CWRU DSCI351-351m-451: Lab Exercise LE3

Normal Approximation, GGPlot, Functions, Data I/O

R. H. French, L. S. Bruckman, P. Leu, K. Davis, S. Cirlos

29 September, 2022

Contents

3.0.1 LE3, 10 points.

Summary of points (use Cntrl + Shift + O for seeing sub-questions easily):-

Coding style: 1 point

- LE3-1: 0.5 point
- LE3-2: 1 point
- LE3-3: 1 point
- LE3-4: 2.5 points
- LE3-5: 2 points
- LE3-6: 2 points

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
```

```
## v readr   2.1.2      v forcats 0.5.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

3.0.1.1 Lab Exercise (LE) 3

3.1 LE3-1. Normal Approximation

According to the Health and Nutrition Examination Survey, the average height of women age 18- 24 is about 64.3 inches.

The standard deviation is about 2.6 inches.

Using the normal curve, estimate the percentage of women with heights

```
percentage_multiplier <- 100
average_height <- 64.3
sd_height <- 2.6
intervalOne <- 60
intervalTwo <- 66
increment <- 1
q1a <- pnorm(intervalTwo, mean = average_height, sd = sd_height, lower.tail = TRUE)
q1a * percentage_multiplier
```

```
## [1] 74.33945
```

```
q1 <- pnorm(intervalOne, mean = average_height, sd = sd_height, lower.tail = TRUE)

q1b <- q1a - q1
q1b * percentage_multiplier
```

```
## [1] 69.43151
```

(a) below 66 inches

ANSWER-> 74.34%

(b) between 60 and 66 inches.

ANSWER-> 69.43%

```
q1LowerPercentile <- 0.1
q1HigherPercentile <- 0.9

q1c <- qnorm(q1LowerPercentile, mean = average_height, sd = sd_height, lower.tail = TRUE)
q1d <- qnorm(q1HigherPercentile, mean = average_height, sd = sd_height, lower.tail = TRUE)
q1c
```

```
## [1] 60.96797
```

```
q1d
```

```
## [1] 67.63203
```

(c) What height is the 10th percentile?

ANSWER-> 60.97 inches

(d) What height is the 90th percentile?

ANSWER-> 67.63 inches

3.2 LE3-2. Normal Approximation for Probability

A coin is tossed 100 times.

Coin tosses follow what is called a binomial distribution (page 149 in OpenStats).

(a) What is the expected number of times that the coin comes up heads?

```
numTosses <- 100
probabilityOfOneSide <- 0.5
expNumHeads <- numTosses * probabilityOfOneSide
expNumHeads
```

```
## [1] 50
```

ANSWER-> 50

(b) What is the standard deviation for the number of times the coin comes up heads

```
varNumHeads <- numTosses * probabilityOfOneSide * (1 - probabilityOfOneSide)
sdNumHeads <- sqrt(varNumHeads)
sdNumHeads
```

```
## [1] 5
```

ANSWER-> 5

(c) If we were to consider a probability histogram for the number of times the coin came up heads, this could be approximated by the normal approximation.

Describe why we could make this approximation.

ANSWER-> `n * p = n * q = numTosses * probabilityOfOneSide = numTosses * (1 - probabilityOfOneSide) = 50 > 5`. Thus, we can make a normal approximation.

(d) Use the normal approximation to estimate the chance of getting exactly 50 heads. Hint: you can find the percentage between 50.5 and 49.5.

```
exactly50Heads <- pnorm(50.5, mean = expNumHeads, sd = sdNumHeads
                        ) - pnorm(49.5, mean = expNumHeads, sd = sdNumHeads)
exactly50Heads
```

```
## [1] 0.07965567
```

ANSWER <- 0.08

(e) Use the normal approximation to estimate the chance of getting between 45 and 55 heads inclusive.

```
between45And55Inc <- pnorm(55, mean = expNumHeads, sd = sdNumHeads
                           ) - pnorm(45, mean = expNumHeads, sd = sdNumHeads)
between45And55Inc
```

```
## [1] 0.6826895
```

ANSWER <- 0.68

(f) Use the normal approximation to estimate the chance of getting between 45 and 55 heads exclusive

```
between45And55Exc <- pnorm(54, mean = expNumHeads, sd = sdNumHeads
                           ) - pnorm(46, mean = expNumHeads, sd = sdNumHeads)
between45And55Exc
```

```
## [1] 0.5762892
```

ANSWER <- 0.58

3.3 LE3-3. Normal Approximation for Probability

We replace the 50/50 coin with a weighted coin

- that comes out to be heads 10% of the time and tails 90% of the time.

We toss it 100 times.

(a) What is the expected number of times that the coin comes up heads?

```
weightedProbOfHeads <- 0.1
wExpNumHeads <- numTosses * weightedProbOfHeads
wExpNumHeads
```

```
## [1] 10
```

ANSWER <- 10

(b) What is the standard deviation for the number of times the coin comes up heads

```
wVarNumHeads <- numTosses * weightedProbOfHeads * (1 - weightedProbOfHeads)
wSdNumHeads <- sqrt(varNumHeads)
wSdNumHeads
```

```
## [1] 5
```

ANSWER-> 5

(c) Can we still use the normal approximation in this case?

ANSWER <- n * p = numTosses * weightedProbOfHeads = 10 > 5$$; $$n * q = numTosses * (1 - weightedProbOfHeads) = 90 > 5 Thus, we can still use normal approximation

(d) Estimate the chance of getting exactly 10 heads.

```
exactly10Heads <- pnorm(10.5, mean = wExpNumHeads, sd = wSdNumHeads
                        ) - pnorm(9.5, mean = wExpNumHeads, sd = wSdNumHeads)
exactly10Heads
```

```
## [1] 0.07965567
```

ANSWER <- 0.08

3.4 LE3-4. Normal Approximation for Data Analysis

A list of exam scores is provided.

- Read these scores in as a data frame.
- Rename the column to `scores`.

For the exam scores, calculate

- the mean and
- standard deviation

```
# read in the dataset
filepath <- "data/exam_scores.csv"
exams <- read.table(file = filepath)
# rename the variable to 'scores'
scoresStr <- "scores"
names(exams) = c(scoresStr)

meanScores <- mean(exams$scores)
sdScores <- sd(exams$scores)
meanScores
```

```
## [1] 72.2
```

```
sdScores
```

```
## [1] 11.5966
```

(a) What is the mean of the exam data?

ANSWER-> 72.2

(b) What is the standard deviation of the exam data?

ANSWER-> 11.6

(c) The first student scored an 84. What percentage of students are equal to or below this score?

```
firstStudentScore <- exams$scores[1]
leFirstStudentScore <- pnorm(firstStudentScore, meanScores, sdScores, lower.tail = TRUE)
leFirstStudentScore * percentage_multiplier
```

```
## [1] 84.55516
```

ANSWER-> 84.56

(d) Now use the normal approximation.

Use the pnorm function to determine what percentage of students would be expected to be below this score?

```
ltFirstStudentScore <- pnorm((firstStudentScore + 1), meanScores, sdScores, lower.tail = TRUE)
ltFirstStudentScore * percentage_multiplier
```

```
## [1] 86.51539
```

ANSWER-> 86.52

(e) Plot a normalized histogram using ggplot.

- Use `geom_histogram(aes(y = ..density..))` in gg plot to plot a normalized histogram.

Plot a normalized Gaussian curve on top for comparison.

- Use `stat_function(fun = dnorm)` with appropriate arguments

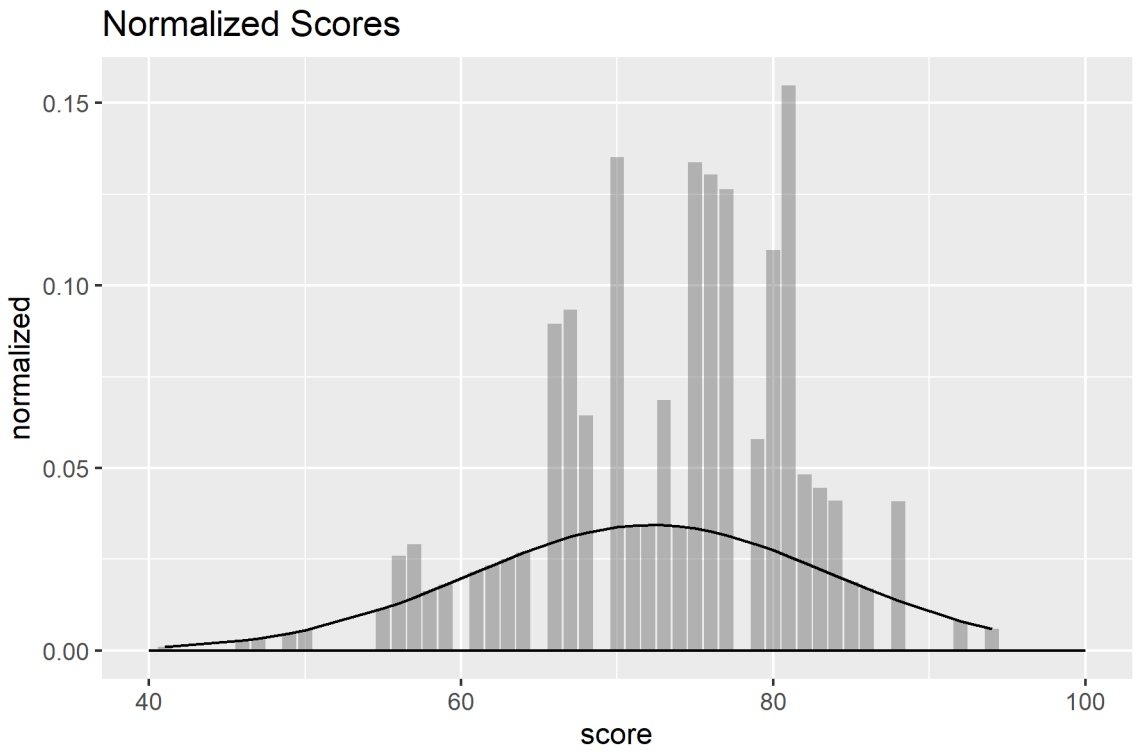
```
## Input: vector of scores data, desired mean, desired sd
## Output: data frame of scores data with corresponding normalized data
NormalFrame <- function(scoresVector, mean, sd) {
  normData <- dnorm(scoresVector, mean, sd)
  frameData <- data.frame(score = scoresVector, normalized = normData)
  return(frameData)
}
```

```r
## Input: NormalFrame data frame
## Output: histogram + gaussian curve of norm data
NormalPlot <- function(scoreData) {
  scoreplot <- ggplot(data = scoreData, mapping = aes(x = score, y = normalized)
                      ) + geom_histogram(alpha = 0.4, stat ='identity'
                      ) + labs(title = "Normalized Scores") + xlim(40, 100
                      ) + stat_function(data = scoreData, fun = dnorm) + geom_line()
  return(scoreplot)
}

library(ggplot2)
standardData <- NormalFrame(exams$scores, mean = meanScores, sd = sdScores)
standardPlot <- NormalPlot(standardData)
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```

```r
standardPlot
```



(f) You want to curve or renormalize the scores

- so that the mean is 77 and
- the standard deviation is 10.

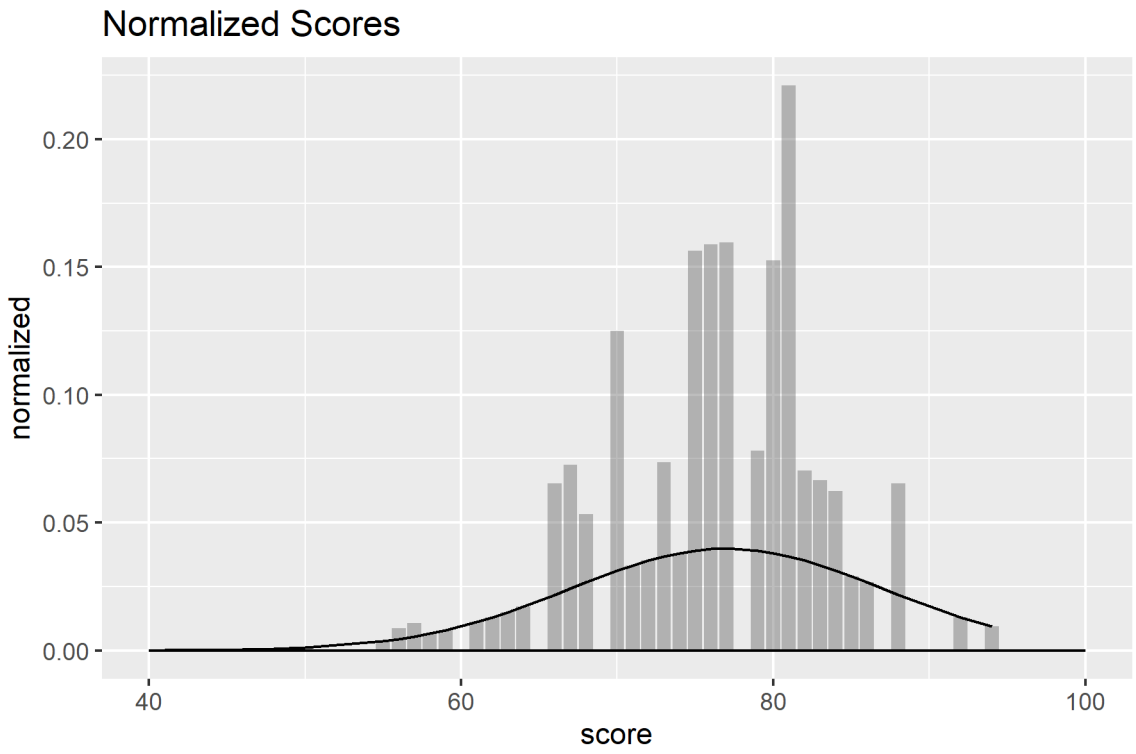
Create a new column in the data frame with the `curved_scores` which are rounded to the nearest integer.

Create a new ggplot with a histogram with the curved scores.

```
meanCurve <- 77
sdCurve <-  10
curveData <- NormalFrame(exams$scores, mean = meanCurve, sd = sdCurve)
curvePlot <- NormalPlot(curveData)
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```

```
curvePlot
```



3.5 LE3-5. Text Mining of Song Lyrics:

In LE2-3c-d, you created word clouds for Elton John and Eminem. As in LE2, the dataset for this assignment is a collection of the information and lyrics from every top 100 billboard song since 1965. Let's modify that code so that it is more flexible and extensible. The modified code will work with

- an arbitrary list of artists `artists_select` and
- an arbitrary number of `max_wordcloud_words`

```
library(tidytext)
library(tm)  # the Text Mining Package
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(NLP) # the Natural Language Processing package
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(magrittr)
```

```
##
## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':
##
##     set_names

## The following object is masked from 'package:tidyr':
##
##     extract
```

```
library(dplyr)
# load in the dataset
billboard_df <- read.csv('./data/billboard_lyrics_1964-2015.csv') %>%
  as.data.frame()
```

3.5.1 LE3-5a. Creating Extensible and Flexible Code

Write a function, named `GenerateWordCloud` that creates a wordcloud

- for each artist in an arbitrarily chosen list of artists with `max_wordcloud_words`
- So this function needs to work for 1, 2, 3 or more artists
- The word cloud should not include any `stop_words` as in LE#2
- Write your code below for the `GenerateWordCloud` function

```
data("stop_words")
# Use VCorpus(VectorSource(word) in wordcloud to eliminate warnings
GenerateWordCloud <-
  function(artists_select, billboard_df, max_wordcloud_words) {
    # This function generates word clouds for each artist in the list artists_select
    # with the maximum number of words max_wordcloud_words
    #
    # Write your code here for the GenerateWordCloud function
    #
    for(artist in artists_select) {
      billboard_artist <- billboard_df[billboard_df$Artist == artist, ]$Lyrics
      row.names(billboard_artist) <- NULL
      artist_words <- tibble(text = billboard_artist)
      artist_words <- artist_words %>% unnest_tokens(word, text) %>% anti_join(stop_words)
```

```
    artist_words_df <- as.data.frame(table(artist_words))
    names(artist_words_df) <- c("word", "freq")
    artist_words_df <- artist_words_df[order(artist_words_df$freq, decreasing = T), ]
    row.names(artist_words_df) <- NULL
    wordcloud_words <- artist_words_df$word[1:max_wordcloud_words]
    wordcloud_freq <- artist_words_df$freq[1:max_wordcloud_words]
    artist_wordcloud <- wordcloud(wordcloud_words, wordcloud_freq)
    artist_wordcloud
  }
}
```

3.5.2 LE3-5b Testing your function

When writing functions, it is useful to have some test scripts to make sure that the function works under a variety of test cases. Use the following test cases to make sure the code is working properly on different cases.

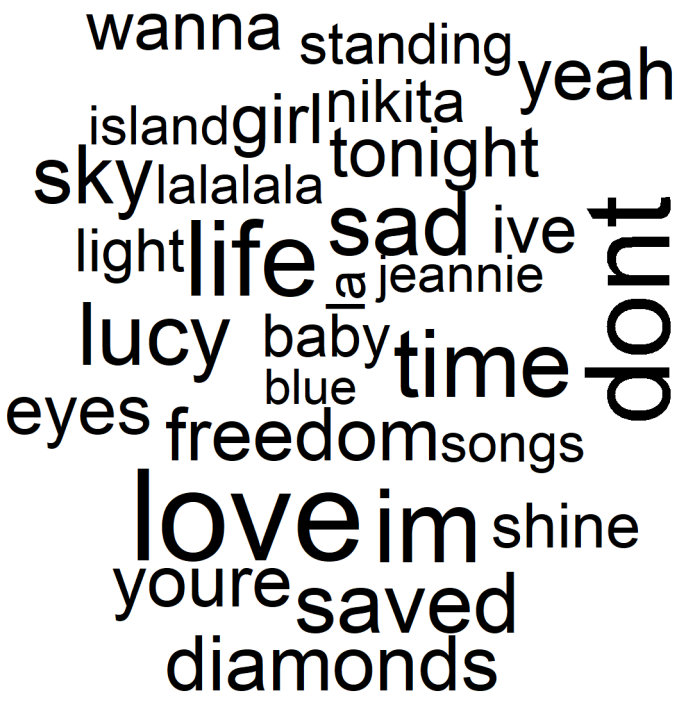
Test your function for 2 arbitrary artists.

```
max_wordcloud_words <- 30
artists_select <- c("elton john","eminem")
GenerateWordCloud(artists_select, billboard_df, max_wordcloud_words)
```

```
## Joining, by = "word"

## Warning in wordcloud(wordcloud_words, wordcloud_freq): philadelphia could not be
## fit on page. It will not be plotted.

## Joining, by = "word"
```



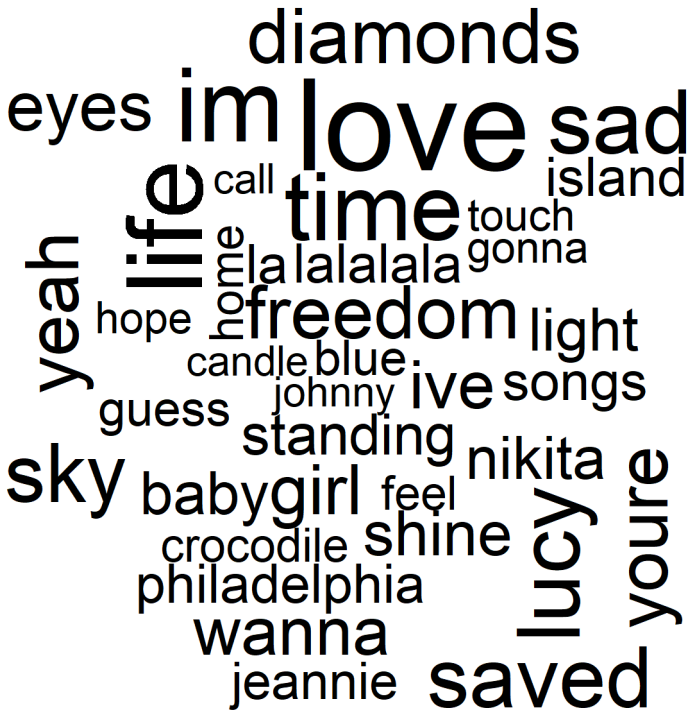
Test your function on 1 artist.

```
max_wordcloud_words <- 40
artists_select <- c("elton john")
GenerateWordCloud(artists_select, billboard_df, max_wordcloud_words)
```

```
## Joining, by = "word"
```

```
## Warning in wordcloud(wordcloud_words, wordcloud_freq): dont could not be fit on
## page. It will not be plotted.
```

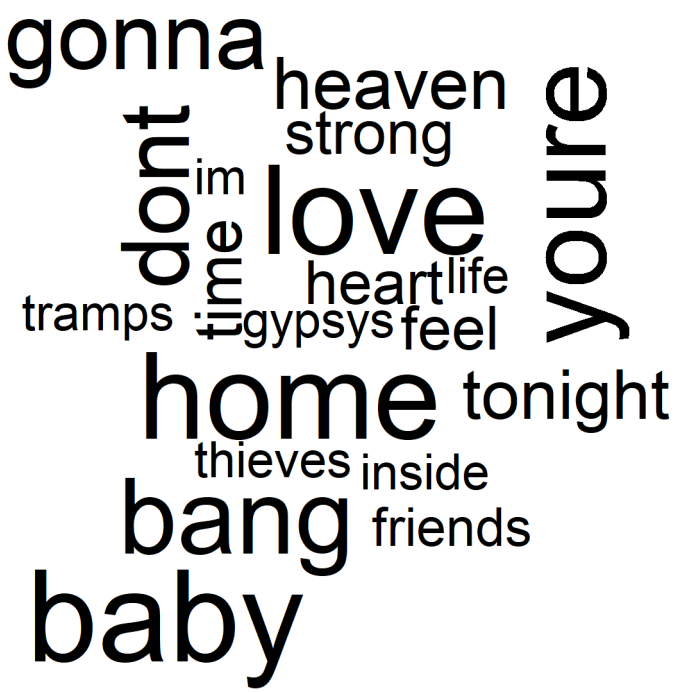
```
## Warning in wordcloud(wordcloud_words, wordcloud_freq): tonight could not be fit
## on page. It will not be plotted.
```



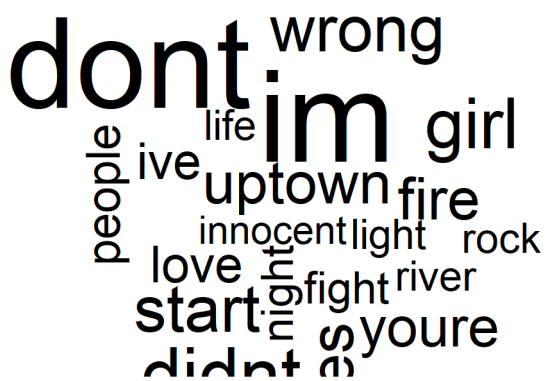
And on 3 artists.

```
max_wordcloud_words <- 20
artists_select <- c("cher", "madonna", "billy joel")
GenerateWordCloud(artists_select, billboard_df, max_wordcloud_words)
```

```
## Joining, by = "word"
## Joining, by = "word"
```




```
## Joining, by = "word"
```



3.6 LE3-6. Acrylic Hardcoats: Reading in data from csv files

You will practice reading in data and writing efficient code.

You will create a color files dataframe by going into multiple folders and concatenating data from multiple .csv files into one large data frame.

A good practice in developing code is to write out pseudocode, which serves an outline for your code. The pseudocode has been written out here for you.

Write a function that will

- Go into a particular folder.

- Get all the files in that folder using `files <- list.files(path = folder_name)`
- Read the .csv files
- Bind the .csv files together by row.
- Read in the `data_sample_key` dataframe with `data_sample_key <- read.csv('data/acryhc-key.csv')`
- The dataframes can be merged using `merge(x = data_total, y = data_sample_key, by.x = 'ID', by.y = 'Sample.Number')` as the `ID` in the color files are matched to the `Sample.Number` in `data_sample_key`
- Add an additional column to the dataframe with the `step_number`

Now use your function on all all color datafiles in the `./data/color` folder. Add a `for` loop or use `map_dfr` to apply the same function over multiple files.

Your final data frame should be `data_color_all` and have 775 observations with 10 variables.

Key: Do not duplicate code.

As you are developing your code, initially test out the function on a single file. Then, test it on a list of files in one folder. Then, test it on all folders. This will make debugging easier.

If you write your code efficiently, you should be able to code everything in only 30 lines of code or less.

```
library("dplyr")                                        # Load dplyr package
library("plyr")                                         # Load plyr package
```

```
## ------------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```
## ------------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## The following object is masked from 'package:purrr':
##
##     compact
```

```
library("readr")                                        # Load readr package

## Perform this:
## -   Go into a particular folder.
## -   read data/acryhc-key.csv as data_sample_key
## -   Get all the .csv files in that folder
## -   Read the .csv files
## -   Bind the .csv files together by row.
## -   merge row-binded df with data_sample_key obtained from on Sample.Number~ID
## -   Add step number
ColorToAcrylic <- function(folder, subfolder, data_sample_key) {
    dirSeperator <- "/"
    stepStr <- substr(subfolder, 5, 5)
    stepNum <- as.integer(stepStr)
    subfolderpath <- paste(folder, subfolder, dirSeperator,
                           sep = dirSeperator)
    files_t <- list.files(path = subfolderpath,
                          pattern = "*.csv",
                          full.names = TRUE) %>%
      lapply(read_csv) %>%
      bind_rows
    data_total <- as.data.frame(files_t)
    data_total %>% merge(x = data_total,
                         y = data_sample_key,
```

```r
                         by.x = 'ID',
                         by.y = 'Sample.Number')
    data_total$step_number <- stepNum
    return(data_total)
}

## Perform ColorToAcrylic (primary logic) on entire color folders
## return df containing all the data from the step<i> folder for i = 1..4
MapColorsToAcrylicHardcoats <- function(folder) {
  subfolders <- list.files(path = folder)
  data_sample_key <- read.csv('./data/acryhc-key.csv')
  data_color_all <- data.frame()
  for (subfolder in subfolders) {
    data_color <- ColorToAcrylic(folder, subfolder, data_sample_key)
    data_color_all <- bind_rows(data_color_all, data_color)
  }
  return(data_color_all)
}

## Execute Code
colorFolder <- './data/color/'
data_color_all <- MapColorsToAcrylicHardcoats(colorFolder)
```

```
## Rows: 30 Columns: 6

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L., a., b., YI.E313..D65.10., Haze...D65.10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 30 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 30 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 30 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 30 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 30 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L., a., b., YI.E313..D65.10., Haze...D65.10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 30 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 30 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 32 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 32 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 80 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
```

```
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 40 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 45 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 48 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 36 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 32 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 32 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 36 Columns: 6
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 32 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 47 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 36 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 63 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): ID
## dbl (5): L*, a*, b*, YI E313 [D65/10], Haze % D65/10
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
summary(data_color_all)
```

```
##       ID                  L.              a.                b.
##  Length:831         Min.   :95.76   Min.   :-0.2000   Min.   :0.3200
##  Class :character   1st Qu.:95.81   1st Qu.:-0.1800   1st Qu.:0.3500
##  Mode  :character   Median :96.28   Median :-0.1100   Median :0.6500
##                     Mean   :96.30   Mean   :-0.1065   Mean   :0.6477
##                     3rd Qu.:96.80   3rd Qu.:-0.0300   3rd Qu.:0.9500
##                     Max.   :96.86   Max.   :-0.0200   Max.   :0.9700
##                     NA's   :771     NA's   :771       NA's   :771
##  YI.E313..D65.10. Haze...D65.10         L*              a*
##  Min.   :0.5600   Min.   :1.000   Min.   :85.67   Min.   :-0.9200
##  1st Qu.:0.6175   1st Qu.:1.700   1st Qu.:95.14   1st Qu.:-0.2400
##  Median :1.1300   Median :2.100   Median :95.81   Median :-0.1700
```

```
##  Mean   :1.1333   Mean   :2.255   Mean   :95.54   Mean   :-0.1758
##  3rd Qu.:1.6500   3rd Qu.:2.625   3rd Qu.:96.55   3rd Qu.:-0.0300
##  Max.   :1.6900   Max.   :6.300   Max.   :96.88   Max.   : 0.7300
##  NA's   :771      NA's   :771     NA's   :60      NA's   :60
##        b*        YI E313 [D65/10] Haze % D65/10    step_number
##  Min.   :0.240   Min.   : 0.340   Min.   : 0.90   Min.   :0.000
##  1st Qu.:0.615   1st Qu.: 1.135   1st Qu.: 2.40   1st Qu.:0.000
##  Median :0.970   Median : 1.740   Median : 4.40   Median :1.000
##  Mean   :1.109   Mean   : 1.972   Mean   :12.17   Mean   :1.437
##  3rd Qu.:1.290   3rd Qu.: 2.310   3rd Qu.: 9.70   3rd Qu.:2.000
##  Max.   :7.850   Max.   :15.120   Max.   :97.80   Max.   :4.000
##  NA's   :60      NA's   :60       NA's   :60
```

3.6.0.1 Links

http://www.r-project.org

http://rmarkdown.rstudio.com/