

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)
library(dplyr)

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: list pdxTreesSmall: data frame ht: vector
- b. Find the 10th, 11th, and 12th values of ht.

```
ht10_12 <- ht[10:12]
print(ht10_12)
```

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

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

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

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

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

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

```
dat$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.)

```
DF_DBHcon4 <- pdxTreesSmall %>%
  filter(Common_Name == "Douglas-Fir") %>%
  select("Condition", "DBH") %>%
  slice(4)

print(DF_DBHcon4)
```

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

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

confidence_calc <- function(data, variable) {
  n <- length(data[[variable]])
  mean <- mean(data[[variable]])
  se <- sd(data[[variable]])/sqrt(n)
  t_val <- qt(p = 0.975, df = n - 1)

  lower <- mean - t_val * se
  upper <- mean + t_val * se

  results_ci <- list(
    n = n,
    mean = mean,
    se = se,
    t_value = t_val,
    lower_ci = lower,
    upper_ci = upper
  )

  return(results_ci)
}

confidence_calc(pdxTrees, "DBH")

```

```

## $n
## [1] 25534
##
## $mean
## [1] 20.61408
##
## $se
## [1] 0.08380945
##
## $t_value
## [1] 1.960057
##
## $lower_ci
## [1] 20.44981
##
## $upper_ci
## [1] 20.77835

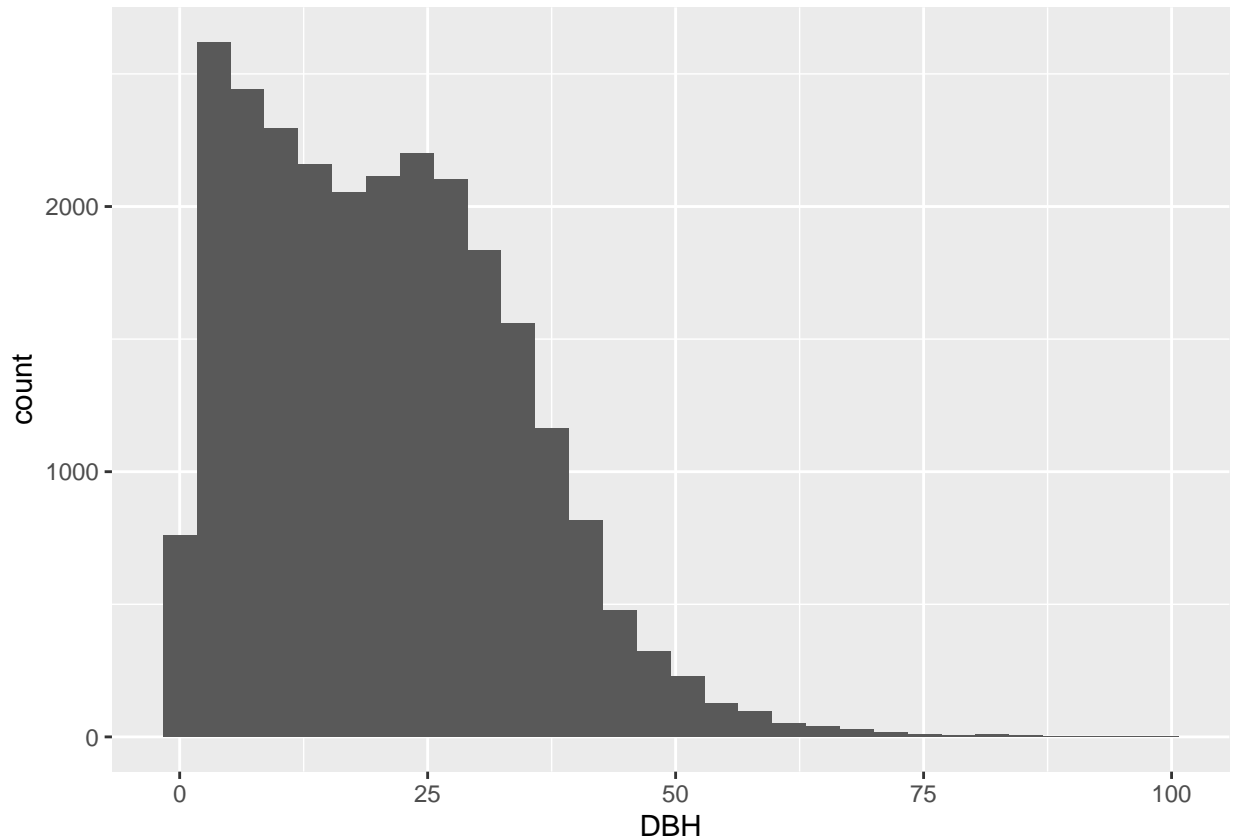
```

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`.

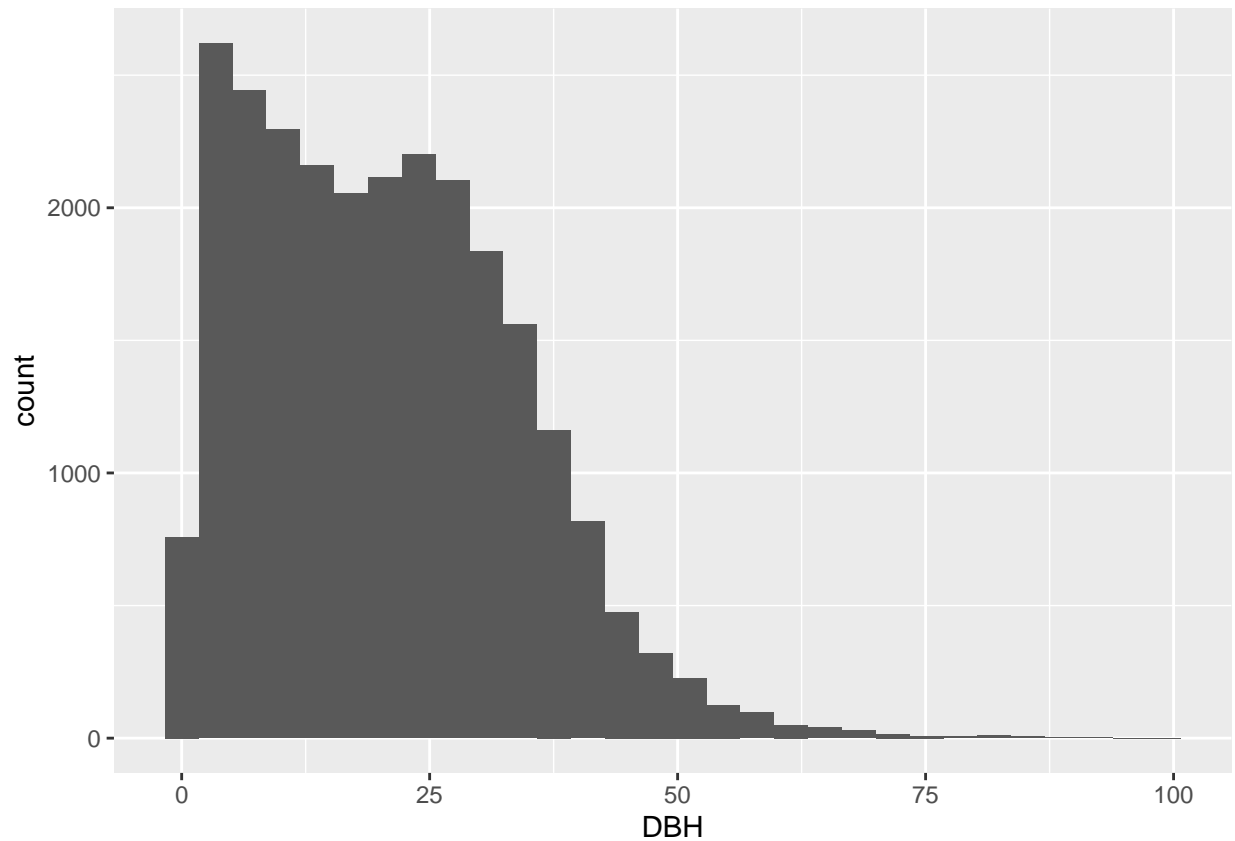
Here's an example of a wrapper for a histogram. Notice that I can't just list the variable name as an argument. The issue has to do with how many of the **tidyverse** functions evaluate the arguments. Therefore we have to quote (`enquo()`) and then unquote (`!!`) the arguments. (If you want to learn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

```
# 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()
}

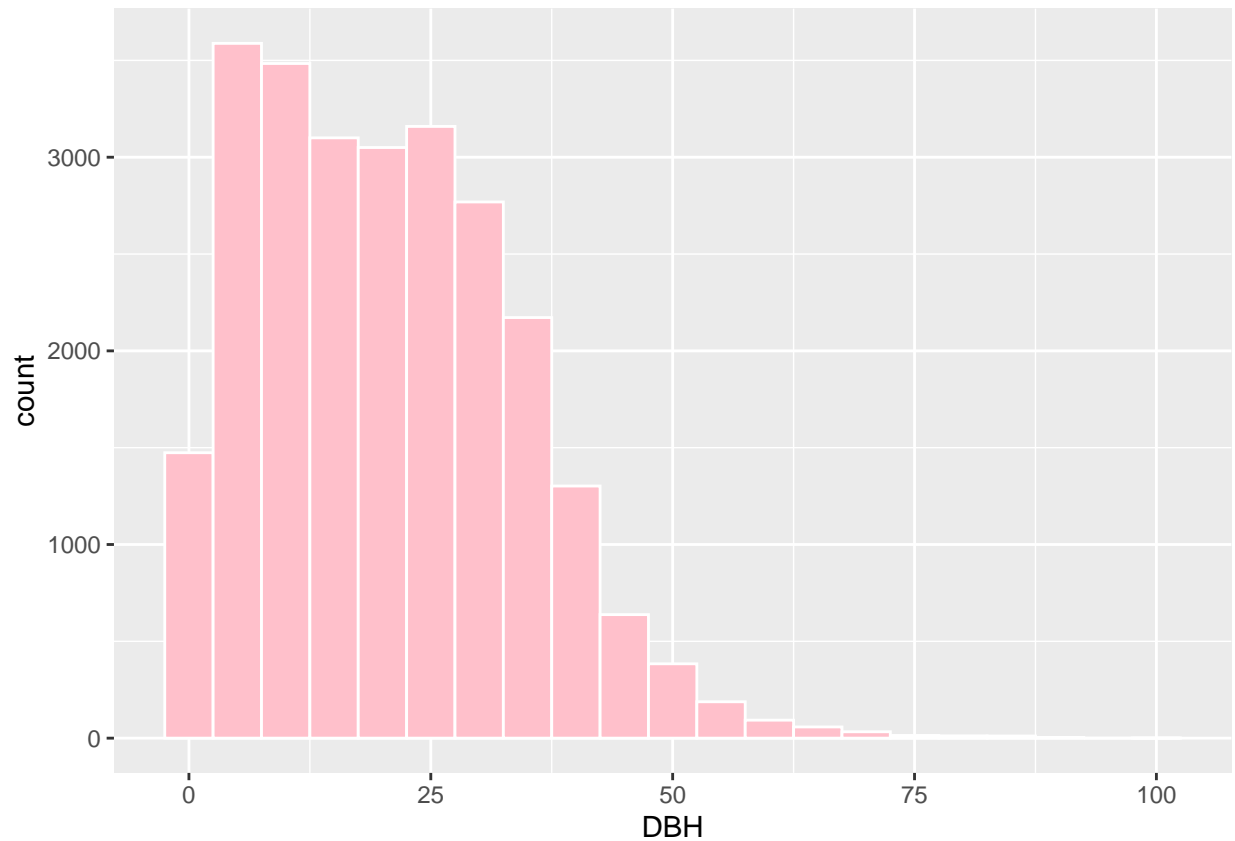
# Test it
histo(pdxTrees, DBH)
```



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

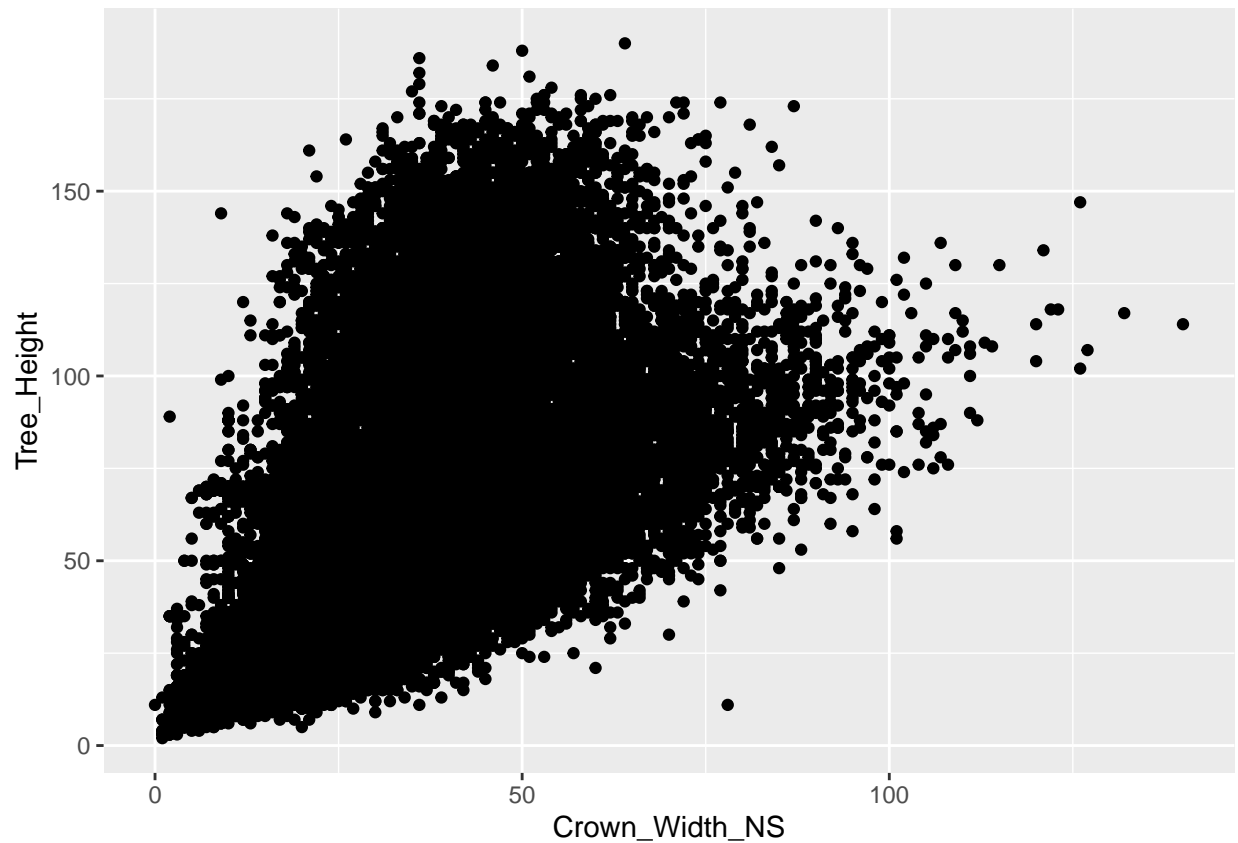
- The number of bins
- The fill color for the bars
- The color outlining the bars

```
histo <- function(data, x, bins = 5, fill_color = "grey", outline_color = "black"){  
  x <- enquo(x)  
  ggplot(data = data, mapping = aes(x = !!x)) +  
    geom_histogram(binwidth = bins, fill = fill_color, color = outline_color)  
}  
  
histo(pdxTrees, DBH, 5, "pink", "white")
```



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

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

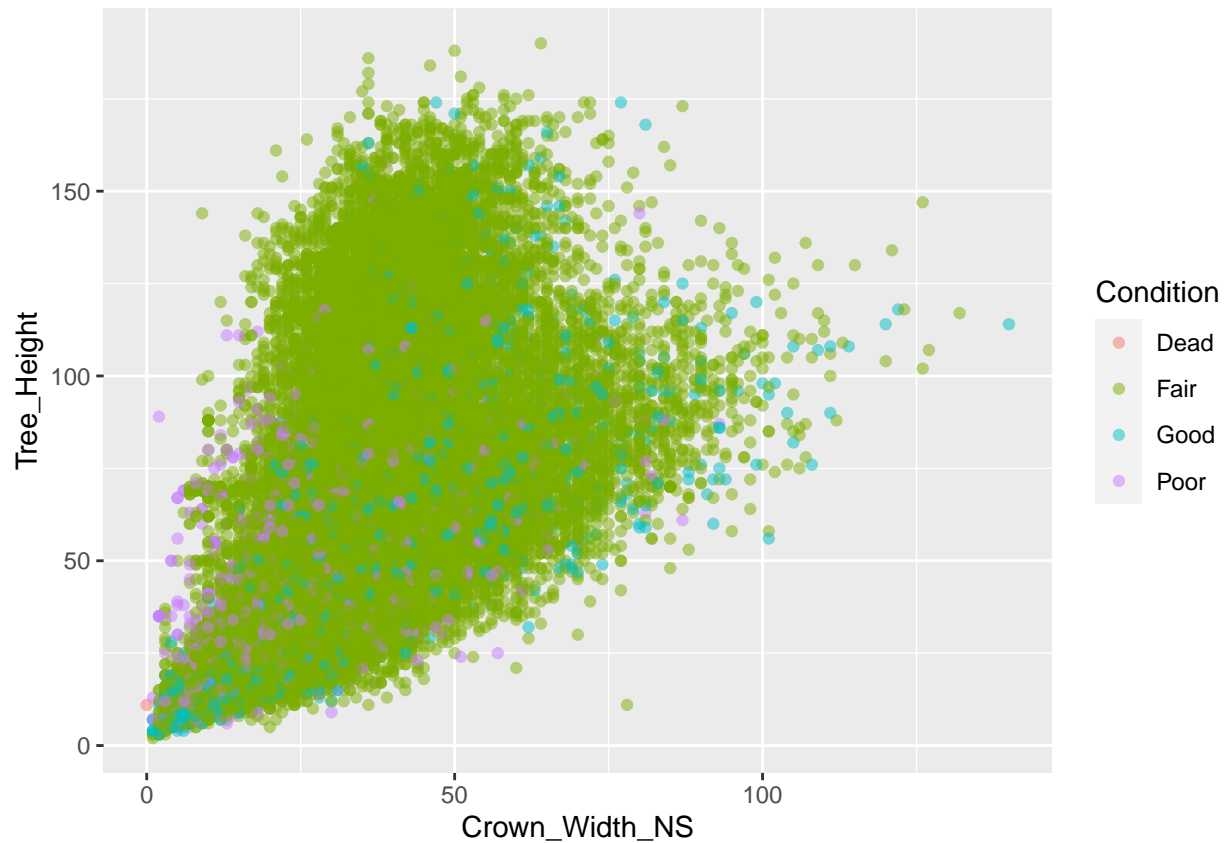


c. Modify your scatterplot function to allow the user to ...

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

```
scatter <- function(data, x, y, color_alt = NULL, transparency = 1.0){
  x <- enquo(x)
  y <- enquo(y)
  color_alt <- enquo(color_alt)
  ggplot(data = data, mapping = aes(x = !!x, y = !!y, color = !!color_alt)) +
    geom_point(alpha = transparency)
}

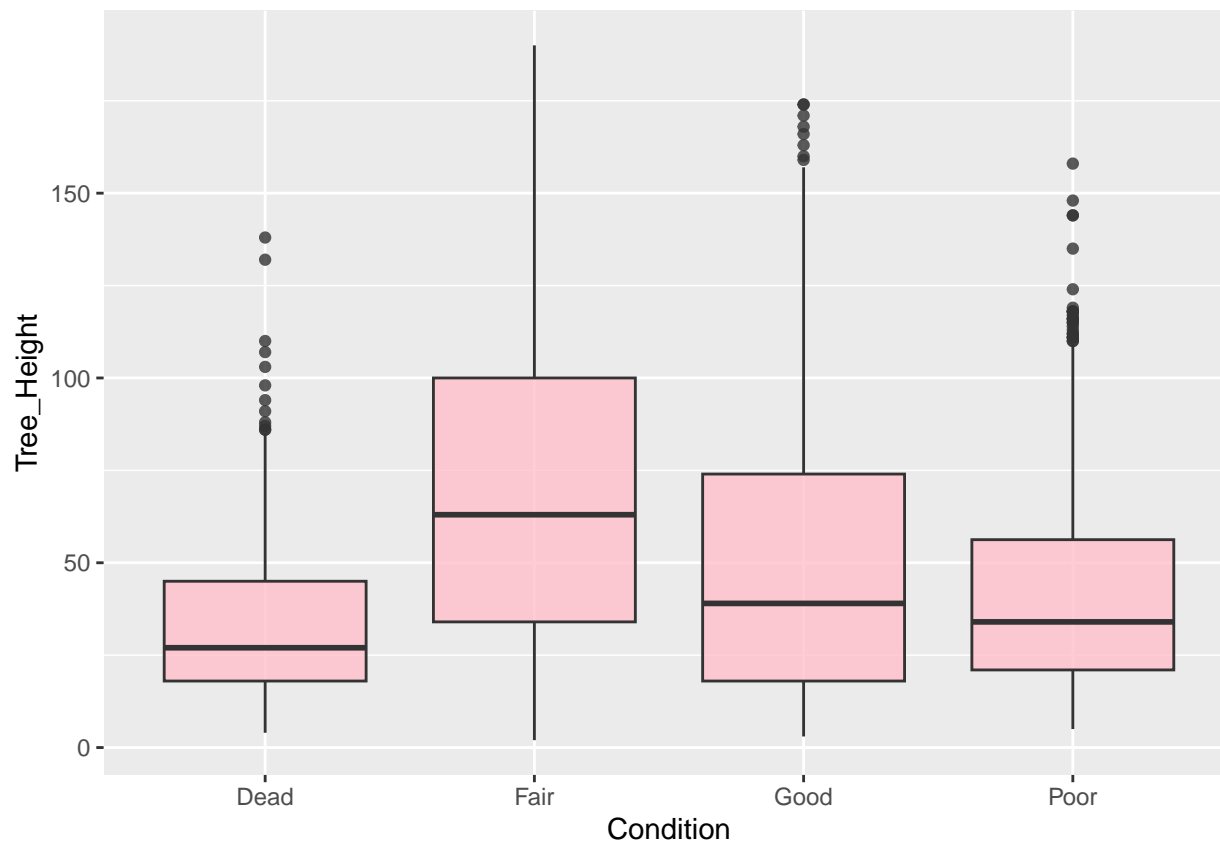
scatter(pdxTrees, Crown_Width_NS, Tree_Height, Condition, 0.5)
```



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

```
boxplot <- function(data, x, y, fill_color = "grey", transparency = 1.0){
  x <- enquo(x)
  y <- enquo(y)
  ggplot(data = data, mapping = aes(x = !!x, y = !!y)) +
    geom_boxplot(fill = fill_color, alpha = transparency)
}

boxplot(pdxTrees, Condition, Tree_Height, "pink", 0.8)
```

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
```

```
prop_tbl <- function(data, group_var, count_var) {
  group_var <- enquo(group_var)
  count_var <- enquo(count_var)
  data %>%
    count(!!group_var, !!count_var) %>%
    group_by(!!group_var) %>%
    mutate(prop = n / sum(n)) %>%
    ungroup()
}

prop_tbl(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.

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

data_stats(pdxTrees, Condition, Crown_Width_NS)
```

```
## # A tibble: 4 x 8
##   Condition mean median    sd   min   max count missing_values
##   <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <int>      <int>
```

## 1	Dead	15.7	10	19.1	0	37	264	261
## 2	Fair	36.3	35	18.0	1	132	22279	1
## 3	Good	32.0	27.5	22.4	1	140	1646	0
## 4	Poor	24.5	21	15.5	1	93	1345	1

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) {
  myyear <- enquote(myyear)
  results <- babynames %>%
    filter(name == myname, year == !!myyear)
  if(nrow(results) == 0) {
    stop("No matching rows for the provided name and year combination")
  }
  return(results)
}

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
```