Lab 5

Alyssa Andrichik

Math 241, Week 7

```
# Put all necessary libraries here
library(tidyverse)
```

Due: Thursday, March 12th 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 and be on the look out for code smells.
3. Practice creating tables.
4. Practice exploring the functionality of a package that is new to you.

Note

In some of your chunks, you will be testing your functions and want to see how your functions behavior when they error out. For those chunks, make sure to include `error = TRUE` in the chunk options. Otherwise, your document won't knit!

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)

# Creating the objects
dats <- list(pdxTrees  = head(get_pdxTrees_parks()),
             Births2015 = head(Births2015),
             HELPrct = head(HELPrct),
             sets = c("pdxTrees", "Births2015",
                      "HELPrct"))

pdxTreesSmall  <- head(get_pdxTrees_parks())

ht <- head(get_pdxTrees_parks()$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 is a list, pdxTreesSmall is a dataframe, and ht is a numeric vector.

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

```
class(dats)
```

```
## [1] "list"
```

```
ht[c(10,11,12)]
```

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

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

```
species <- pdxTreesSmall %>%
  select(Species)
species
```

```
## # A tibble: 6 x 1
##   Species
##   <chr>
## 1 PSME
## 2 PSME
## 3 CRLA
## 4 QURU
## 5 PSME
## 6 PSME
```

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

```
species.vector <- as.vector(species$Species)
species.vector
```

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

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

```
dats[["sets"]]
```

```
## [1] "pdxTrees"   "Births2015" "HELPrct"
```

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.
- Check that any data inputs are the appropriate classes.
  - And, provide a helpful error message if they aren't.
- 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)
pdxTrees_parks <- get_pdxTrees_parks()
thing1 <- length(pdxTrees_parks$DBH)
thing2 <- mean(pdxTrees_parks$DBH)
thing3 <- sd(pdxTrees_parks$DBH)/sqrt(thing1)
thing4 <- qt(p = .975, df = thing1 - 1)
thing5 <- thing2 - thing4*thing3
thing6 <- thing2 + thing4*thing3
```

```r
confidence.interval <- function(x, percent = 0.975, na.rm = TRUE, null = 0) {
  stopifnot(is.numeric(x) && is.numeric(percent) && percent <= 1 && percent >= 0)

  length <- length(x)
  mean <- mean(x)
  standard.error <- sd(x)/sqrt(length)
  tscore <- qt(p = percent, df = length - 1)
  lower.bound <- mean - tscore*standard.error
  upper.bound <- mean + tscore*standard.error

  return(data.frame(lower.bound = lower.bound, upper.bound = upper.bound))
  }

# Test it
confidence.interval(x = pdxTrees_parks$DBH)
```

```
##   lower.bound upper.bound
## 1    20.44981    20.77835
```

Problem 3: Wrapper Function for your ggplot

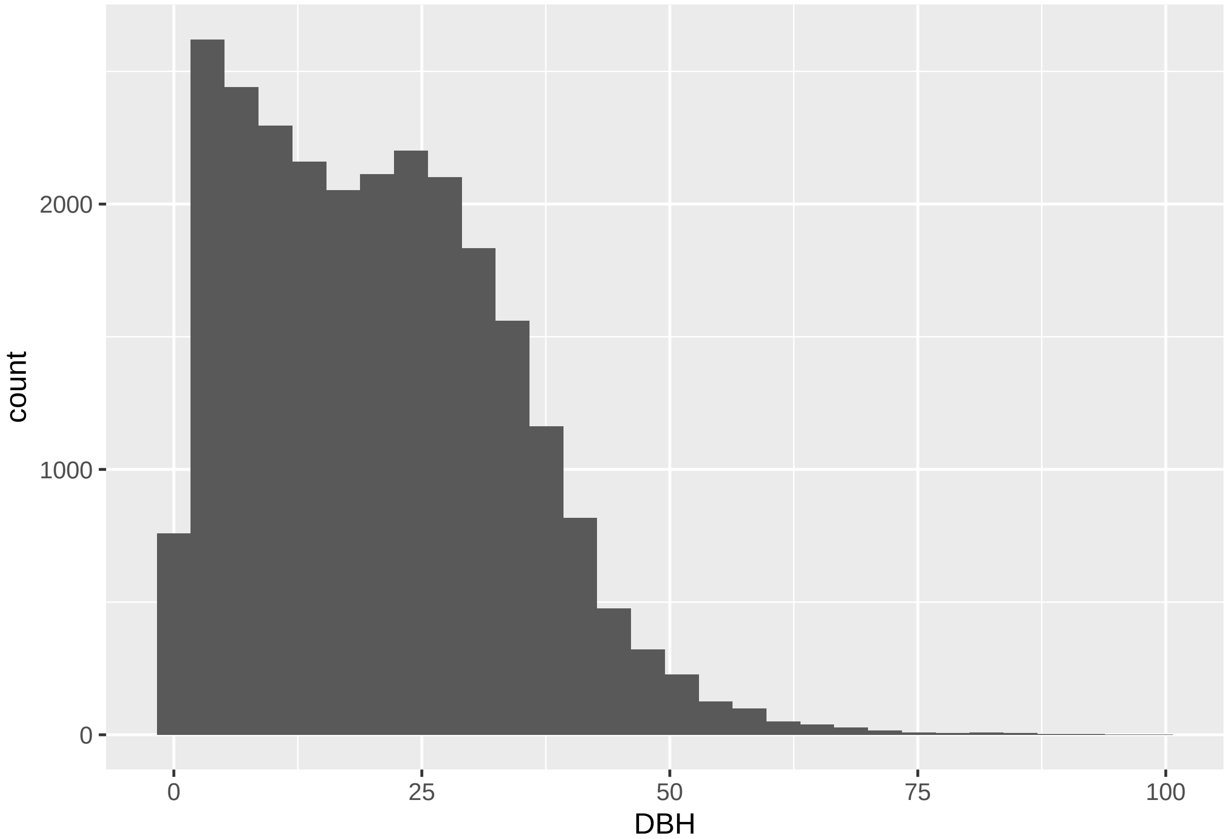
While we (i.e. Math 241 students) all love the grammar of graphics, not everyone else does. So for this problem, we are going to practice creating wrapper functions for `ggplot2`. **Note: Don't worry about checking the class of your inputs!**

Recall our discussion from class on tidy evaluation. To learn more check out these pages:

- For using `ggplot2` functions in your own functions
- For using `dplyr`functions in your own functions

Here's our example of a wrapper for a histogram.

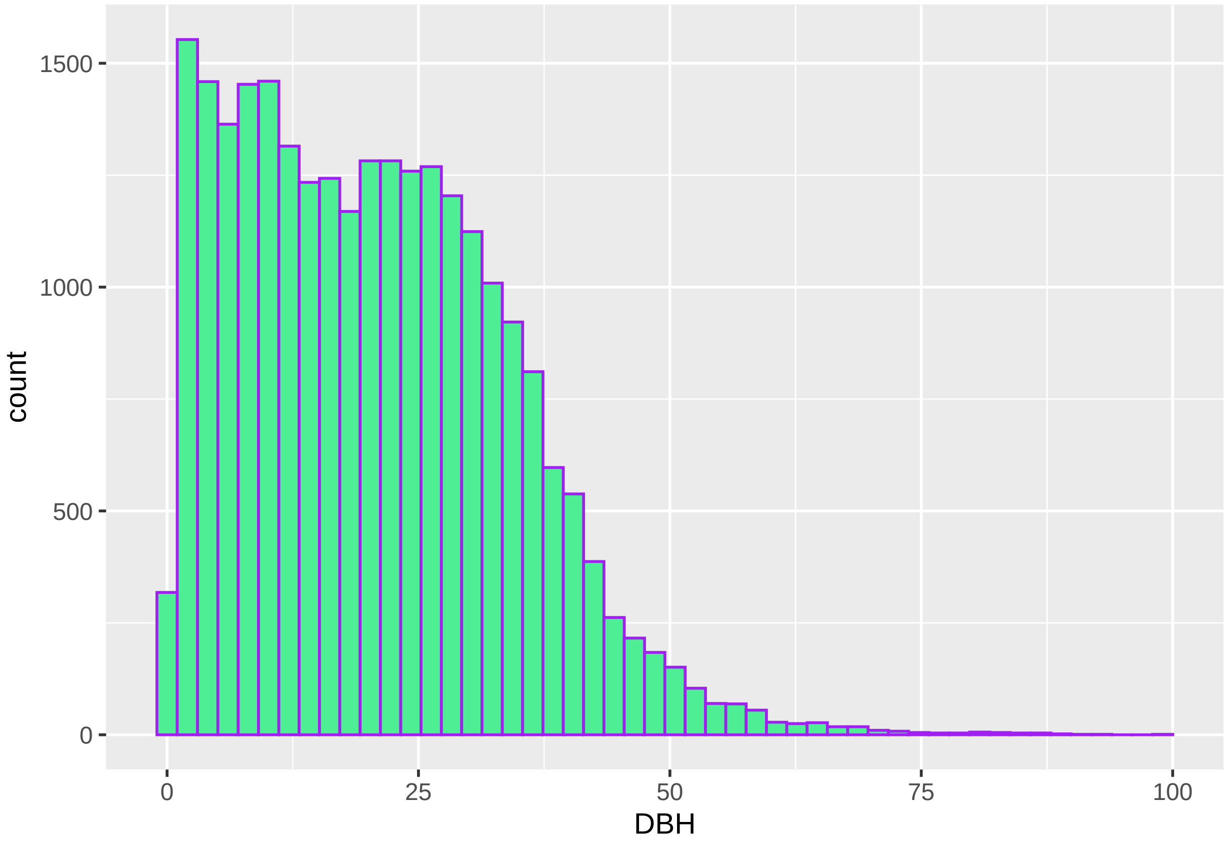
```r
# Minimal viable product working code
ggplot(data = pdxTrees_parks, mapping = aes(x = DBH)) +
  geom_histogram()
```



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

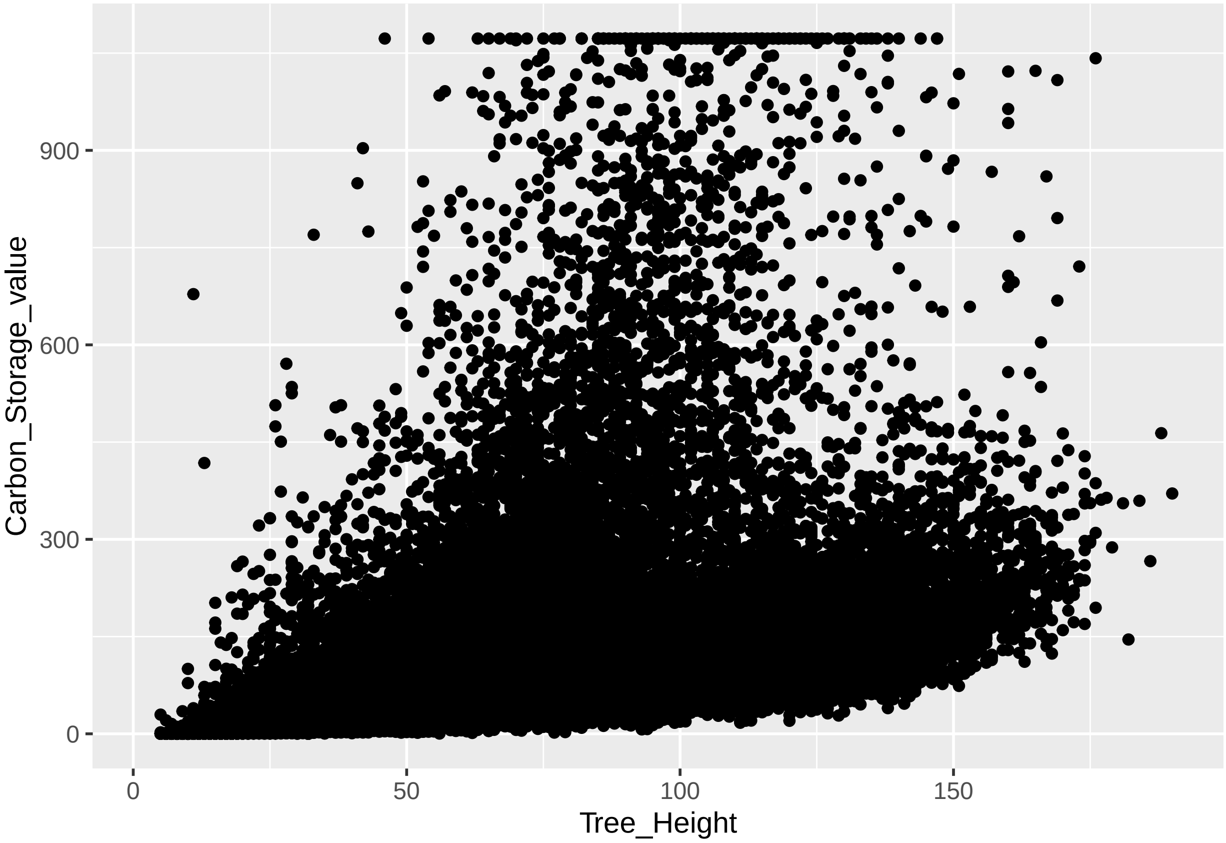
- The number of bins
- The fill color for the bars
- The color outlining the bars

```
# Function
histo <- function(data, x, bins = 30, fill.color = "dark grey", outline.color = NULL){
  ggplot(data = data, mapping = aes(x = {{ x }})) +
    geom_histogram(bins = bins, color = outline.color, fill = fill.color)
}

# Test it
histo(data = pdxTrees_parks, x = DBH, bins = 50, fill.color = "seagreen2", outline.color = "purple")
```



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

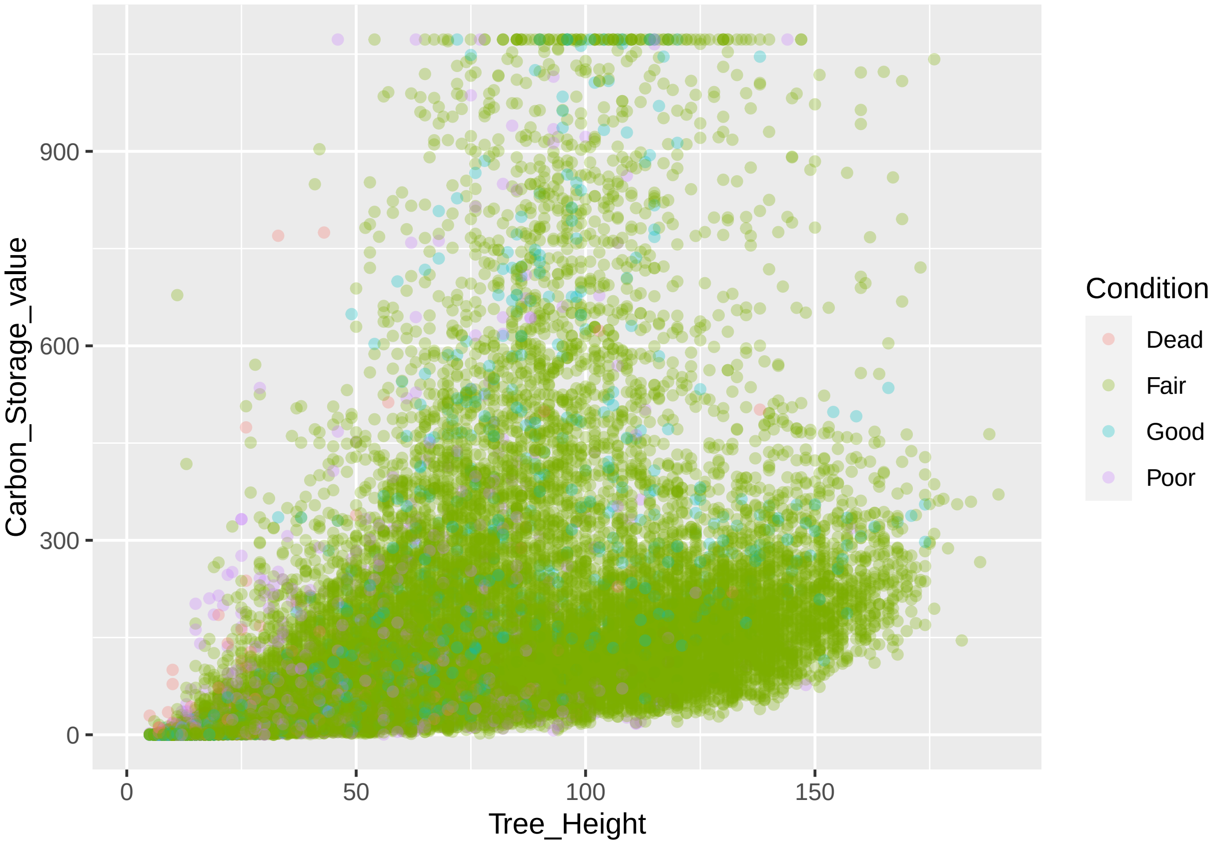
```
scatterplot <- function(data, x, y){
  ggplot(data = data, mapping = aes(x = {{x}}, y = {{y}}))+
    geom_point()
}
# Test it
scatterplot(data = pdxTrees_parks, x = Tree_Height, y = Carbon_Storage_value)
```



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

- Color the points by another variable.
- Set the transparency.
    - And include a check that the transparency input is within the appropriate range.

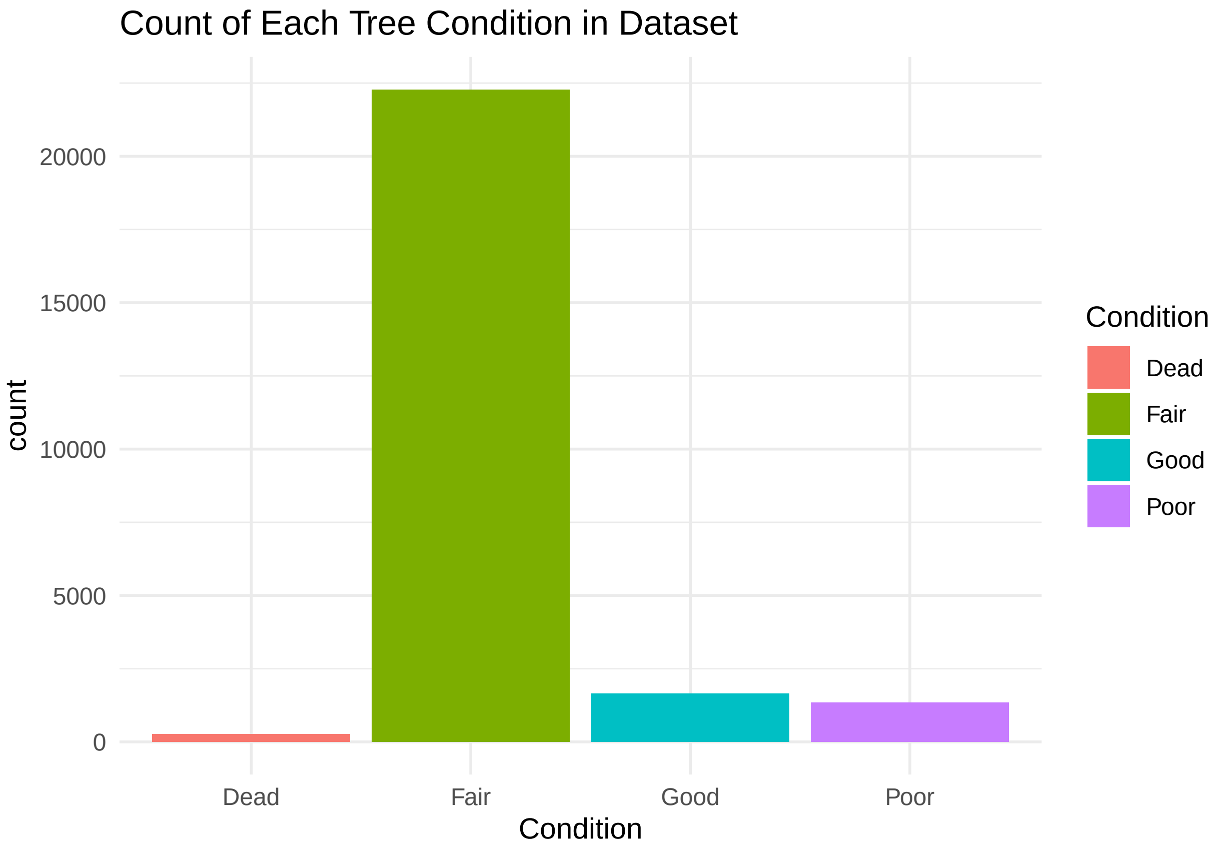
```
scatterplot <- function(data, x, y, fill = NULL, alpha = 1){
  stopifnot(alpha <= 1 && alpha > 0)
  ggplot(data = data, mapping = aes(x = {{x}}, y = {{y}}, color = {{fill}}))+
    geom_point(alpha = alpha)
}
# Test it
scatterplot(data = pdxTrees_parks, x = Tree_Height, y = Carbon_Storage_value, fill = Condition, alpha =
```



d. Write and test a function for your favorite `ggplot2` graph. Make sure to give the user at least 3 optional inputs that they can change to customize the plot.

The optional inputs I am including is the y axis variable, the fill variable, and the title. The data and the x axis variable are required inputs.

```r
the_best_plot <- function(data, x, y = NULL, variable.fill = NULL, title = NULL){
  if(is.null(y)){
    ggplot(data = data, mapping = aes(x = {{x}}, fill = {{variable.fill}}))+
      geom_bar() +
      theme_minimal() +
      labs(title = title)
  }else{
    ggplot(data = data, mapping = aes(x = {{x}}, y = {{y}}, fill = {{variable.fill}}))+
      geom_point(stat = 'identity') +
      theme_minimal() +
      labs(title = title)
  }
}
# Test it
MYtitle <- "Count of Each Tree Condition in Dataset"
the_best_plot(data = pdxTrees_parks, x = Condition, variable.fill = Condition, title = MYtitle)
```



Problem 4: Functioning `dplyr`

How many times did I ask you to make a table (data frame) of conditional proportions on Lab 3?! Who is wishing they'd written a swanky R function to do the work for them? Let's practice writing functions for common data wrangling operations. (Again, don't worry about checking the class of the input data.)

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 handle the tidy evaluation. And, make sure to test your function!

```r
conditional.proportions.table <- function(data, variable1, variable2, na.rm = FALSE) {
  data %>%
    count({{variable1}}, {{variable2}}) %>%
    group_by({{variable1}}) %>%
    mutate(prop = n/sum(n)) %>%
    ungroup()
  }

conditional.proportions.table(data = pdxTrees_parks, variable1 = Native, variable2 = 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).

```
summary.table <- function(data, char.variable, quant.variable, na.rm = FALSE) {
  data %>%
    group_by({{char.variable}}) %>%
    summarise(
      sample_size = n(),
      mean = mean({{quant.variable}}, na.rm = TRUE),
      median = median({{quant.variable}}, na.rm = TRUE),
      sd = sd({{quant.variable}}, na.rm = TRUE),
      min = min({{quant.variable}}, na.rm = TRUE),
      max = max({{quant.variable}}, na.rm = TRUE),
      num_na = sum(is.na({{quant.variable}}))
           ) %>%
    ungroup()
  }

summary.table(data = pdxTrees_parks, char.variable = Condition, quant.variable = Tree_Height)
```

```
## # A tibble: 4 x 8
##   Condition sample_size  mean median    sd   min   max num_na
## * <chr>           <int> <dbl>  <dbl> <dbl> <dbl> <dbl>  <int>
## 1 Dead              264  34.2     27  23.3     4   138      0
## 2 Fair            22279  68.7     63  40.4     2   190      0
## 3 Good             1646  50.2     39  37.6     3   174      0
## 4 Poor             1345  41.1     34  26.2     5   158      1
```

Problem 5: Your Turn!

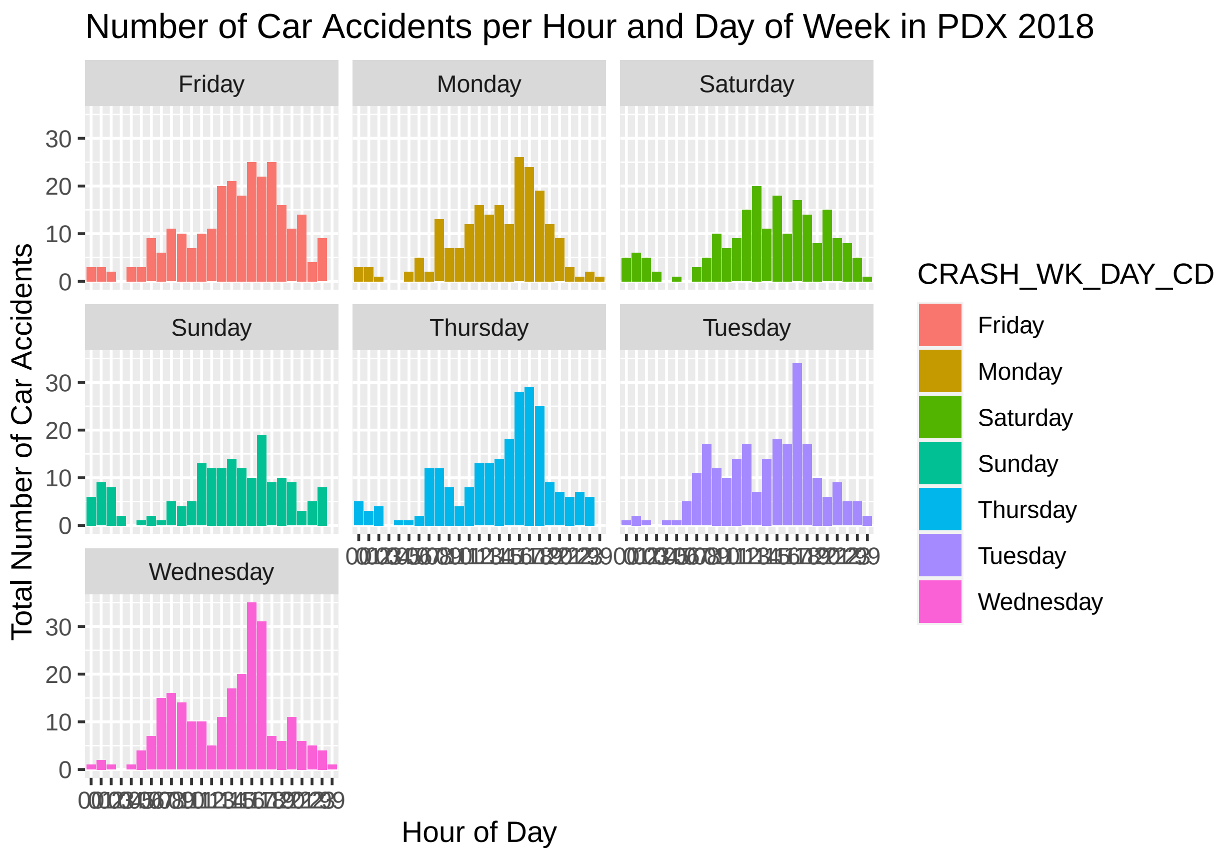
Find some R code you have written that could use some refactoring. It could be a chunk from a previous lab, from your mini-project 1, from your senior thesis, from an intro Chem lab, or from somewhere else entirely. It needs to be at least 30 lines long. (If you are having trouble finding that much code, you can grab multiple chunks from a previous lab.)

a. Paste the code in an R chunk.

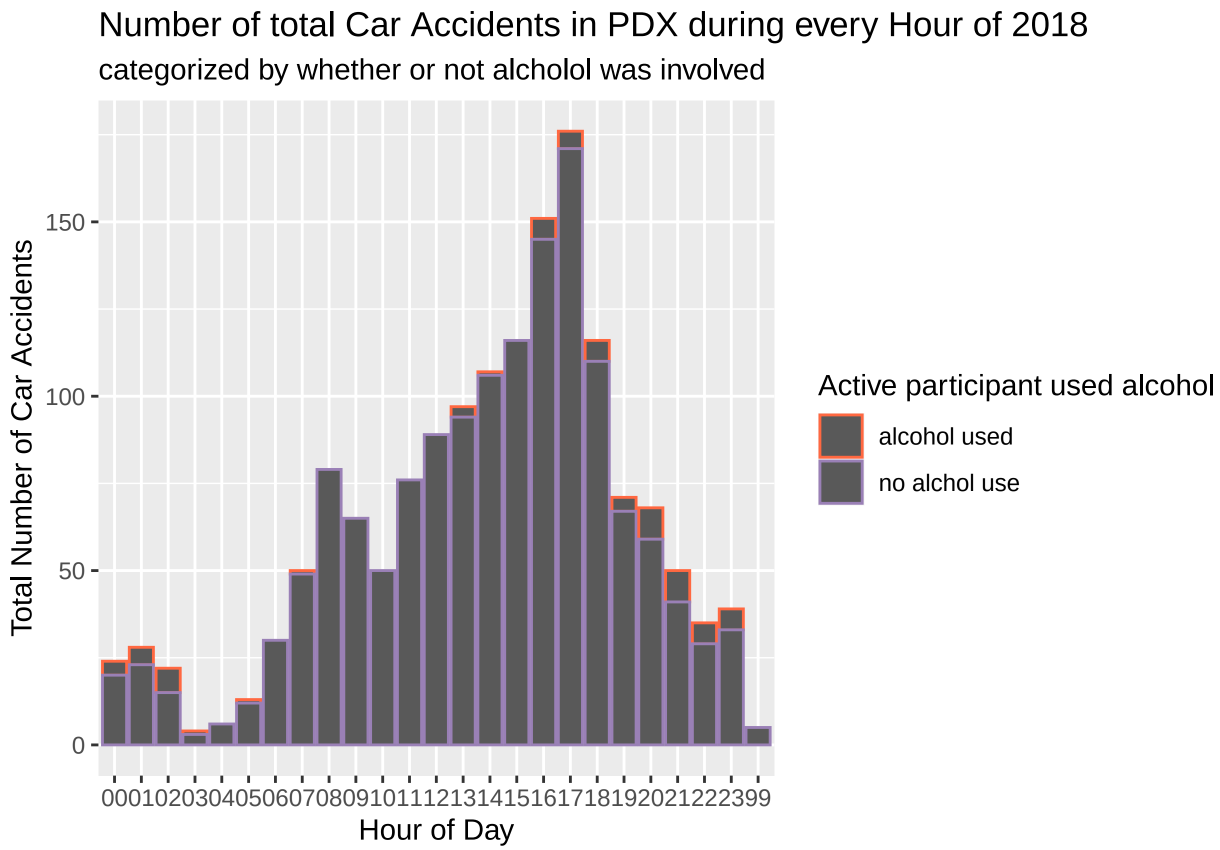
```
#From Lab 2
pdx_crash_2018 <- read_csv("/home/courses/math241s21/Data/pdx_crash_2018_page1.csv")
library(plyr) #messes with rename function if I put it earlier in the file
```

```
alyssa_crash <- pdx_crash_2018 %>%
  select(CRASH_HR_NO, CRASH_WK_DAY_CD, HWY_MED_NM, ALCHL_INVLV_FLG)

alyssa_crash$CRASH_WK_DAY_CD <- as.character(alyssa_crash$CRASH_WK_DAY_CD)
alyssa_crash$ALCHL_INVLV_FLG <- as.character(alyssa_crash$ALCHL_INVLV_FLG)
alyssa_crash$CRASH_WK_DAY_CD <- revalue(alyssa_crash$CRASH_WK_DAY_CD,
                                        c("1"="Sunday", "2"="Monday",
                                          "3" = "Tuesday", "4" = "Wednesday",
                                          "5" = "Thursday", "6" = "Friday",
                                          "7" = "Saturday"))
```

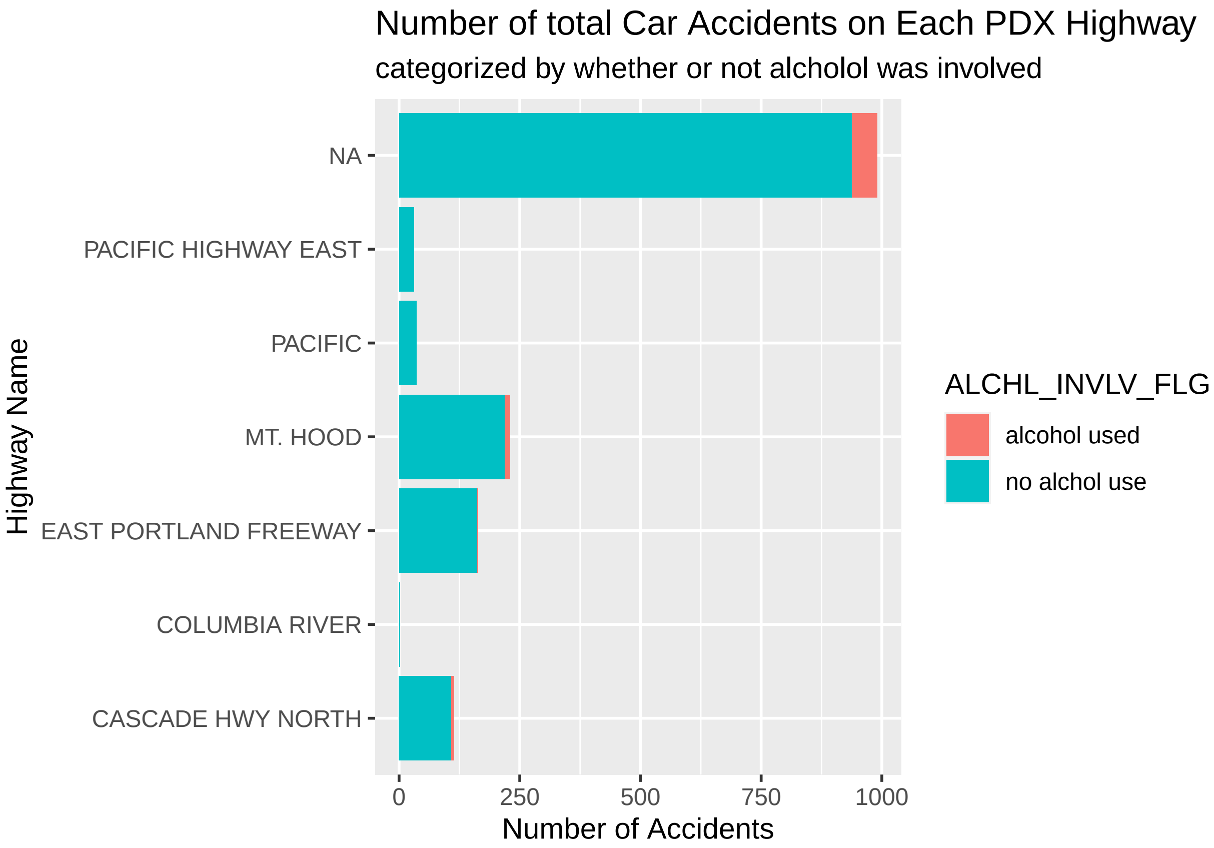
```
alyssa_crash$ALCHL_INVLV_FLG <- revalue(alyssa_crash$ALCHL_INVLV_FLG,
                                       c("0"="no alchol use", "1" = "alcohol used"))

#plot 1
ggplot(alyssa_crash, aes(x = CRASH_HR_NO, fill = CRASH_WK_DAY_CD)) +
  geom_bar(aes(y = ..count..), stat = "count") +
  labs(x = "Hour of Day", y = "Total Number of Car Accidents",
        title = "Number of Car Accidents per Hour and Day of Week in PDX 2018",
        color = "Day of Week") +
  facet_wrap(~ CRASH_WK_DAY_CD)
```



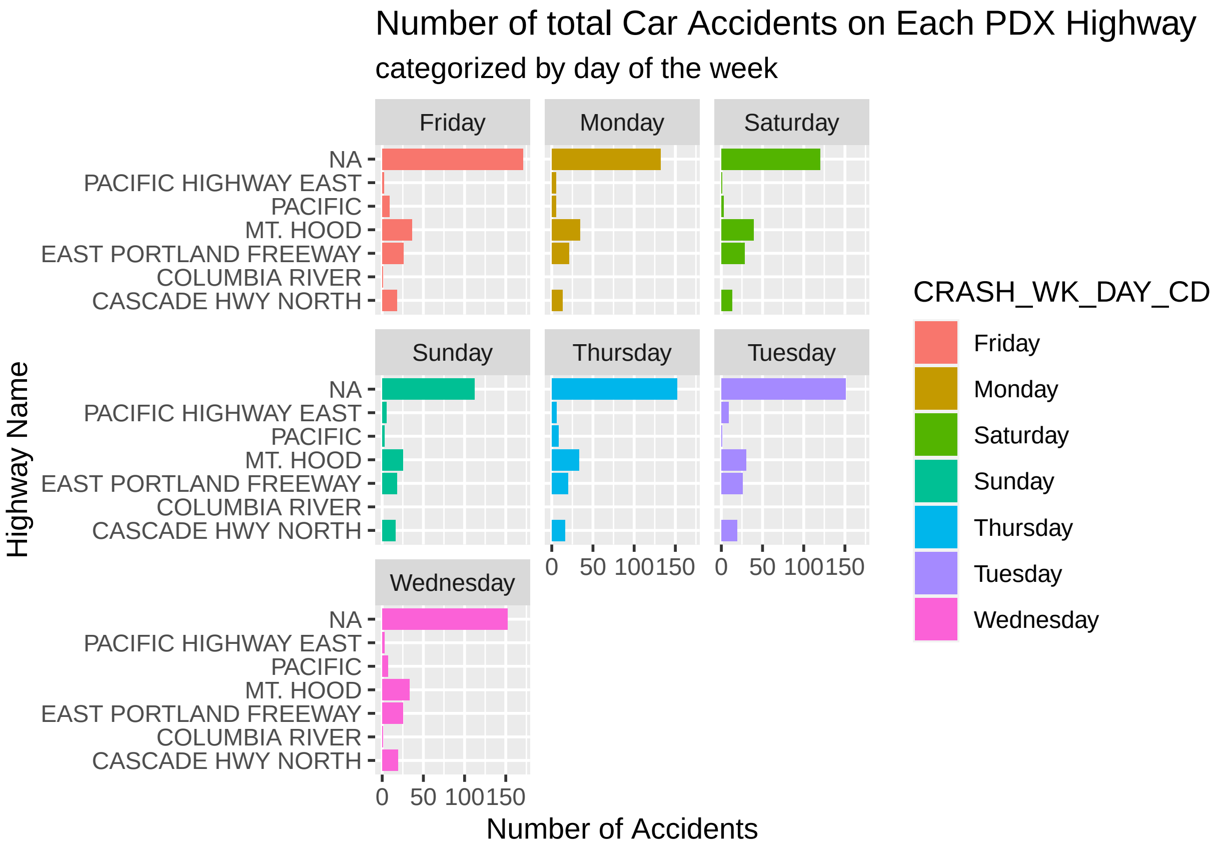
```
#plot 2
ggplot(alyssa_crash, aes(x = CRASH_HR_NO, color = ALCHL_INVLV_FLG)) +
  geom_bar(aes(y = ..count..), stat = "count")+
  scale_color_manual(values = c("#FF6841", "#9A80B6")) +
  labs(x = "Hour of Day", y = "Total Number of Car Accidents",
       title = "Number of total Car Accidents in PDX during every Hour of 2018",
       subtitle = "categorized by whether or not alcholol was involved",
       color = "Active participant used alcohol")
```



```
#plot 3
ggplot(alyssa_crash, aes(y = HWY_MED_NM, fill = ALCHL_INVLV_FLG)) +
  geom_bar(aes(x = ..count..), stat = "count") +
  scale_color_manual(values = c("#FF6841", "#9A80B6")) +
  labs(x = "Number of Accidents", y = "Highway Name",
       title = "Number of total Car Accidents on Each PDX Highway",
       subtitle = "categorized by whether or not alcholol was involved",
       color = "Active participant used alcohol")
```



```
#plot 4
ggplot(alyssa_crash, aes(y = HWY_MED_NM, fill = CRASH_WK_DAY_CD)) +
  geom_bar(aes(x = ..count..), stat = "count")+
  facet_wrap(~ CRASH_WK_DAY_CD)+
  labs(x = "Number of Accidents", y = "Highway Name",
        title = "Number of total Car Accidents on Each PDX Highway",
        subtitle = "categorized by day of the week",
        color = "Day of week")
```

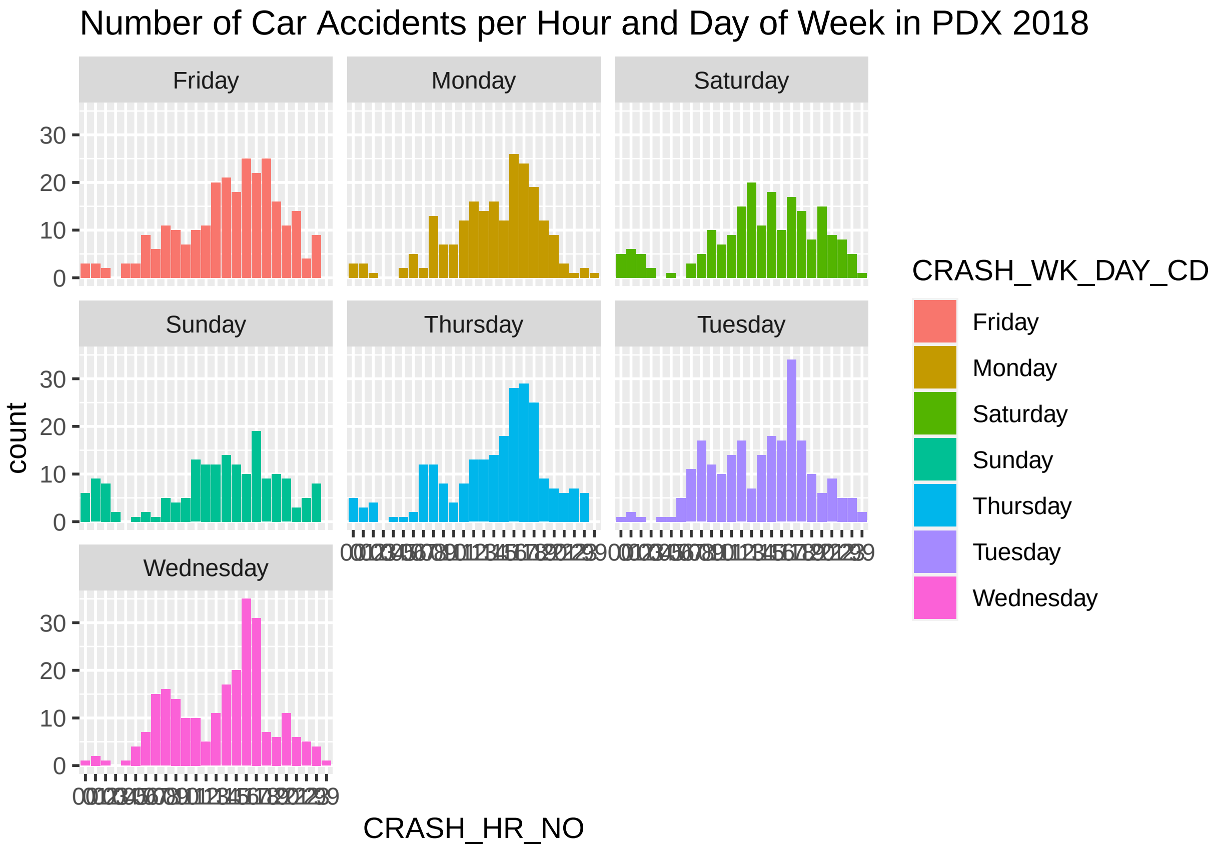


b. Discuss any code smells you see in your code. I repeated code to make each individual plot, making it super long and hard to read. This code also has a few misspellings and weird spacings.

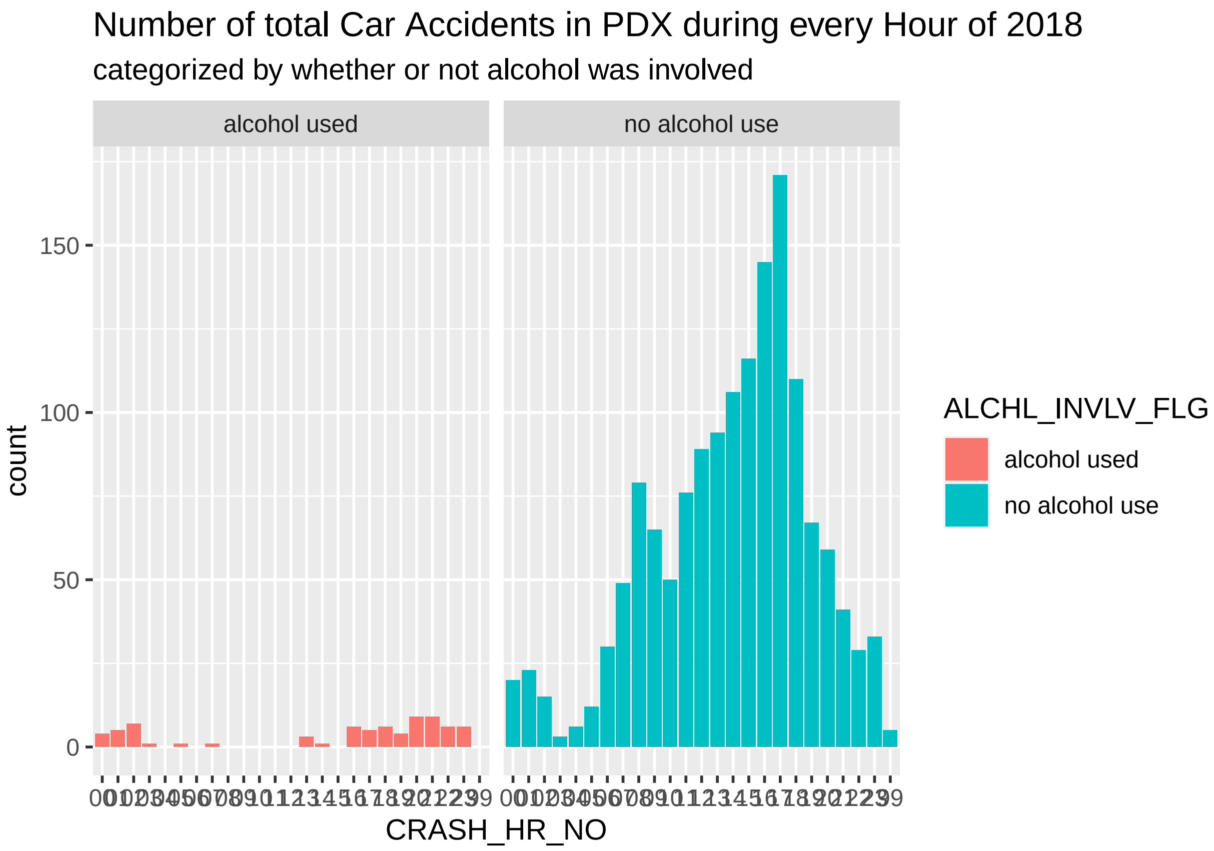
c. Refactor the code. Use the ideas we discussed over this week (i.e., consider creating functions, good naming practices, removing `setwd()`, . . .).

```
#subset of variables used for my plots
alyssa_crash <- pdx_crash_2018 %>%
  select(CRASH_HR_NO, CRASH_WK_DAY_CD, HWY_MED_NM, ALCHL_INVLV_FLG)
#revalue and recode
alyssa_crash$CRASH_WK_DAY_CD <- as.character(alyssa_crash$CRASH_WK_DAY_CD)
alyssa_crash$ALCHL_INVLV_FLG <- as.character(alyssa_crash$ALCHL_INVLV_FLG)
alyssa_crash$CRASH_WK_DAY_CD <- revalue(alyssa_crash$CRASH_WK_DAY_CD,
                                        c("1"="Sunday", "2"="Monday",
                                          "3" = "Tuesday", "4" = "Wednesday",
                                          "5" = "Thursday", "6" = "Friday",
                                          "7" = "Saturday"))
alyssa_crash$ALCHL_INVLV_FLG <- revalue(alyssa_crash$ALCHL_INVLV_FLG,
                                        c("0" = "no alcohol use", "1" = "alcohol used"))
#create plotting function
car.crash.plot <- function(data, x, flip.axis = "no", facet = "no", variable.fill = NULL,
                           title = NULL, subtitle = NULL){
  plot <- ggplot(data = data, mapping = aes(x = {{x}}, fill = {{variable.fill}}))+
    geom_bar() +
    labs(title = title,
         subtitle = subtitle)
  if(flip.axis == "no" && facet == "no"){
    return(plot)
```

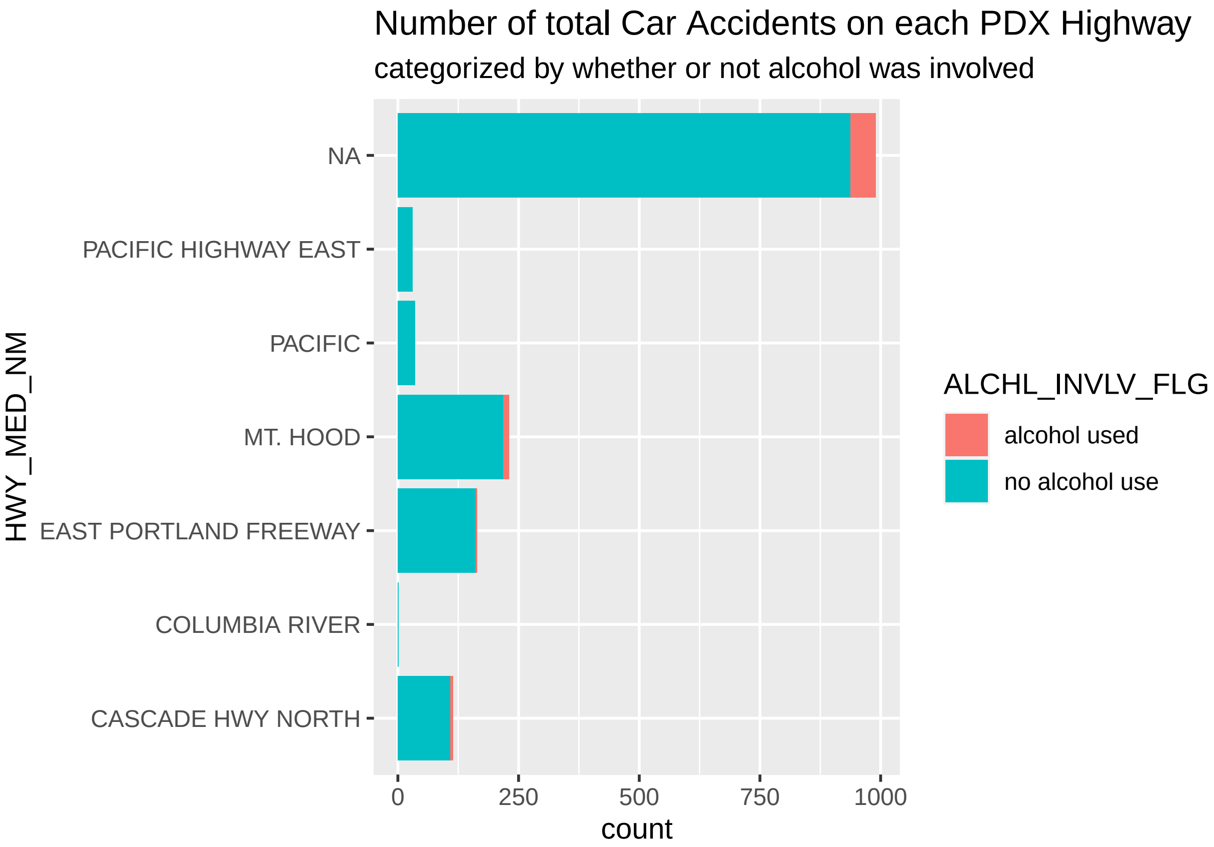
```r
  }else if(flip.axis == "yes" && facet == "no"){
    plot <- plot + coord_flip()
    return(plot)
  }else if(flip.axis == "yes" && facet == "yes"){
    plot <- plot + facet_wrap(vars({{variable.fill}})) + coord_flip()
    return(plot)
  }else if(flip.axis == "no" && facet == "yes"){
    plot <- plot + facet_wrap(vars({{variable.fill}}))
    return(plot)
  }
}
#plot 1
title1 <- "Number of Car Accidents per Hour and Day of Week in PDX 2018"
car.crash.plot(data = alyssa_crash, x = CRASH_HR_NO, facet = "yes",
               variable.fill = CRASH_WK_DAY_CD, title = title1)
```



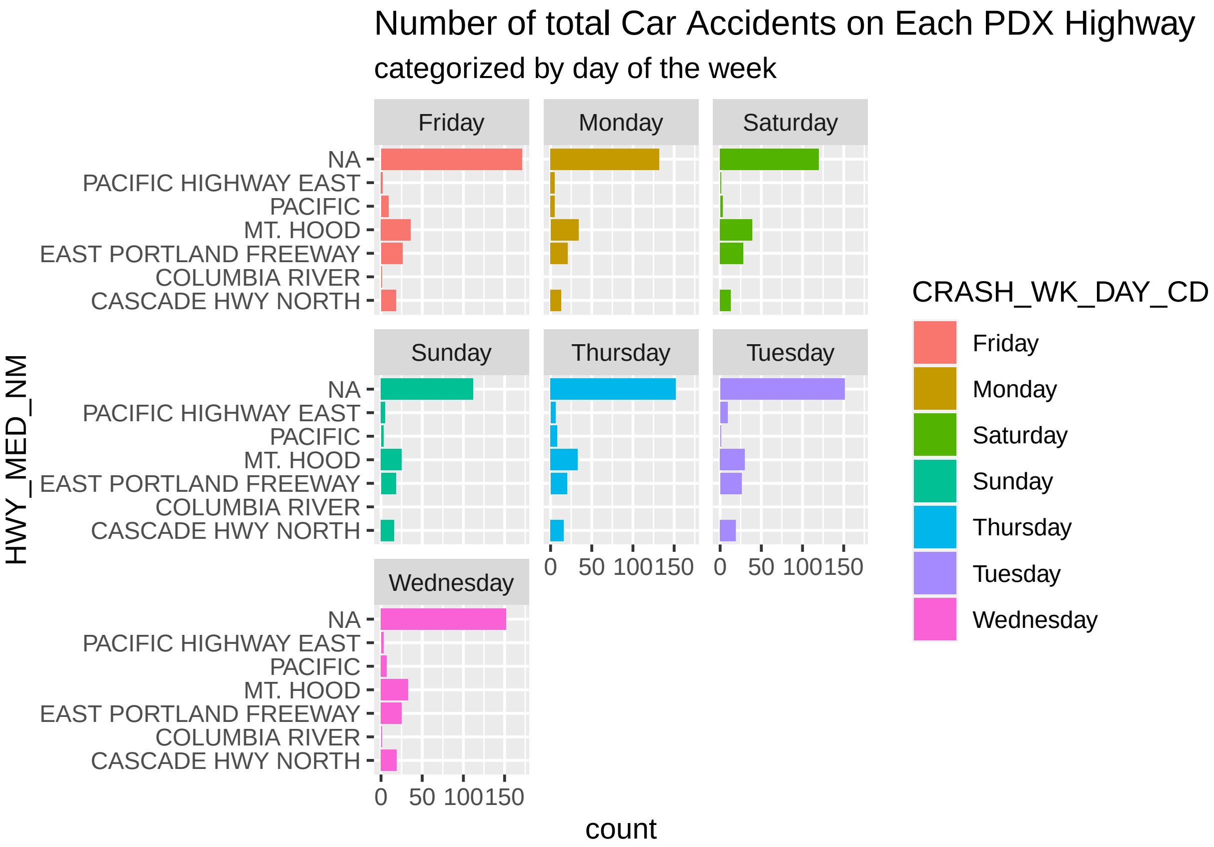
```r
#plot 2
title2 <- "Number of total Car Accidents in PDX during every Hour of 2018"
subtitle2 <- "categorized by whether or not alcohol was involved"
car.crash.plot(data = alyssa_crash, x = CRASH_HR_NO, facet = "yes",
               variable.fill = ALCHL_INVLV_FLG, title = title2, subtitle = subtitle2)
```



```r
#plot 3
title3 <- "Number of total Car Accidents on each PDX Highway"
subtitle3 <- "categorized by whether or not alcohol was involved"
car.crash.plot(data = alyssa_crash, x = HWY_MED_NM, flip.axis = "yes",
               variable.fill = ALCHL_INVLV_FLG, title = title3, subtitle = subtitle3)
```



```
#plot 4
title4 <- "Number of total Car Accidents on Each PDX Highway"
subtitle4 <- "categorized by day of the week"
car.crash.plot(data = alyssa_crash, x = HWY_MED_NM, flip.axis = "yes", facet = "yes",
               variable.fill = CRASH_WK_DAY_CD, title = title4, subtitle = subtitle4)
```



d. Explain the changes you made to the code and why you think they made the code better.

I created a function that will make the plots how I want them without repeating code over and over again. I specifically make the function ask only for the x axis variable as all of my y axis are counts of the x variable by the fill variable. I create an option for the x and y axes to flip if the plot would look better with the bars horizontal rather than vertical. I also create the option for the plot to be facet_wrap. This simplifies the code for each plot since I can call the function I created to execute the specific ggplot code I am looking for.

Beyond creating a function to do my plotting for me, I also fixed misspellings and weird spacing, as well as added notes to explain what I am doing in the code to make it more readable. The code ends up being slightly longer, but FAR more digestible. Plus, if I wanted to make a couple more plots, it would end up being shorter if I used the function I created.

Problem 6: Tubular Tables Time!

For this problem, let's practice creating display tables.

a. Let's first return to the Reed graduate rates table from Lab 4, Problem 3. This time I want you to scrape the table from Reed's website but leave it in the untidy format where each row represents a year. Use `gt` to create a nice display table. Make sure your table includes:

```
library(gt)
library(rnoaa)
library(rvest)
library(httr)
#Store url
url2 <- "https://www.reed.edu/ir/gradrateshist.html"

# Ask first
```

```r
robotstxt::paths_allowed(url2)
```

```
## [1] TRUE
```

```r
## Scrape html and store table

#Option 1: Grab all the tables and then navigate to the one you wanted.
tables2 <- url2 %>%
  read_html() %>%
  html_nodes(css = "table")

grad_time <- html_table(tables2[[1]], fill = TRUE)
colnames(grad_time)[3:5] <-c("4 years","5 years", "6 years")
grad_time = grad_time[-1,]
```

- A title
- A column spanner for “Graduated in”
- A footnote for which cells represent preliminary data (See more examples of locating the correct cells here)
- Color for rates and cohort sizes (Note: These won’t appear in the output pdf. That is okay.)
- Sources at the bottom to link to the website and to list the caveats.

```r
grad_time %>%
  gt() %>%
  tab_spanner(label = "Graduated in",
              columns = vars("4 years", "5 years", "6 years")) %>%
  tab_header(title = md("**Graduation Rate of Reedies**"),
             subtitle = "from years 1982-2016")
```

Graduation Rate of Reedies

from years 1982-2016

		Graduated in		
First-year students who entered fall of...	Number in Cohort	4 years	5 years	6 years
2016	353	66%*	-	-
2015	418	61%	70%*	-
2014	346	62%	73%	77%*
2013	354	64%	72%	76%
2012	320	68%	78%	81%
2011	372	65%	77%	80%
2010	373	66%	76%	78%
2009	367	69%	79%	82%
2008	330	66%	77%	79%
2007	337	70%	80%	82%
2006	371	60%	73%	74%
2005	348	59%	76%	80%
2004	333	59%	76%	79%
2003	298	57%	76%	78%
2002	307	60%	76%	77%
2001	349	58%	71%	75%
2000	358	57%	72%	75%
1999	331	52%	68%	73%
1998	338	49%	66%	70%
1997	315	46%	67%	72%
1996	357	45%	63%	68%

1995	352	47%	66%	70%
1994	301	46%	62%	66%
1993	327	45%	63%	65%
1992	310	48%	64%	68%
1991	293	47%	65%	66%
1990	282	32%	50%	56%
1989	305	42%	61%	66%
1988	311	42%	61%	63%
1987	313	40%	67%	69%
1986	322	33%	58%	65%
1985	300	36%	56%	63%
1984	244	33%	55%	63%
1983	297	31%	52%	58%
1982	242	28%	47%	54%

b. With tens of thousands of available packages, there are many packages in `R` with similar goals and often one should experiment with multiple packages to see which package best achieves one's objectives. For display tables, `gt` is the new kid on the block and is `tidyverse` adjacent. A more mature competitor is `kableextra` (in conjunction with `kable()` from `knitr`).

Here's some documentation on `kableExtra` (the first two links are most relevant). Create the Reed graduation rates table but this time use `kableExtra`.

c. Compare and contrast your `gt` and `kableExtra` tables. Which do you think produces the better display table? Justify your answer.