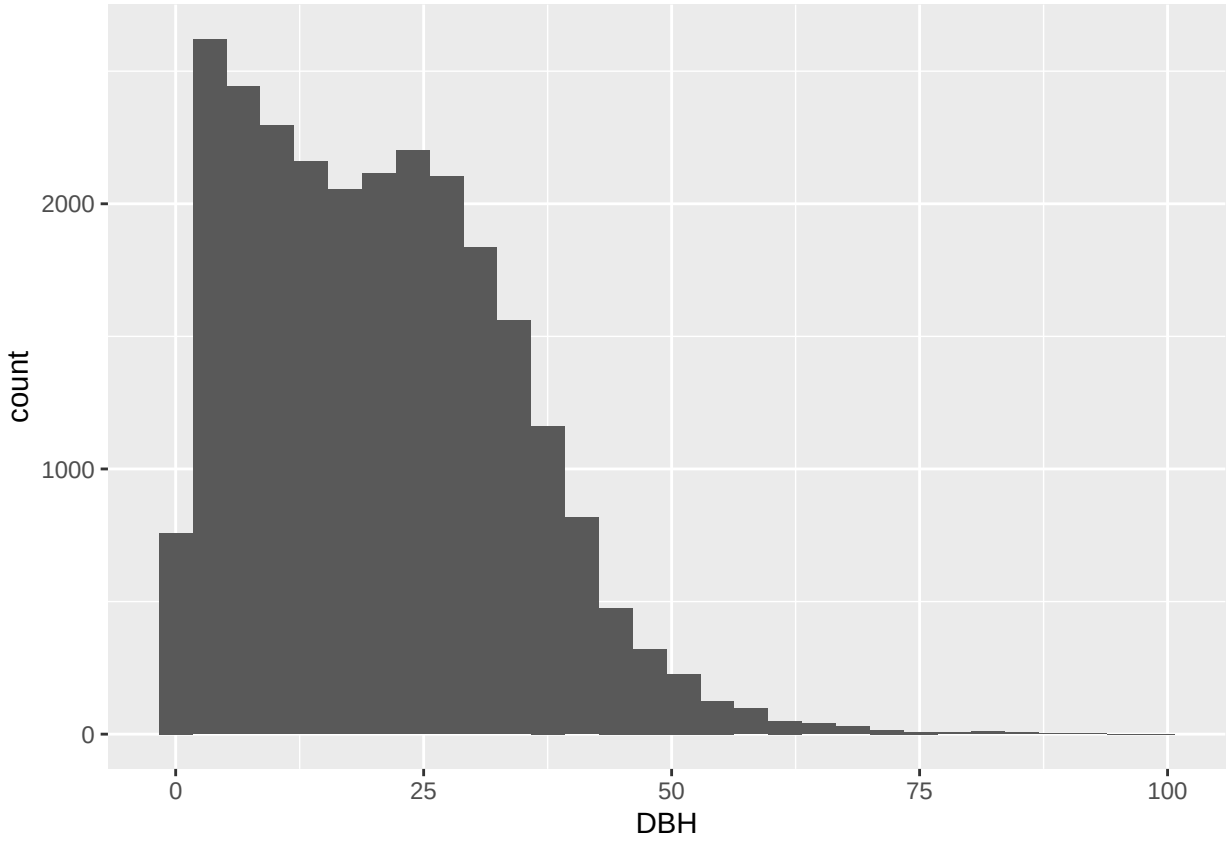
While we (i.e. Math 241 students) all love the grammar of graphics, not everyone else does. So for this problem, we are going to practice creating wrapper functions for `ggplot2`.

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()
```



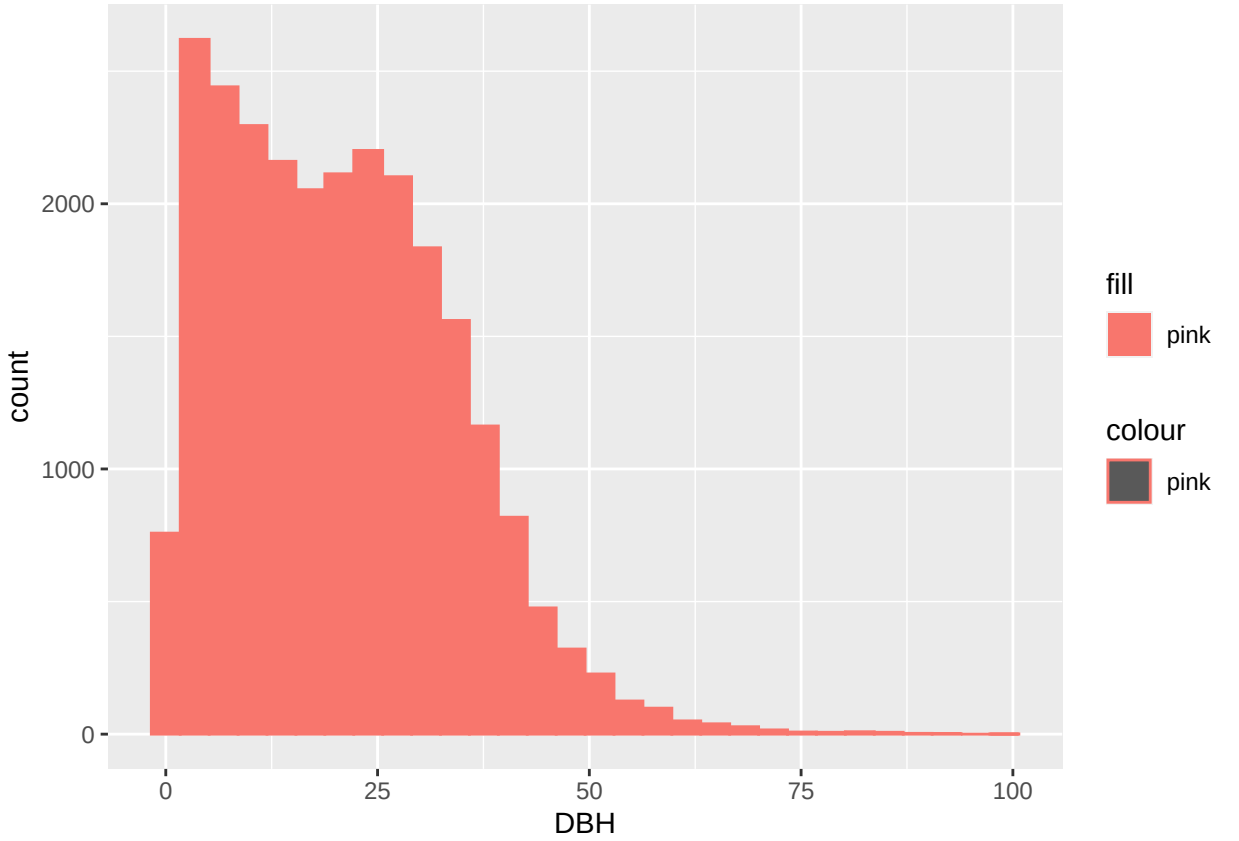

```
# Shorthand histogram function
histo <- function(data, x){
  x <- enquo(x)
  ggplot(data = data, mapping = aes(x = !!x)) +
    geom_histogram()
}

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




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

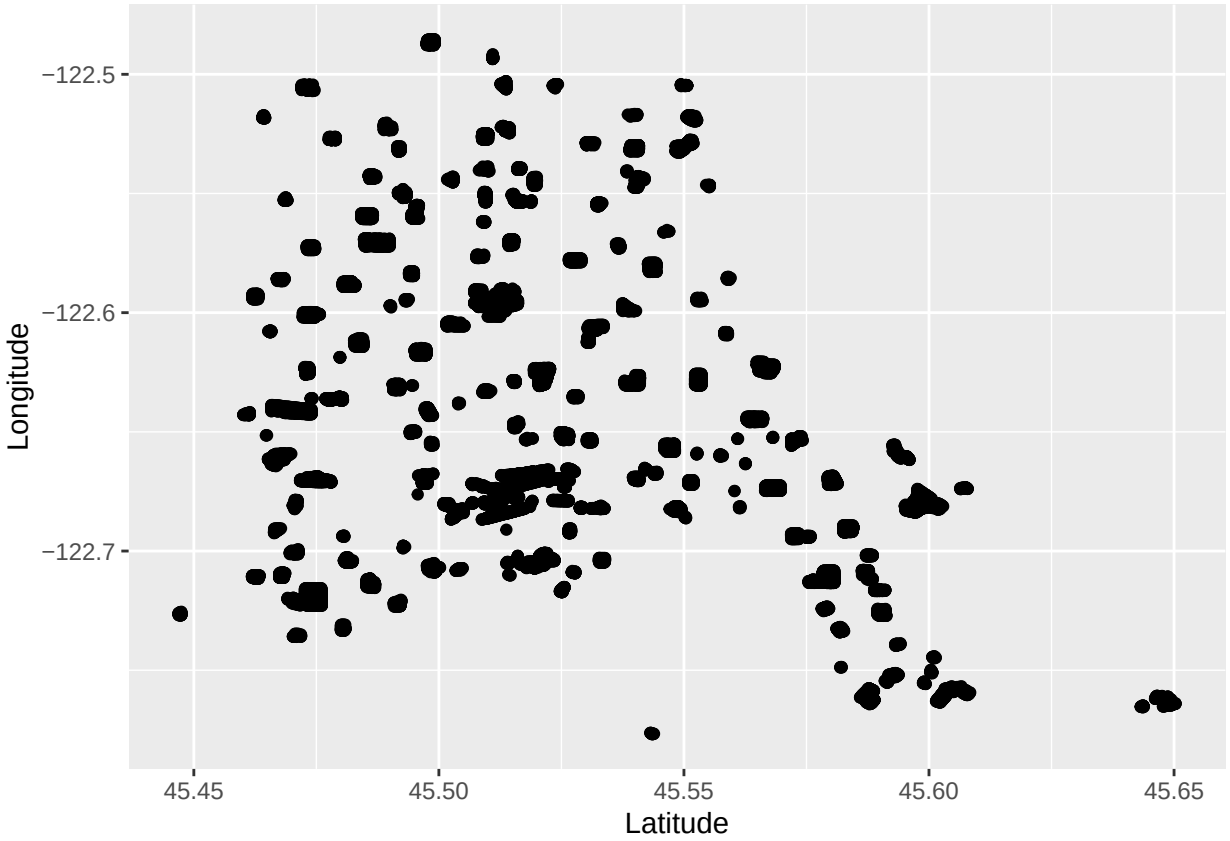
- The number of bins
- The fill color for the bars
- The color outlining the bars

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

histo(pdxTrees, DBH)
```




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

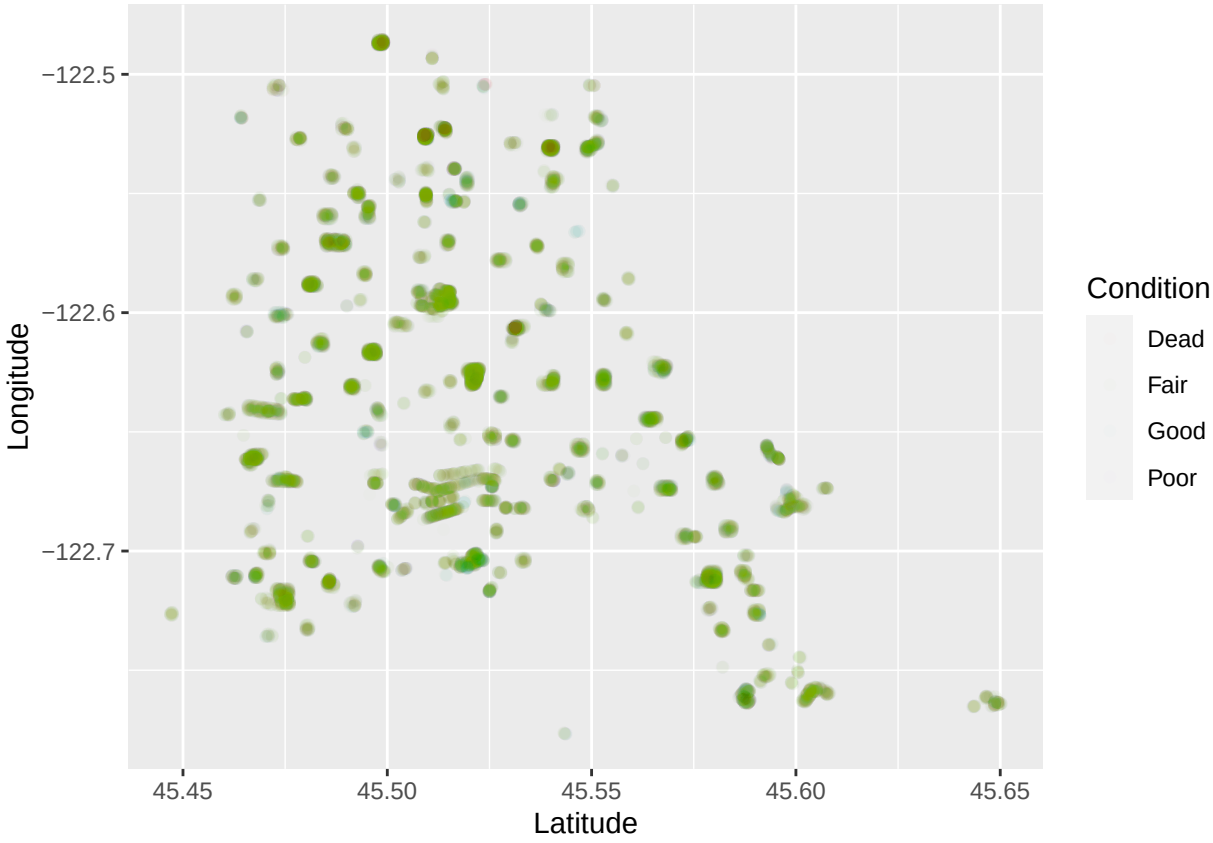
```
scat <- function(data, x, y){
  x <- enquo(x)
  y <- enquo(y)
  ggplot(data = data, mapping = aes(x = !!x, y = !!y)) +
    geom_point()
}

scat(pdxTrees, Latitude, Longitude)
```




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

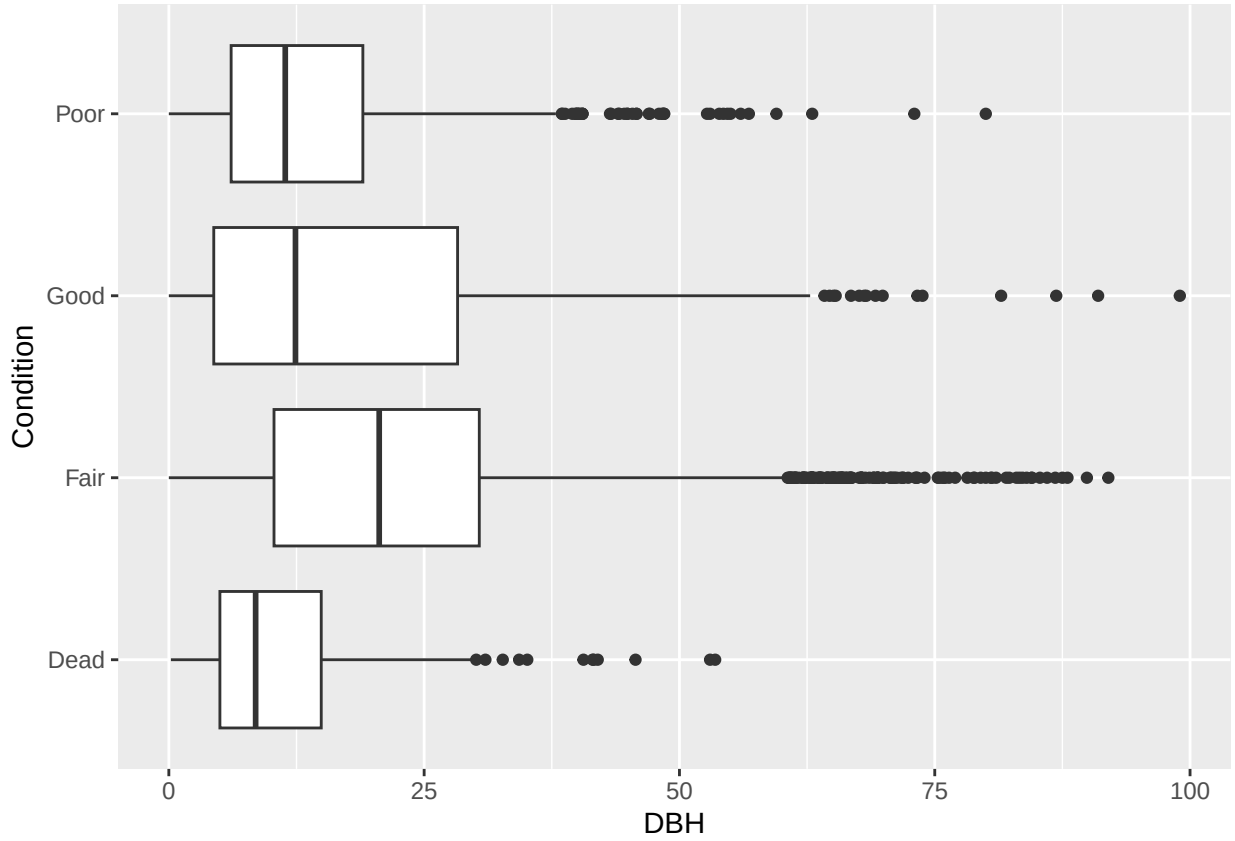
- Color the points by another variable.
- Set the transparency.

```
scatter <- function(data, x, y, color, alpha = 0.1) {
  x <- enquo(x)
  y <- enquo(y)
  color <- enquo(color)
  ggplot(data = data, mapping = aes(x = !!x, y = !!y, color = !!color)) +
    geom_point(alpha = alpha)
}

scatter(pdxTrees, Latitude, Longitude, Condition, alpha = 0.01)
```




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

```
box <- function(data, x, y){
  x <- enquo(x)
  y <- enquo(y)

  ggplot(data = data, mapping = aes(x = !!x, y = !!y)) +
    geom_boxplot()
}

box(pdxTrees, DBH, 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!

```
conditional_prop_table <- function(data, x, y){
  x <- enquo(x)
  y <- enquo(y)

  data %>%
    count(!!x, !!y) %>%
    group_by(!!x) %>%
    mutate(prop = n/sum(n)) %>%
    ungroup()
}

conditional_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.

```
summary <- function(data, x, y){
  x <- enquo(x)
  y <- enquo(y)

  data %>%
    group_by(!!y) %>%
    summarise(
      mean = mean(!!x),
      median = median(!!x),
      sd = sd(!!x),
      min = min(!!x),
      max = max(!!x),
      sample_size = n(),
      n_missing = sum(is.na(!!x))
    )
}

summary(pdxTrees, DBH, Condition)
```

```
## # A tibble: 4 x 8
##   Condition  mean median    sd   min   max sample_size 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**.

```
grab_name <- function(myname, myyear) {
  if (is.element(myname, babynames$name)) { babynames %>%
      filter(name == myname) %>%
       filter(year == myyear) %>%
      return()
```

```
    }
  else {
    stop("Name not found")
  }
}

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
```