Statistics 5014: Homework 5

Due Tuesday Sep 26, 9am

2018-09-26

Problem 3: What makes a good figure?

A good figure needs to tell a good story about the data in question. In order to do so, it should be concise, clear, and easy to read in a minimal amount of time. Simultaneously, it should contain sufficient information to send a few key messages, and the information should be categorized in colors, shapes, and patterns that are easily distinguishable.

Problem 4

a. Create a function that computes the proportion of successes in a vector. Use good programming practices.

```
pSuccess <- function(bin_vect){
  # Check if bin_vect is a binary vector and execute an error if not
  if( any(which(bin_vect[which(bin_vect != 1)] != 0)) ){
    stop("Input should be a binary vector")
  }
  # Calculate proportion of success
  length(which(bin_vect == 1))/length(bin_vect)
}
```

b. Create a matrix to simulate 10 flips of a coin with varying degrees of "fairness" as follows:

```
set.seed(12345)
# Original code, repeats a single vector of 10 random flips 10 times,
# and so all 10 columns are the same.
P4b_data <- matrix(rbinom(10, 1, prob = (30:40)/100), nrow = 10, ncol = 10)
```

c. Use your function in conjunction with apply to compute the proportion of success in P4b_data by column and then by row. What do you observe? What is going on?

When applying the function over the columns, the proportion of success of each column vector is returned, resulting in a vector showing the probability of each column. The same performed with the rows when applying the function over the rows. However, the given matrix `P4b_data` repeats a single vector of 10 random flips 10 times, and so all 10 columns are the same. This results in the probabilities of all the columns to be equal, as shown in `success_col`, while `success_row` only returns probabilities of 0 or 1, since each element in the same row is repeated.

```
success_col <- apply(P4b_data, 2, pSuccess) # Compute probabiilities by col
success_col
```

```
##  [1] 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6
```

```
success_row <- apply(P4b_data, 1, pSuccess) # Compute probabiilities by row
success_row
```

```
##  [1] 1 1 1 1 0 0 0 0 1 1
```

d. You are to fix the above matrix by creating a function whose input is a probability and output is a vector whose elements are the outcomes of 10 flips of a coin. Now create a vector of the desired

probabilities. Using the appropriate apply family function, create the matrix we really wanted above. Prove this has worked by using the function created in part a to compute and tabulate the appropriate marginal successes.

```
rSuccess <- function(prob_vect){
  # Check if prob_vect is a vector of probabilities and execute an error if not
  if( any(prob_vect < 0 | prob_vect > 1) ){
    stop("Vector elements are probabilities. They should range from 0 to 1.")
  }
  # Create a matrix that is dependent on the length and values of prob_vect
  # and returns a matrix of 10 randomly generated binary values
  # for each element of prob_vect
  matrix(rbinom(10*length(prob_vect), 1, prob = prob_vect),
         ncol = length(prob_vect), byrow = T)
}
# Test run of rSuccess and pSuccess
test_vect <- runif(12, 0, 1) # Randomly create a vector of probabilities
test_matrix <- rSuccess(test_vect)

# Perform Monte-Carlo simulation to confirmthe original probabilities
set.seed(8034)
test_list <- list() # Make an empty list for the simulated data
M <- 2000 # Number of simulations
# Make an empty dataset for simulated probabilities
test_prob <- data.frame(matrix(NA, nrow = M, ncol = length(test_vect)))
# Fill the list with simulated data
for(i in 1:M){
  test_list[[i]] <- rSuccess(test_vect)
  test_prob[i,] <- apply(test_list[[i]], 2, pSuccess)
}

test_results_col <- apply(test_prob, 2, mean) # Average probabilities by column
test_results_row <- apply(test_matrix, 1, mean) # Average probabilities by row

kable(as.data.frame(rbind(test_vect, test_results_col),
      row.names = c("Actual probability", "Simulated probability")),
      caption = "Marginal successes by column", digits = 2, format = "pandoc")
```

Table 1: Marginal successes by column

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12
Actual probability	0.03	0.15	0.74	0	0.39	0.46	0.39	0.4	0.18	0.95	0.45	0.33
Simulated probability	0.03	0.15	0.74	0	0.39	0.47	0.39	0.4	0.18	0.95	0.45	0.33

```
kable(as.data.frame(rbind(test_results_row),
                    row.names = c("Probability of success")),
      caption = "Marginal successes by row", digits = 2, format = "pandoc")
```

Table 2: Marginal successes by row

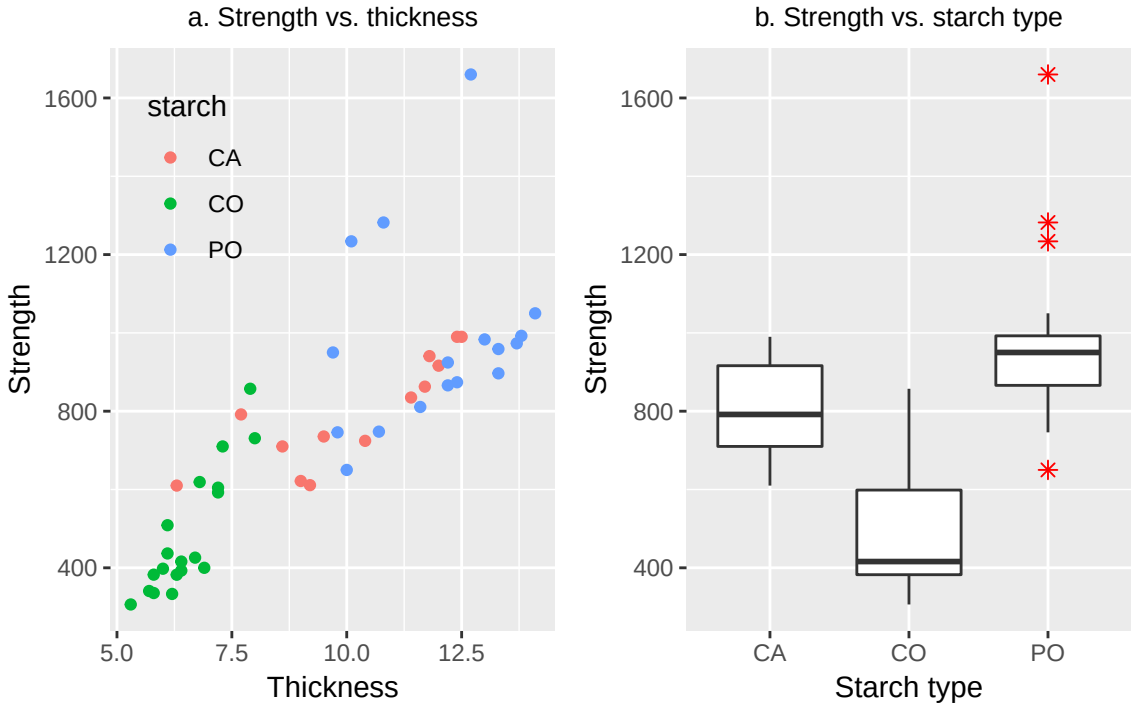
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
Probability of success	0.42	0.33	0.42	0.58	0.5	0.42	0.42	0.5	0.58	0.42

Problem 5

Overall, there is a positive relationship between strength and thickness. Among the different starch types, there is some overlap in the distributions of strength and thickness. The median strength of the PO type starch appears to be the greatest, and a few outliers can be observed for the group.

```
# Store URL for starch data
url <- "http://www2.isye.gatech.edu/~jeffwu/book/data/starch.dat"
starch_data <- fread(url, header = T, fill = T, sep = " ") # Import data
# Scatterplot of numeric data
starch_plot1 <- ggplot(starch_data, aes(x = thickness, y = strength,
                                        col = starch)) + geom_point() +
  theme(plot.title = element_text(size = 10, hjust = 0.5),
        legend.justification=c(0,1), legend.position=c(0.05, 0.95),
        legend.background = element_blank(), legend.key = element_blank()) +
  labs(title = "a. Strength vs. thickness", x = "Thickness", y = "Strength")
# Box plot of strength
starch_plot2 <- ggplot(starch_data, aes(x = starch, y = strength)) +
  geom_boxplot(outlier.colour="red", outlier.shape = 8, outlier.size = 2) +
  labs(title = "b. Strength vs. starch type", x = "Starch type",
       y = "Strength") +
  theme(plot.title = element_text(size = 10, hjust = 0.5))
grid.arrange(starch_plot1, starch_plot2, nrow = 1,
             top = "Figure 1. Exploratory analysis of starch data")
```

Figure 1. Exploratory analysis of starch data



Problem 6: Create an annotated map of the US

Part a. Get and import a database of US cities and states.

```
#we are grabbing a SQL set from here
# http://www.farinspace.com/wp-content/uploads/us_cities_and_states.zip
#download the files, looks like it is a .zip
library(downloader)
download("http://www.farinspace.com/wp-content/uploads/us_cities_and_states.zip", dest = "us_cities_sta
unzip("us_cities_states.zip", exdir="./")
#read in data, looks like sql dump, blah
states <- fread(input = "./us_cities_and_states/states.sql", skip = 23, sep = "'",
                sep2 = ",", header = F, select = c(2,4))
cities <- fread(input = "./us_cities_and_states/cities.sql", sep = "'",
                sep2 = ",", header = F, select = c(2,4))
colnames(states) <- c("state", "abbr")
colnames(cities) <- c("city", "abbr")
states_50 <- states[states$abbr != "DC"] # Limit states to the 50
```

Part b. Create a summary table of the number of cities included by state.

```
# Remove rows that have states not listed in the states dataset
uscities <- subset(cities, abbr %in% states_50$abbr)
table(uscities$abbr) # View summary table
```

```
##
##   AK   AL   AR   AZ   CA   CO   CT   DE   FL   GA   HI   IA   ID   IL   IN
##  229  579  605  264 1239  400  269   57  524  629   92  937  266 1287  738
##   KS   KY   LA   MA   MD   ME   MI   MN   MO   MS   MT   NC   ND   NE   NH
##  634  803  479  511  430  461  885  810  942  440  360  762  373  528  255
##   NJ   NM   NV   NY   OH   OK   OR   PA   RI   SC   SD   TN   TX   UT   VA
##  579  346   99 1612 1069  585  379 1802   70  377  364  548 1466  250  839
##   VT   WA   WI   WV   WY
##  288  493  753  753  176
```

```
citynum <- data.frame(table(uscities$abbr))
colnames(citynum) <- c("abbr", "num")
states_50 <- cbind(states_50, num = citynum$num)
```

Part c. Create a function that counts the number of occurrences of a letter in a string. The input to the function should be "letter" and "state_name". The output should be a scalar with the count for that letter.

Create a for loop to loop through the state names imported in part a. Inside the for loop, use an apply family function to iterate across a vector of letters and collect the occurrence count as a vector.

```
# Function that counts the number of letters in a string
getCount <- function(letter, state_name){
  state_lower <- tolower(state_name)
  count <- sum(strsplit(state_lower, "")[[1]] == letter)
  return(count)
}

# Collect the occurrence count of state names as a matrix
letter_count <- data.frame(matrix(NA, nrow = 50, ncol = 26))

for(i in 1:50){
  letter_count[i,] <- sapply(letters, getCount, state_name = states_50$state[i])
```

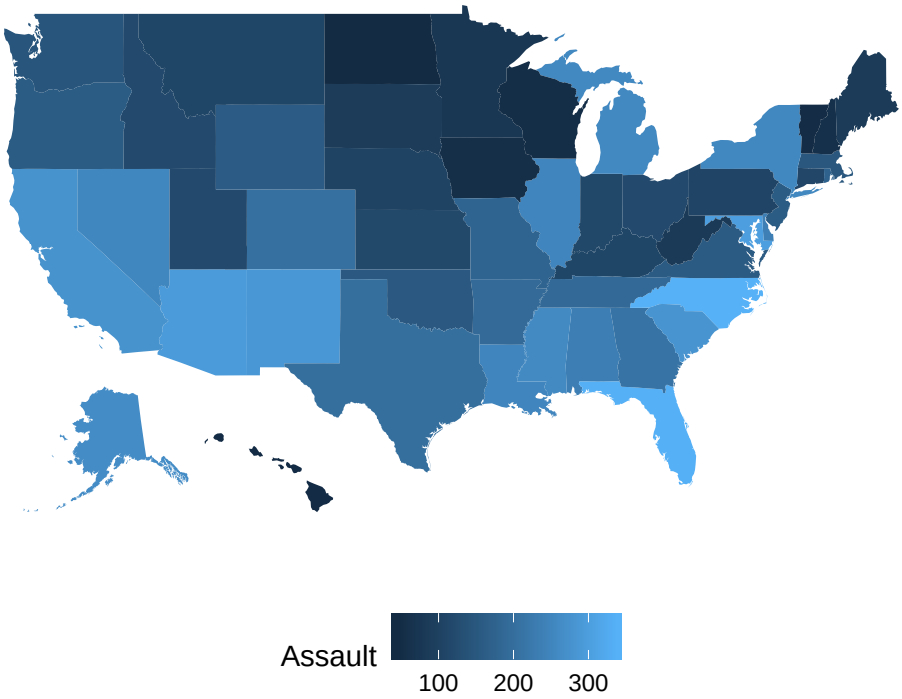
```
}
```

Part d.

Create 2 maps to finalize this. Map 1 should be colored by count of cities on our list within the state. Map 2 should highlight only those states that have more than 3 occurrences of ANY letter in their name.

Quick and not so dirty map:

```
#https://cran.r-project.org/web/packages/fiftystater/vignettes/fiftystater.html
library(fiftystater)
data("fifty_states") # this line is optional due to lazy data loading
crimes <- data.frame(state = tolower(rownames(USArrests)), USArrests)
# map_id creates the aesthetic mapping to the state name column in your data
p1 <- ggplot(crimes, aes(map_id = state)) +
  # map points to the fifty_states shape data
  geom_map(aes(fill = Assault), map = fifty_states) +
  expand_limits(x = fifty_states$long, y = fifty_states$lat) +
  coord_map() +
  scale_x_continuous(breaks = NULL) +
  scale_y_continuous(breaks = NULL) +
  labs(x = "", y = "") +
  theme(legend.position = "bottom",
  panel.background = element_blank())
p1
```




```
# Map states with more than 3 of the same letter
states_50$state <- tolower(states_50$state)
state_3lett <- states_50[apply(letter_count, 1, max) > 3]

p2 <- ggplot(state_3lett, aes(map_id = state)) +
  geom_map(aes(fill = num), map = fifty_states) +
    expand_limits(x = fifty_states$long, y = fifty_states$lat) +
```

```
  coord_map() +
  scale_x_continuous(breaks = NULL) +
  scale_y_continuous(breaks = NULL) +
  labs(x = "", y = "") +
  theme(legend.position = "bottom", panel.background = element_blank())
p2
```