HW6_Brown_Nick

Nick Brown

10/14/2019

```
knitr::opts_chunk$set(echo = TRUE, include=TRUE, eval=TRUE)
library(knitr)
library(ggplot2)
```

Problem 1 I completed the Swirl lesson parts 10, 11, and 12.

Problem 2 Homework is saved as a pdf in accordance with naming convention study

Problem 3

a. Create a function that computes the proportion of successes in a vector

A matrix of this sort will have at least least two vectors: name and success. A success will be denoted as "1" and a non-success will be "0".

A function that computes the proporition of successes in a vector would be simply the mean of the success column. Whatever the mean, then this is the proportion of success.

Thus, we can use a mean function as follows:

```
teams <- c("Redskins", "Cowboys", "Eagles", "Patriots", "Dolphins", "Chiefs")
sunday_win <- c(1,0,0,1,0,1)
success_df <- data.frame(teams,sunday_win)

victory_calc <- function(winning) {
  prop_of_success <- mean(winning)
  return(prop_of_success)
}
victory_calc(sunday_win)
```

```
## [1] 0.5
```

As noted, the mean of the success column will represent the proportion of success in a given numeric vector.

b. Create a matrix to simulate 10 flips of a coin with varying degrees of "fairness" (columns = probability) as follows:

```
set.seed(12345)
P4b_data <- matrix(rbinom(10, 1, prob = (30:40)/100), nrow = 10, ncol = 10, byrow = FALSE)
```

c. Use your function in conjunction with apply to compute the proportion of success in P4b_data by column and then by row.

```
column_successes <- apply(P4b_data, 2, victory_calc)
row_successes <- apply(P4b_data, 1, victory_calc)

column_successes
```

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

```
row_successes
```

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

What do you observe?

The rows and columns have different success probabilities. Every column has a success proportion of 0.6, but the row vectors have 6 rows where success is guaranteed and 4 rows where defeat is expected.

What is going on?

The coin flips are systematic and identical; therefore, the success rates are the same in every column. When we calculate across rows, there is not a variation in outcomes, and as a result, the row successes will be completely binary, whereas the column successes will be a proportion between 0 and 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.

```
new_outcomes <- function(success_prob) {
  success_vector <-t(rbinom(10, 1, prob = success_prob))
  return(success_vector)
}

blank_matrix <- matrix(nrow = 10, ncol = 10)
vector_of_success_probs <- c(.1,.2,.3,.4,.5,.1,.2,.3,.4,.5)

new_matrix <- apply(t(vector_of_success_probs), 2, new_outcomes)

new_matrix
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    0    1    1    1    0    0    1    1     0
##  [2,]    0    0    0    0    1    0    0    0    1     1
##  [3,]    0    1    0    1    0    1    0    0    0     1
##  [4,]    0    0    0    1    0    0    0    0    1     1
##  [5,]    0    0    0    0    1    1    0    0    1     1
##  [6,]    0    0    0    0    0    0    0    0    0     1
##  [7,]    0    0    1    0    1    1    1    1    1     1
##  [8,]    0    0    1    0    0    0    0    0    1     1
##  [9,]    0    0    0    0    0    0    0    1    0     0
## [10,]    1    0    0    1    0    1    0    0    0     0
```

The new matrix is outputted above and it shows the variability occurring within each vector across columns and across rows. This is the randomness we were seeking when performing 10 flips with random probabilities in a given set, then doing these sets 10 times for a total of 100 flips.

Prove this has worked by using the function created in part a to compute and tabulate the appropriate marginal successes.

```
new_column_successes <- apply(new_matrix, 2, victory_calc)
new_row_successes <- apply(new_matrix, 1, victory_calc)

new_column_successes
```

```
##  [1] 0.1 0.1 0.3 0.4 0.4 0.4 0.1 0.3 0.6 0.7
```

```
new_row_successes
```

```
##  [1] 0.5 0.3 0.4 0.3 0.4 0.1 0.7 0.3 0.1 0.3
```

Across margins, we can see the different success rates. This indicates that the function created works appropriately in demonstrating the differences in marginal successes across columns and rows.

Problem 4

In Homework 4, we had a dataset we were to compute some summary statistics from. The description of the data was given as "a dataset which has multiple repeated measurements from two devices by thirteen Observers". Where the device measurements were in columns "dev1" and "dev2". Reimport that dataset, change the names of "dev1" and "dev2" to x and y and do the following:

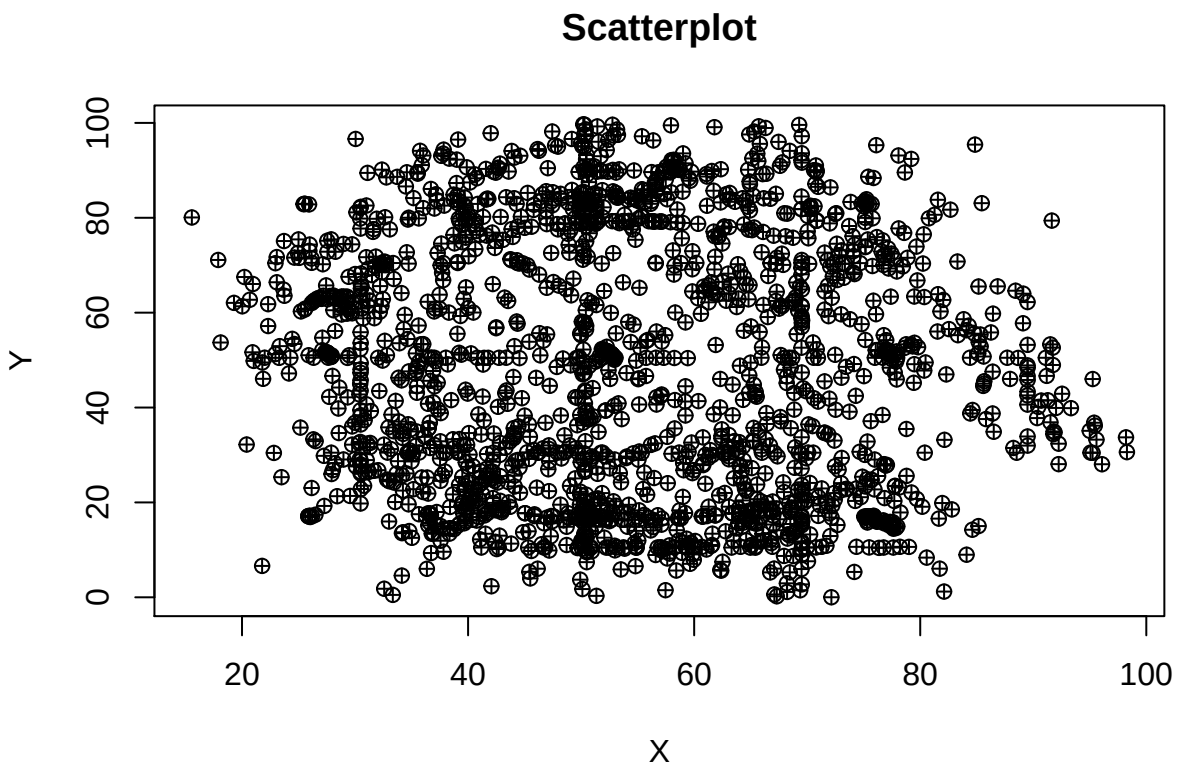
```
  HW6_data <- readRDS("HW4_data.rds")
  colnames(HW6_data) <- c("Observer","x","y")
  head(HW6_data)
```

```
##   Observer       x       y
## 1        4 55.3846 97.1795
## 2        4 51.5385 96.0256
## 3        4 46.1538 94.4872
## 4        4 42.8205 91.4103
## 5        4 40.7692 88.3333
## 6        4 38.7179 84.8718
```

The homework 4 dataset was imported as a homework 6 dataset, and the column names were changed from "dev1" and "dev2" to x and y, respectively.

1. create a function that accepts a dataframe of values, title, and x/y labels and creates a scatter plot

```
overall_scatterplot <- function(df) {
  attach(df)
  all_scatterplot <- plot(x, y,main = "Scatterplot", xlab = "X", ylab = "Y", pch=10)
  return(all_scatterplot)
}

overall_scatterplot(HW6_data)
```



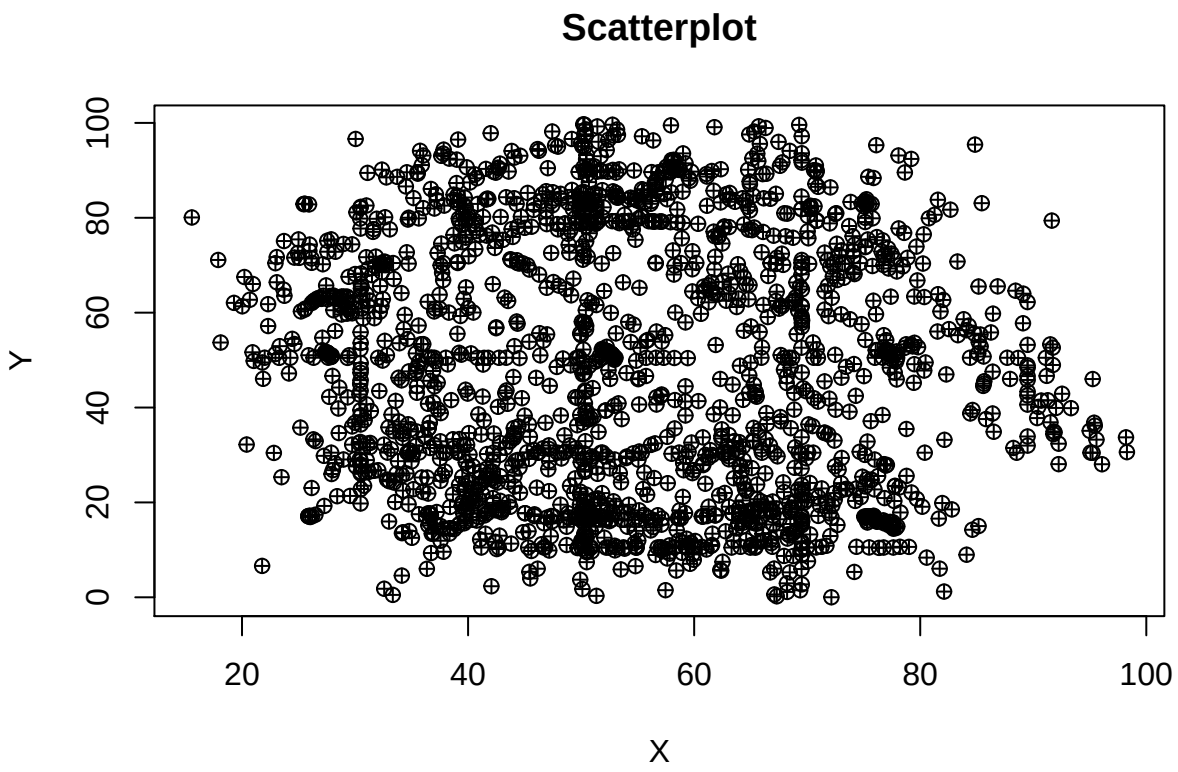

```
## NULL
```

2. use this function to create:

(a) a single scatter plot of the entire dataset

```
overall_scatterplot(HW6_data)
```

```
## The following objects are masked from df (pos = 3):
##
##     Observer, x, y
```

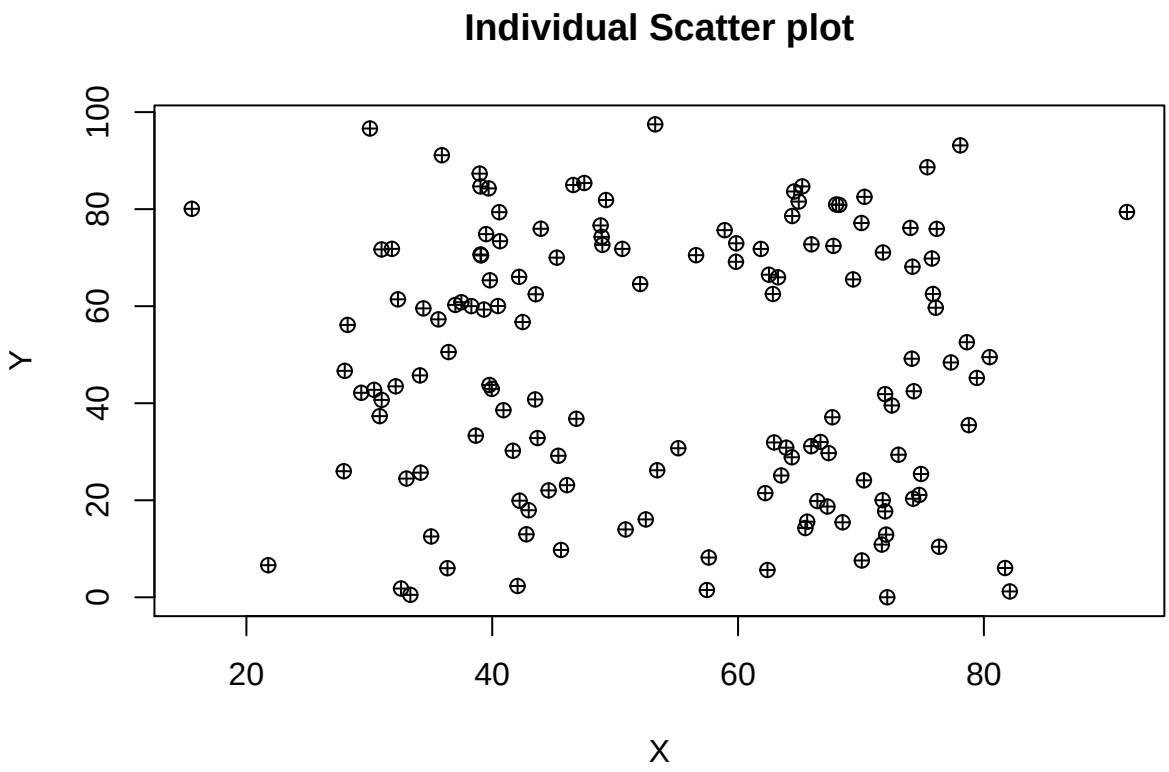


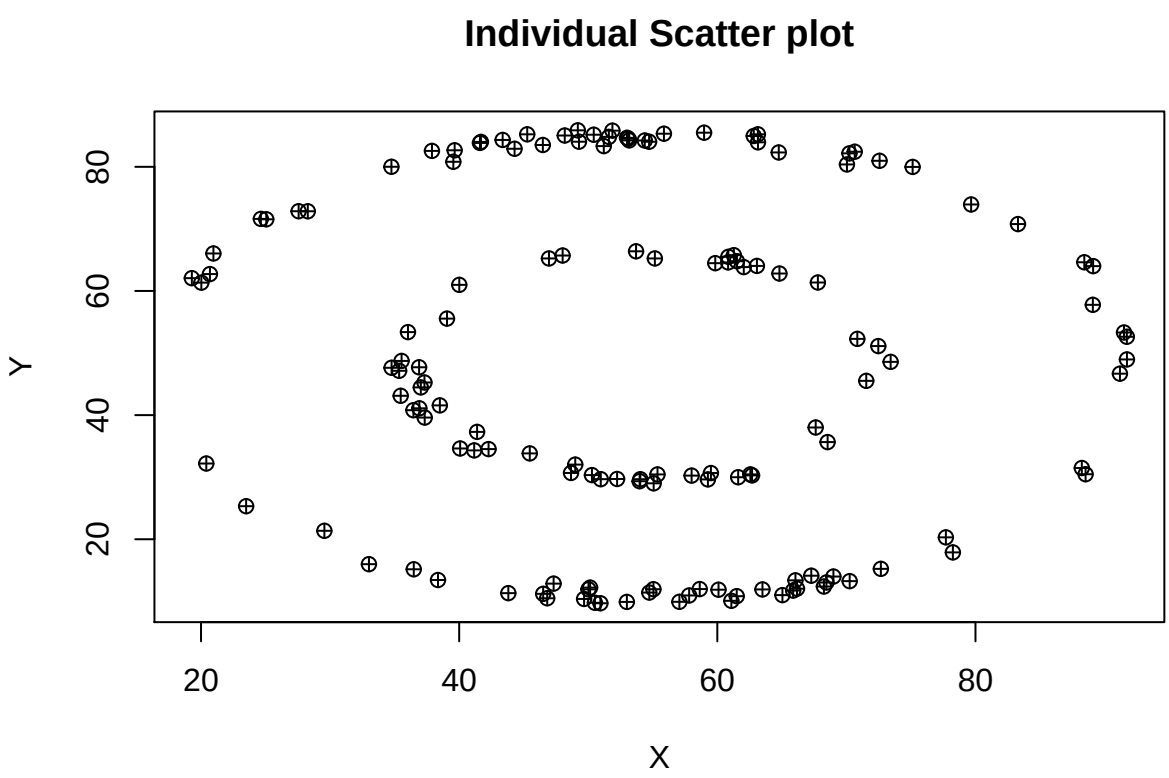

```
## NULL
```

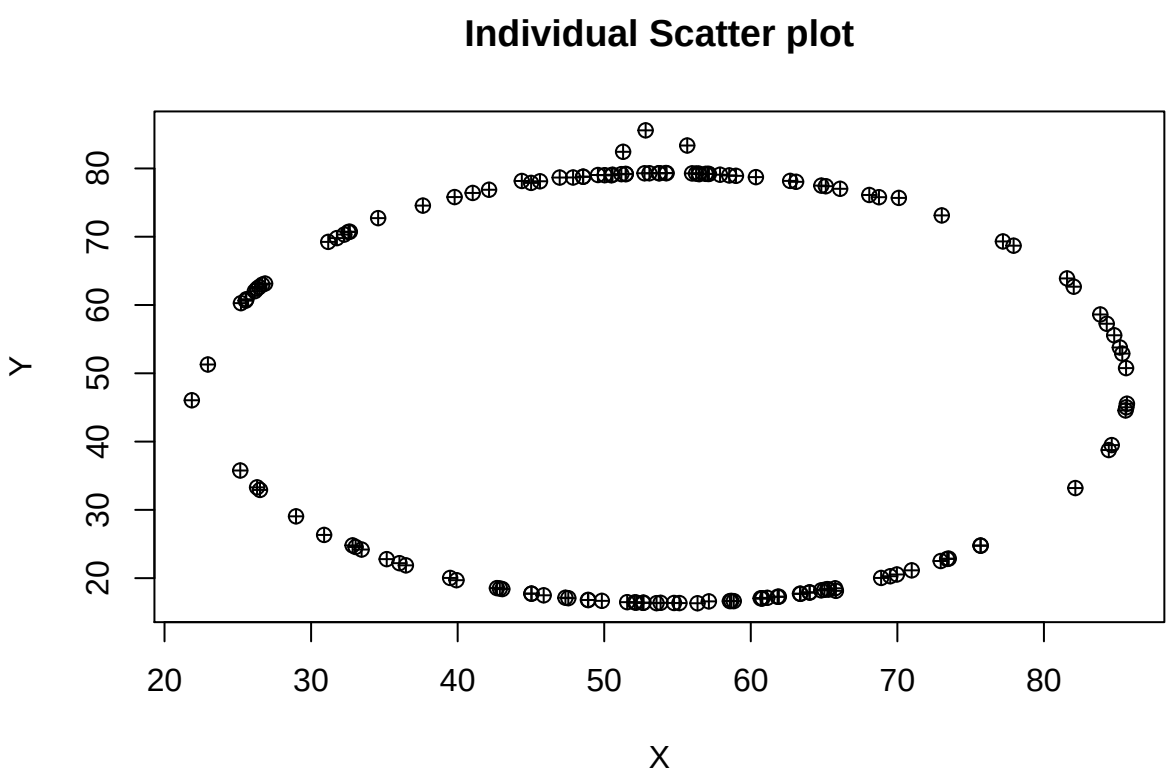
The overall scatter plot of the entire dataset has been created and can be seen above.

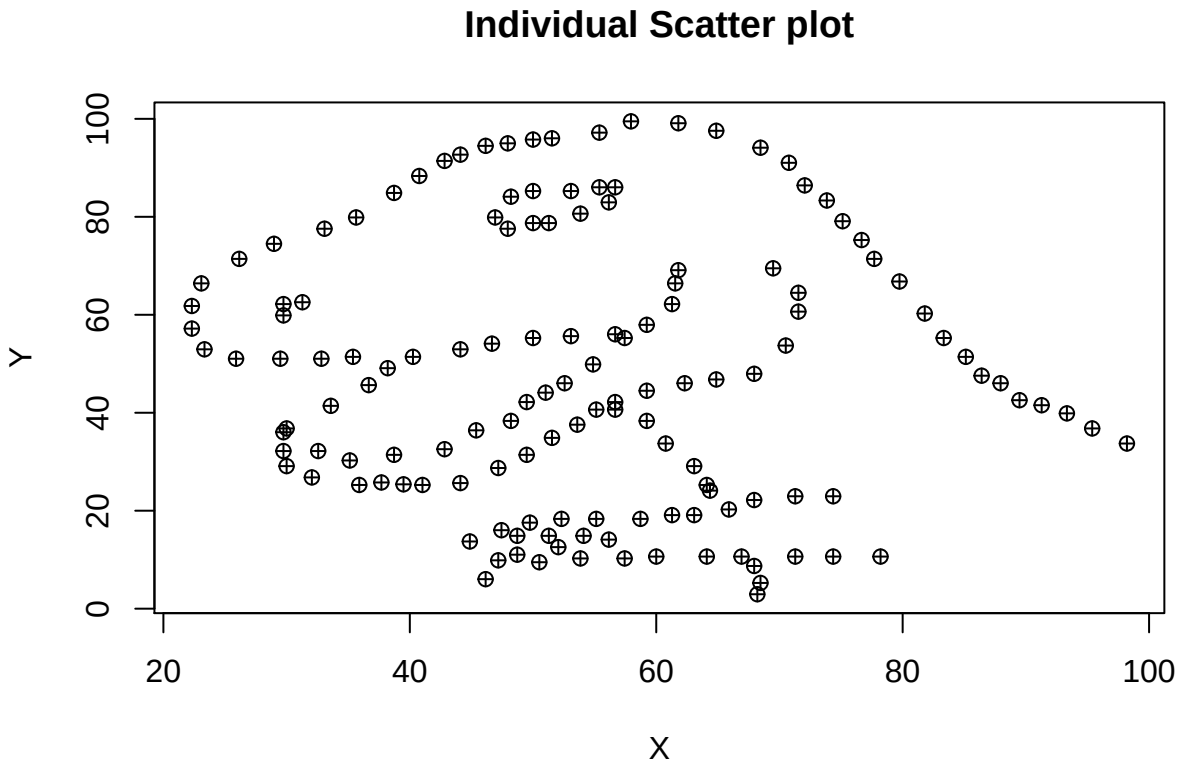
(b) a seperate scatter plot for each observer (using the apply function)

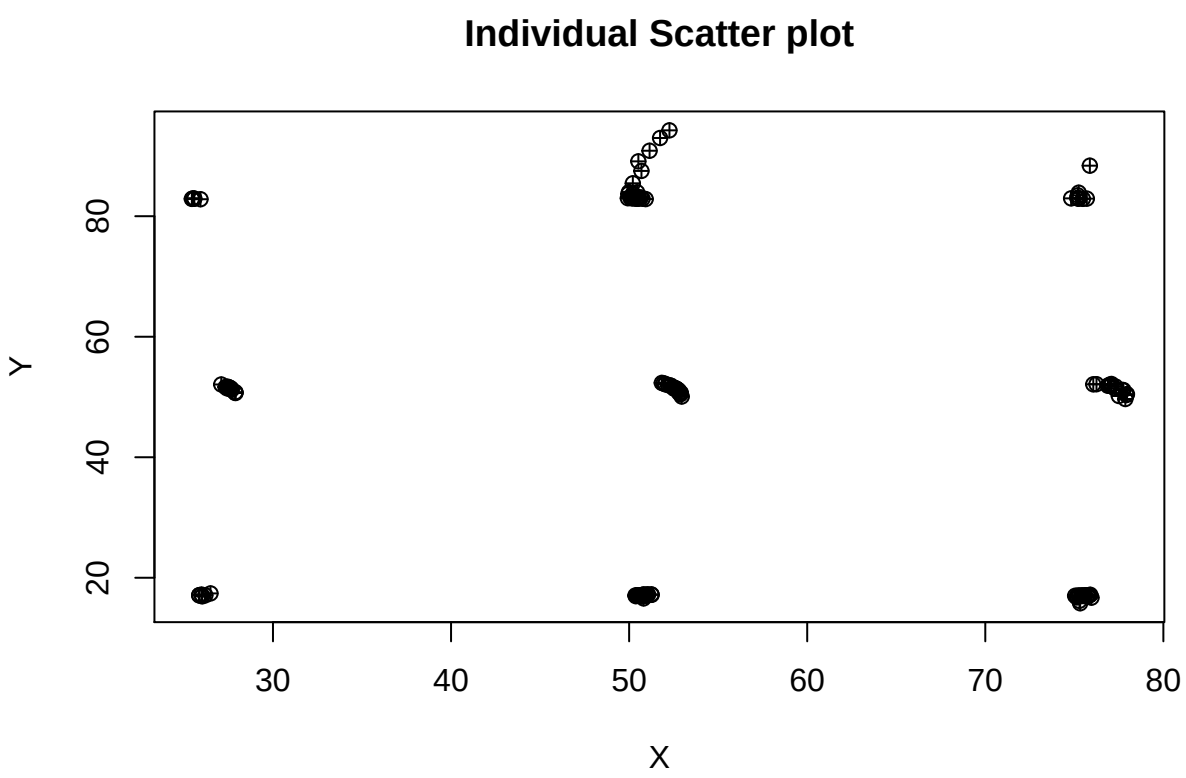
```
individual_scatterplot <- function(obs_input,df) {
  all_scatterplot <- with(df[(df$Observer == obs_input),], plot(x, y,main = "Individual Scatter plot", i
  return(all_scatterplot)
}

lapply(c(1:13), individual_scatterplot,df=HW6_data)
```

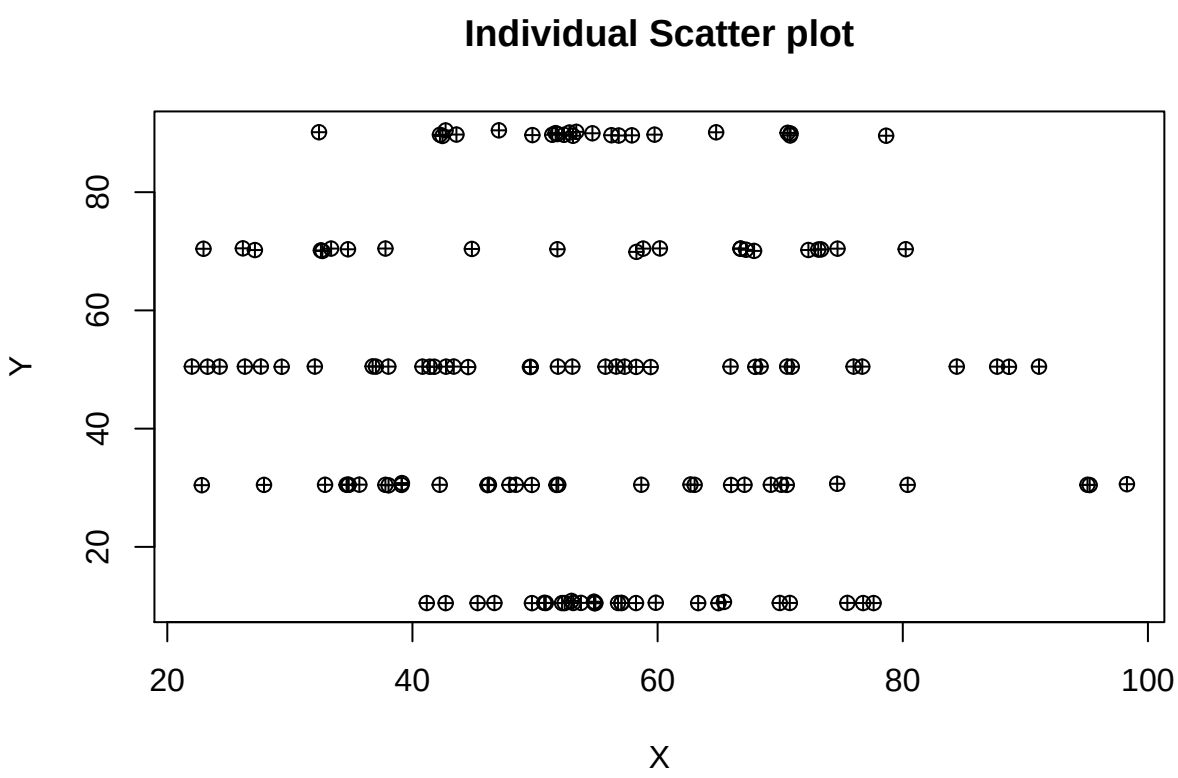

Individual Scatter plot
Y
0
20
40
60
80
100
20
40
60
80
X

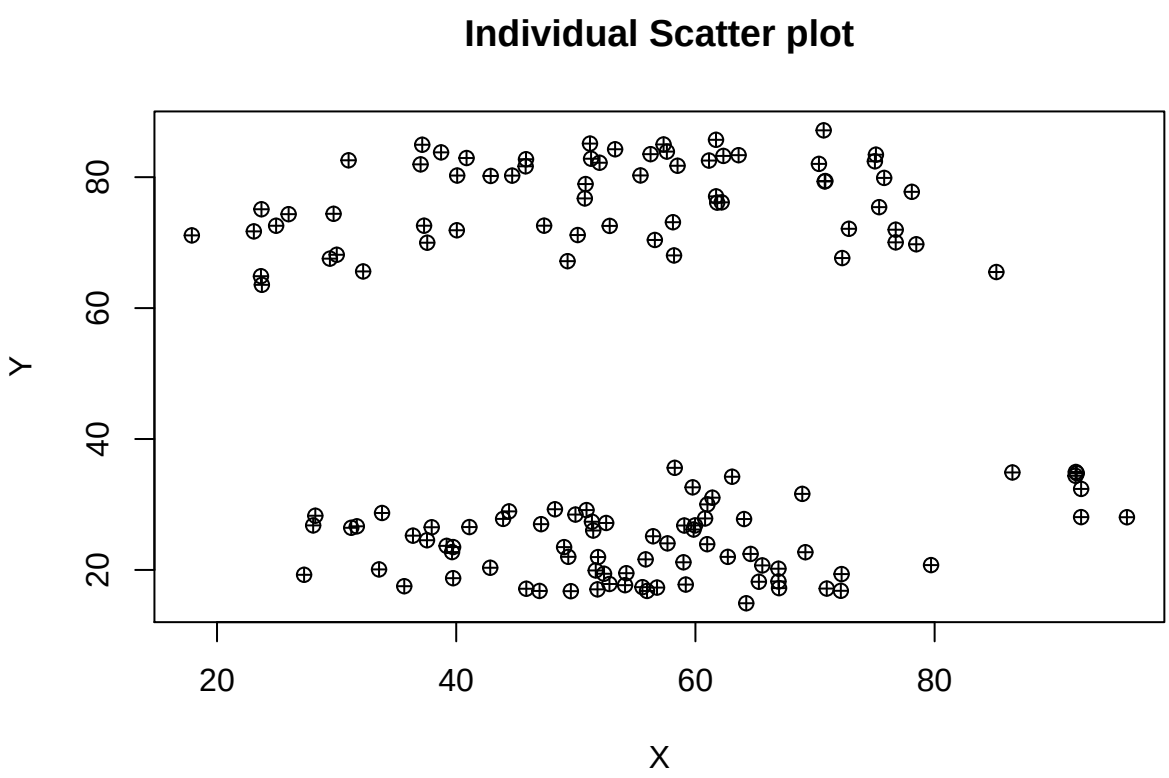

Individual Scatter plot
Y
X
20
40
60
80
20
40
60
80

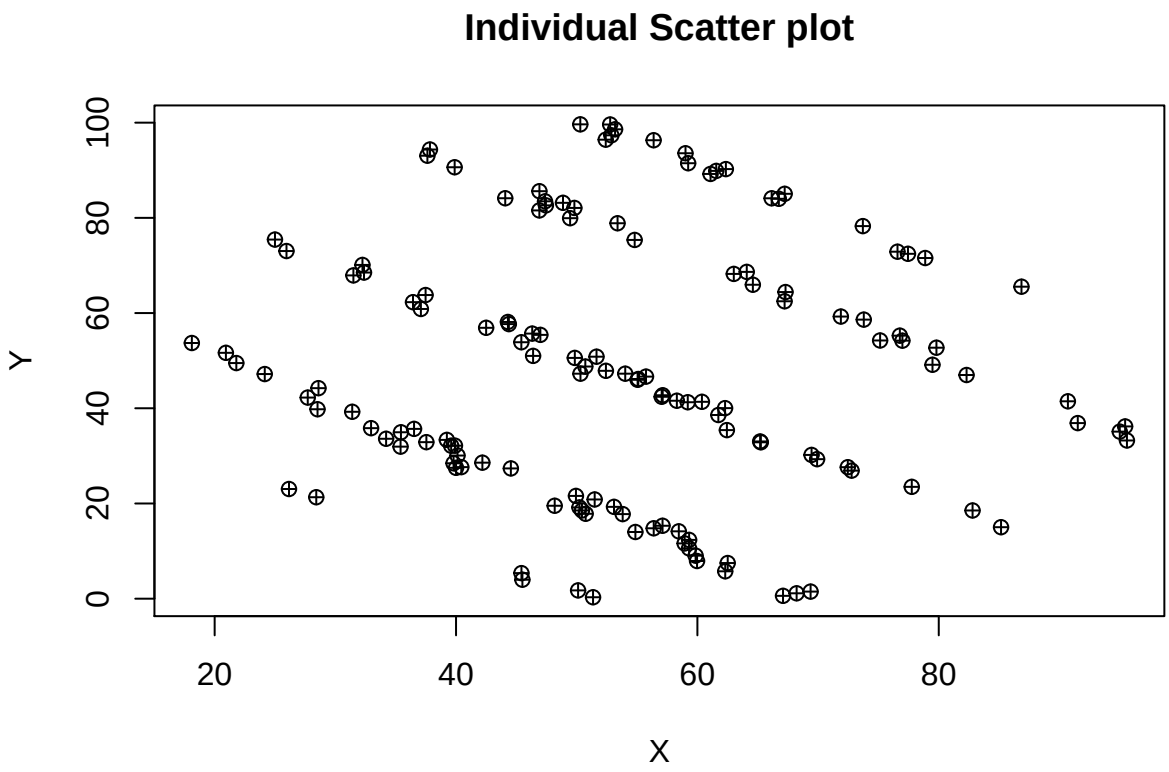

Individual Scatter plot
Y
20
30
40
50
60
70
80
X
20
30
40
50
60
70
80

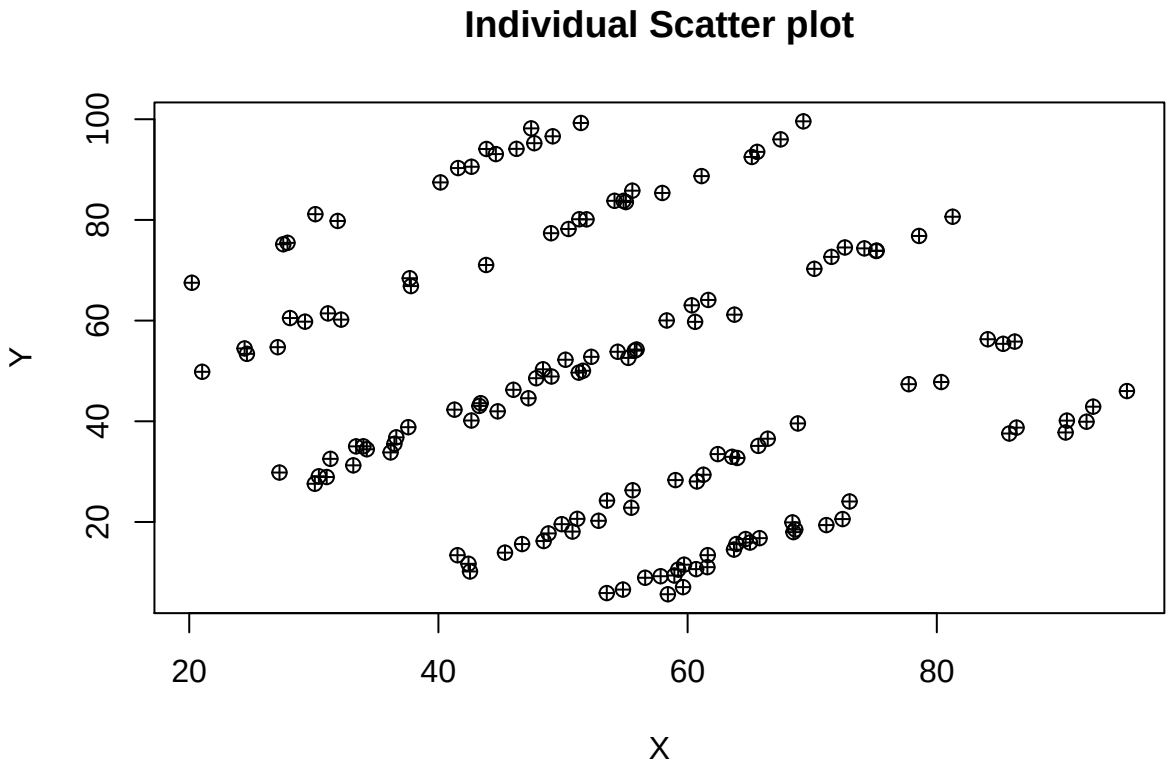

Individual Scatter plot
Y
X
0
20
40
60
80
100
20
40
60
80
100

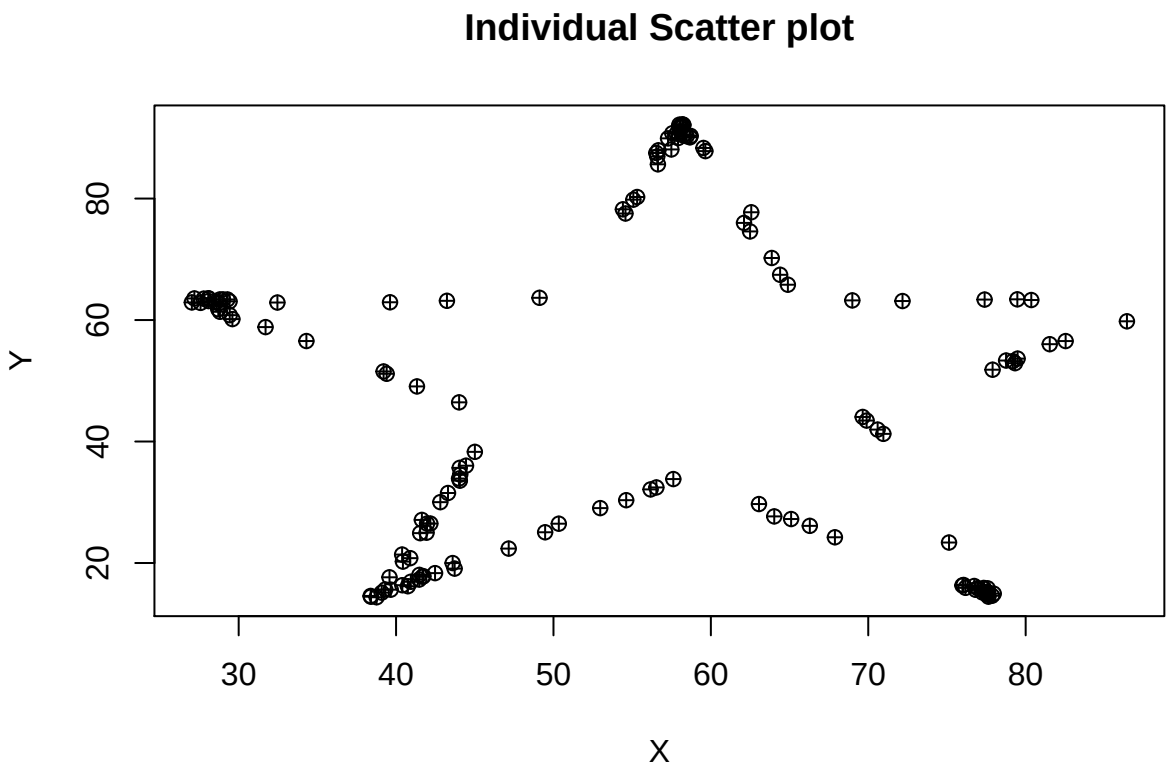

Individual Scatter plot
Y
20
40
60
80
30
40
50
60
70
80
X

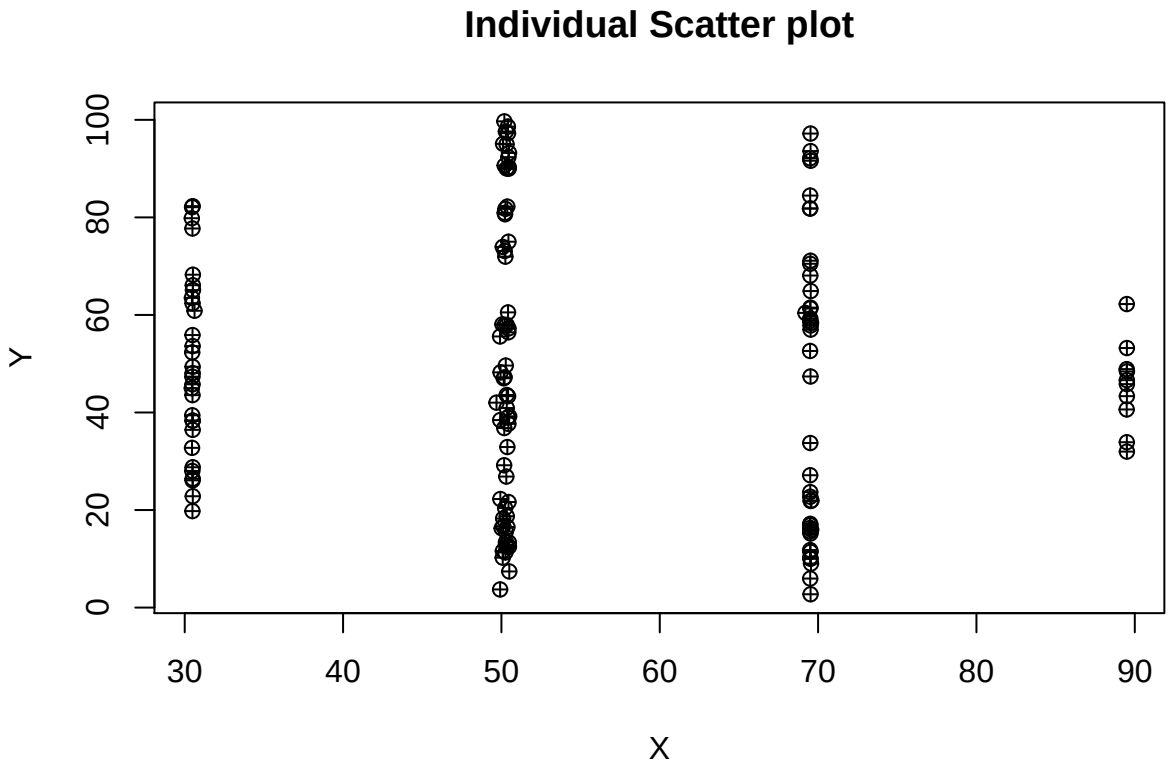

Individual Scatter plot
Y
20
40
60
80
X
20
40
60
80
100

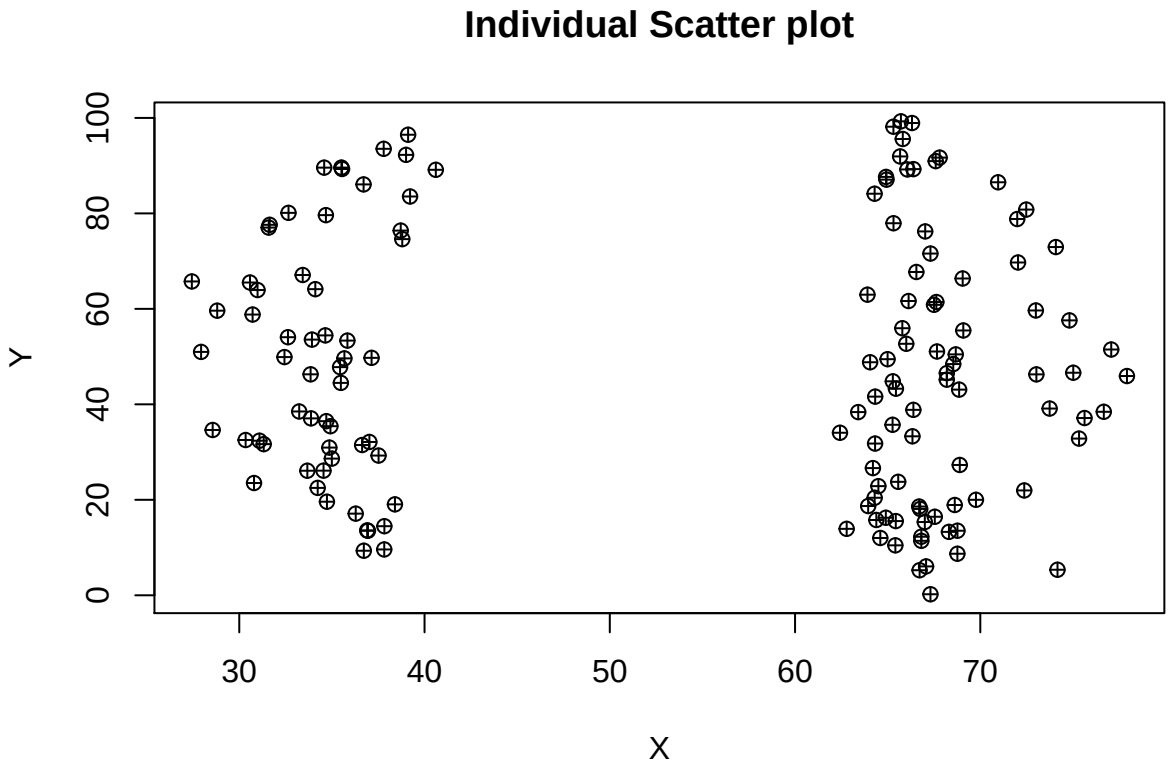

Individual Scatter plot
Y
20
40
60
80
X
20
40
60
80

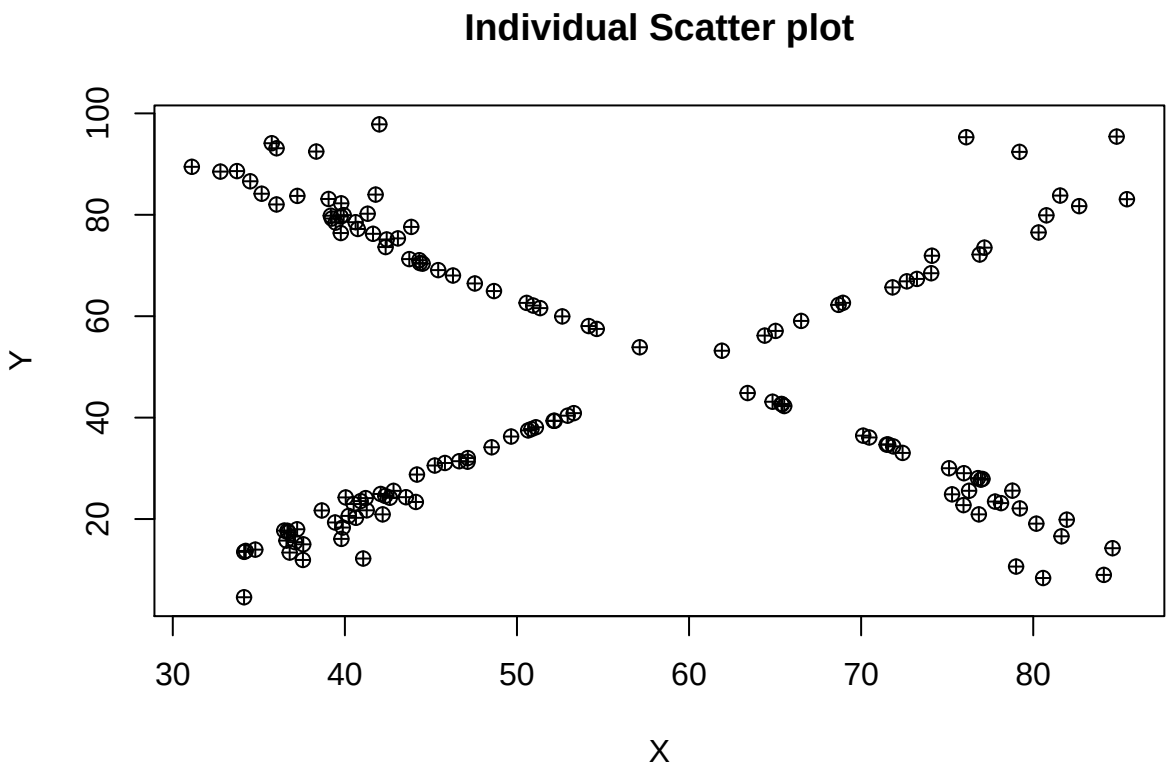

Individual Scatter plot
Y
X
0
20
40
60
80
100
20
40
60
80


Individual Scatter plot
Y
X
20
40
60
80
100
20
40
60
80


Individual Scatter plot
Y
X
20
40
60
80
30
40
50
60
70
80


Individual Scatter plot
Y
X
0
20
40
60
80
100
30
40
50
60
70
80
90


Individual Scatter plot
Y
0
20
40
60
80
100
30
40
50
60
70
X




```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
##
## [[8]]
## NULL
##
## [[9]]
## NULL
```

```
##
## [[10]]
## NULL
##
## [[11]]
## NULL
##
## [[12]]
## NULL
##
## [[13]]
## NULL
```

The 13 scatterplots have been inserted above, each with fascinating designs!

Problem 5

Our ultimate goal in this problem is to create an annotated map of the US. I am giving you the code to create said map, you will need to customize it to include the annotations. Part a. Get and import a database of US cities and states.

```
    #we are grabbing a SQL set from here
    options(repos = c(CRAN="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")

    #read in data, looks like sql dump, blah
    install.packages("data.table")
```

```
## Installing package into 'C:/Users/Niko/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
## Warning: unable to access index for repository http://www.farinspace.com/wp-content/uploads/us_cities
##   cannot open URL 'http://www.farinspace.com/wp-content/uploads/us_cities_and_states.zip/src/contrib/
```

```
## Warning: package 'data.table' is not available (for R version 3.6.1)
```

```
## Warning: unable to access index for repository http://www.farinspace.com/wp-content/uploads/us_cities
##   cannot open URL 'http://www.farinspace.com/wp-content/uploads/us_cities_and_states.zip/bin/windows/
```

```
    library(data.table)
    states <- fread(input = "./us_cities_and_states/states.sql",skip = 23,sep = "'", sep2 = ",", header

    cities <- fread(input = "./us_cities_and_states/cities_extended.sql", skip = 23,sep = "'", sep2 = "
    ### YOU do the CITIES
    ### I suggest the cities_extended.sql may have everything you need
    ### can you figure out how to limit this to the 50?

    head(states)
```

```
##            V2 V4
## 1:     Alaska AK
## 2:    Alabama AL
## 3:   Arkansas AR
## 4:    Arizona AZ
## 5: California CA
## 6:   Colorado CO
```

```
    head(cities)
```

```
##            V2 V4
## 1: Holtsville NY
## 2: Holtsville NY
## 3:   Adjuntas PR
## 4:     Aguada PR
## 5:  Aguadilla PR
## 6:  Aguadilla PR
```

The cities and states have been successfully imported.

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

```
cities_u<-unique(cities)
cities_count <- tapply(cities_u$V2, cities_u$V4, length)
print(cities_count)
```

```
##   AK   AL   AR   AZ   CA   CO   CT   DC   DE   FL   GA   HI   IA   ID   IL
##  229  579  605  264 1239  400  269    3   57  524  629   92  937  266 1287
##   IN   KS   KY   LA   MA   MD   ME   MI   MN   MO   MS   MT   NC   ND   NE
##  738  634  803  479  511  430  461  885  810  942  440  360  762  373  528
##   NH   NJ   NM   NV   NY   OH   OK   OR   PA   PR   RI   SC   SD   TN   TX
##  255  579  346   99 1612 1069  585  379 1802   99   70  377  364  548 1466
##   UT   VA   VT   WA   WI   WV   WY
##  250  839  288  493  753  753  176
```

The cities have been uniquely identified within each state and saved into a variable. From thereafter, I use the tapply function to determine the length in each unique cities dataset.

c. Create a function that counts the number of occurances 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 occurance count as a vector.

```
## code source below is if from https://stackoverflow.com/questions/12427385/how-to-calculate-the-numbe

    string.counter<-function(pattern, strings){
  counts<-NULL
  strings<-tolower(strings)
  for(i in 1:length(strings)){
    counts[i]<-length(attr(gregexpr(pattern,strings[i])[[1]], which = "match.length")[attr(gregexpr(pat
  }
```

```
return(counts)
}

letter_count <- sapply(letters,string.counter,strings=states$V2)

head(letter_count)
```

```
##      a b c d e f g h i j k l m n o p q r s t u v w x y z
## [1,] 3 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
## [2,] 4 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [3,] 3 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 2 0 0 0 0 0 0 0
## [4,] 2 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1
## [5,] 2 0 1 0 0 1 0 0 2 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0
## [6,] 1 0 1 1 0 0 0 0 0 0 0 1 0 0 3 0 0 1 0 0 0 0 0 0 0 0
```

A string.counter function was cited and retrieved from a stackoverflow post for use in letter counts. The sapply function was used to count the letter occurrence in each of the 51 state names.

Only the head function was used to prevent from excessive printing.

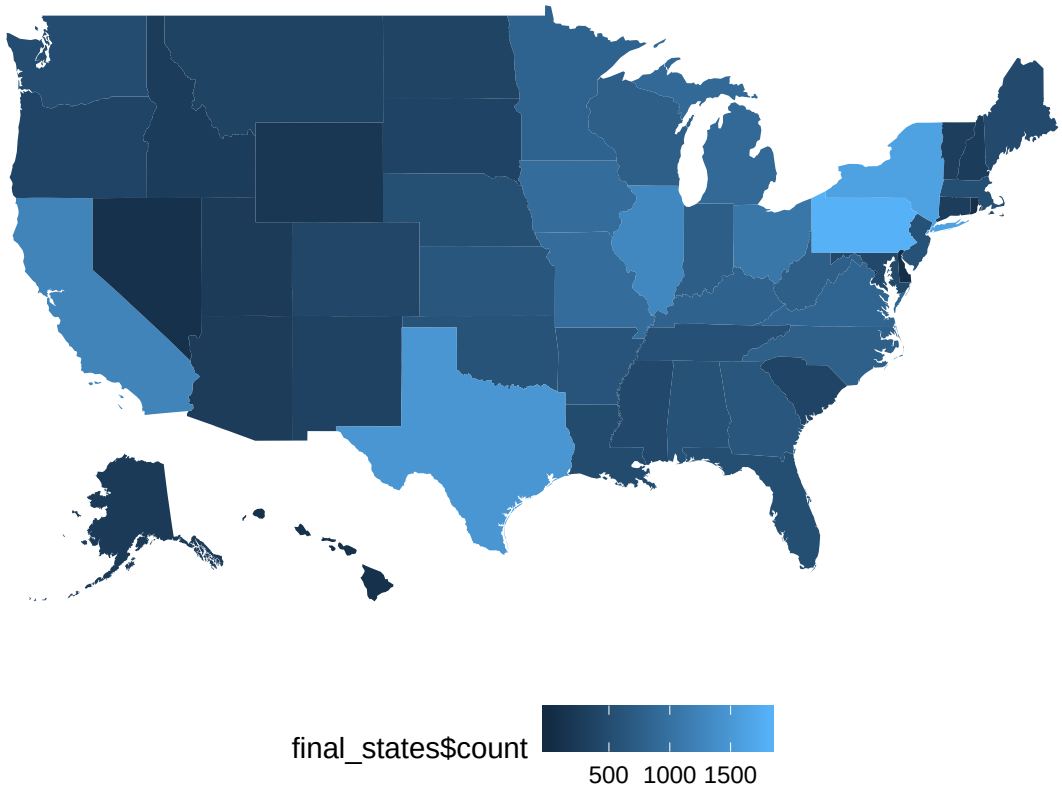
D. Create 2 maps to finalize this.

Map 1 should be colored by count of cities on our list within the state.

```
    #https://cran.r-project.org/web/packages/fiftystater/vignettes/fiftystater.html
    #install.packages("devtools")
    #devtools::install_github("wmurphyrd/fiftystater")
    #install.packages("mapproj")
    library(ggplot2)
    library(fiftystater)
    library(mapproj)
```

```
## Loading required package: maps
```

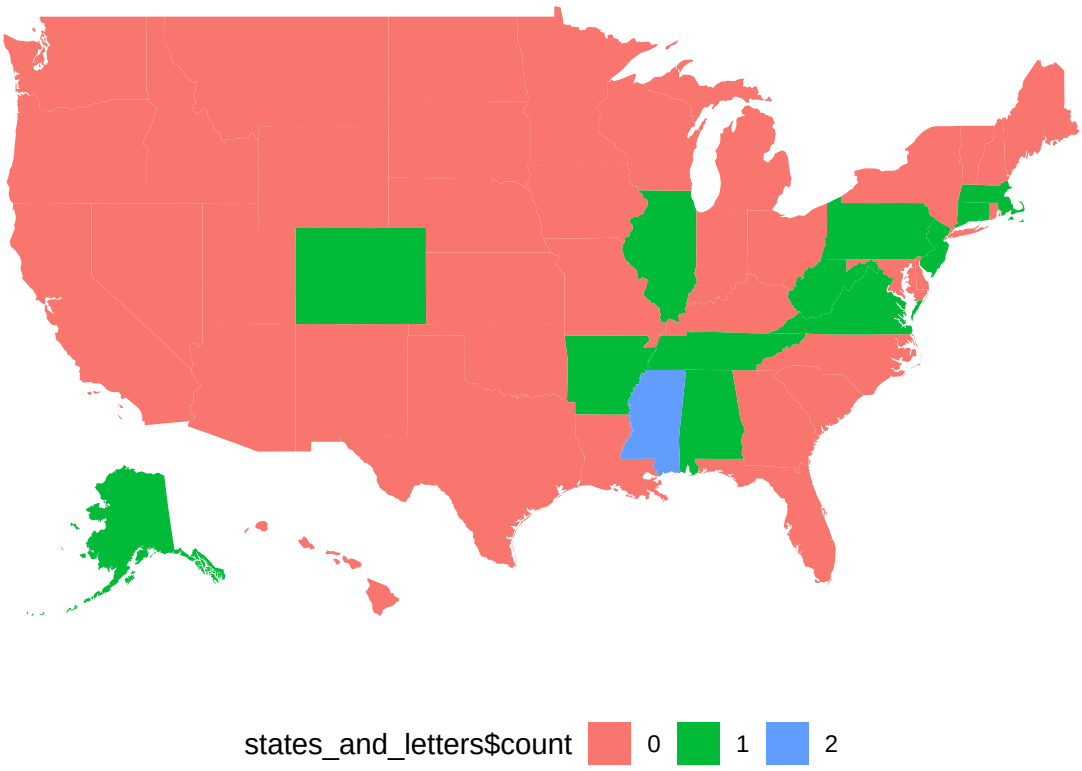
```
    data("fifty_states") # this line is optional due to lazy data loading
    colored_cities <- data.frame(state =rownames(cities_count),count=cities_count)
    deleted_cities <-colored_cities[-c(40),]
    combined_states <-cbind(deleted_cities, states)
    final_states <-combined_states[,-c(1,4)]
    colnames(final_states) <-c("count", "state")
    final_states$state<-tolower(final_states$state)
      map1 <- ggplot(final_states, aes(map_id = state)) +

          # map points to the fifty_states shape data
      geom_map(aes(fill = final_states$count), map = fifty_states) +
      expand_limits(x = fifty_states$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())

    map1
```



The map was created based on the count of cities within each state.

Map 2 should highlight only those states that have more than 3 occurances of ANY letter in thier name.

```
states_and_letters <-cbind(states$V2, rowSums(letter_count>=3))
states_and_letters <- data.frame(states_and_letters)
colnames(states_and_letters) <-c("state","count")

states_and_letters$state<-tolower(states_and_letters$state)
  map2 <- ggplot(states_and_letters, aes(map_id = state)) +

      # map points to the fifty_states shape data
  geom_map(aes(fill = states_and_letters$count), map = fifty_states) +
  expand_limits(x = fifty_states$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())

map2
```


states_and_letters$count
0
1
2