Homework Assignment #6

Matt Kline

March 26, 2017

Part 1 R Programming Structures

1.) In R, functions are objects. They belong to the class of objects appropriately called functions. Use class() or is.function() to verify that each of the objects sd, median, read.table, and data.frame are all functions.1

```
class(sd)

## [1] "function"

class(median)

## [1] "function"

class(read.table)

## [1] "function"

class(data.frame)

## [1] "function"
```

2.) Functions can help organize user interactions. For example, here's a function that quizzes the user to answer a number of questions. It prints out the question, reads a line from standard input, and compares that line to the supplied answer. To print the question in a nice form, we use a low-level printing function, cat(). This function takes any number of arguments and prints them out, with a blank separating data items. The special character "" is the "newline" character. Also, we call a function readline() to read the line from the terminal.

```
quiz <- function(questions, answers) {
  for(i in 1:length(questions)) {
    cat(questions[i], "? ")
    if(readline() == answers[i]) cat("Right!\n")
    else cat("No, the answer is:", answers[i], "\n")
  }
}
```

A session with quiz() would look like:

```
dumbQ <- c("How many planets in the solar system",
           "How many angels on a pin head")
dumbA <- c(8, "Lots")
quiz(dumbQ, dumbA)
```

```
## How many planets in the solar system ?
## No, the answer is: 8
## How many angels on a pin head ?
## No, the answer is: Lots
```

```
## How many planets in the solar system ?
## No, the answer is: 8
## How many angels on a pin head ?
## No, the answer is: Lots
```

Modify quiz() to return a logical vector the same length as questions that specifies which questions the user got wrong. You should set up a vector, say by

```
scores <- NULL
```

and then fill it in.

```
quiz <- function(questions, answers) {
  for(i in 1:length(questions)) {
    cat(questions[i], "? ")
    if(readline() == answers[i]) {
      cat("Right!\n")
      scores <- c(scores, "T")
    } else {
      cat("No, the answer is:", answers[i], "\n")
      scores <- c(scores, "F")
    }
  }
  cat("You scored ", scores)
}
```

3.) When functions call themselves, the process is called recursion. This is often the cleanest way to organize a complicated calculation. Modify quiz() so that it calls itself recursively on the questions that the user got wrong. It should use the vector computed in the previous exercise to select the set of questions and answers for the new call to quiz(). Make sure the function doesn't loop forever if the user gets all the questions right.

```
quiz <- function(questions, answers) {
  for(i in 1:length(questions)) {
    cat(questions[i], "? ")
    if(readline() == answers[i]) {
      cat("Right!\n")
      scores <- c(scores, "T")
    } else {
      cat("No, the answer is:", answers[i], "\n")
      scores <- c(scores, "F")
    }
  }
  quiz(questions[scores = "F"], answers[scores = "F"])
  cat("You scored ", scores)
}
```

4.) Now add to the previous version a little dialog that asks the user whether quiz() should requiz on the errors. You should print out a message, read a line in response, and check whether the reply was "yes" or "no".

```
quiz <- function(questions, answers) {
  for(i in 1:length(questions)) {
    cat(questions[i], "? ")
    if(readline() == answers[i]) {
      cat("Right!\n")
      scores <- c(scores, "T")
    } else {
      cat("No, the answer is:", answers[i], "\n")
      scores <- c(scores, "F")
    }
  }
  cat("Requiz missed questions?")
  if(readline() == "Yes") quiz(questions[scores = "F"], answers[scores = "F"])
  else cat("You scored ", scores)
}
```

5.) Revise the previous version of the quiz() function so that it presents the questions in a random order.

```
quiz <- function(questions, answers) {
  questions <- sample(questions)
  for(i in 1:length(questions)) {
    cat(questions[i], "? ")
    if(readline() == answers[i]) {
      cat("Right!\n")
      scores <- c(scores, "T")
    } else {
      cat("No, the answer is:", answers[i], "\n")
      scores <- c(scores, "F")
    }
  }
  cat("Requiz missed questions?")
  if(readline() == "Yes") quiz(questions[scores = "F"], answers[scores = "F"])
  else cat("You scored ", scores)
}
```

6.) Write a recursive function digits10() that takes an integer as its argument and produces a vector of the digits in its base-10 representation.

```
digits10 <- function(digits) {
  as.integer(substring(digits, seq(nchar(digits)), seq(nchar(digits))))
}

digits10(8289823593)

##  [1] 8 2 8 9 8 2 3 5 9 3
```

7.) Write a function, perm.assign(), that works like the '¡¡-' operator, in that it assigns to the global environment (Workspace), but which is like assign() in that it takes a character string argument for the name. Test your perm.assign() function using the following:

```
perm.assign <- function(x, value) {
  x <- value
```

```
  assign(x, value)
}
```

Part 2 Graphics

8.) R has several built-in data sets. Among them are several data sets related to the 50 states of the United States. Type

```
#?state
```

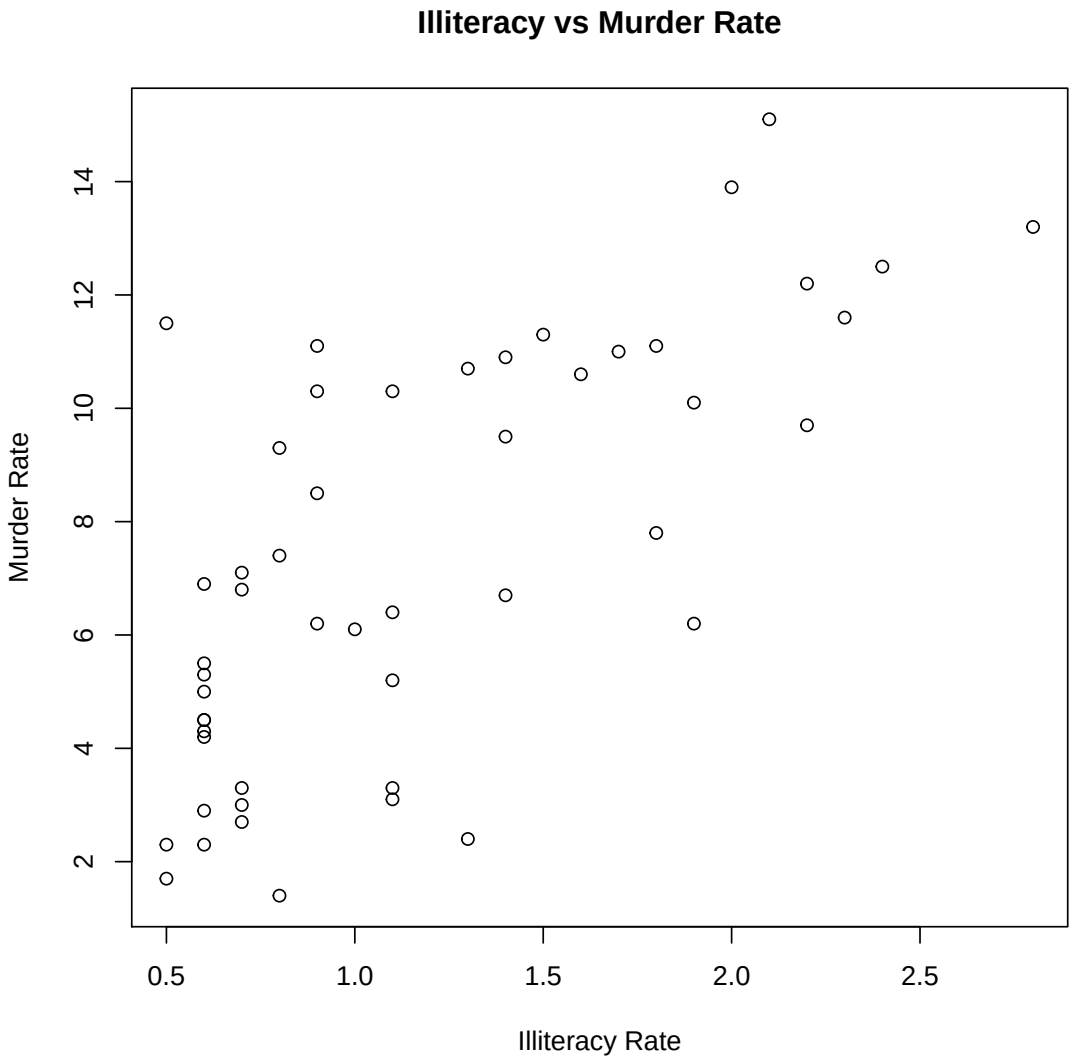
to view their descriptions.

The data set state.x77 is a matrix whose rows each contain the values of several variables for a given state. (Try typing state.x77 at the R command prompt.) Create the following vectors:

```
murder <- state.x77[ , 5]
illit <- state.x77[ , 3]
states <- state.abb
```

a Create a scatterplot of the murder rates (y) versus illiteracy rates (x).

```
plot(x = illit, y = murder, xlab = "Illiteracy Rate", ylab = "Murder Rate",
     main = "Illiteracy vs Murder Rate")
```

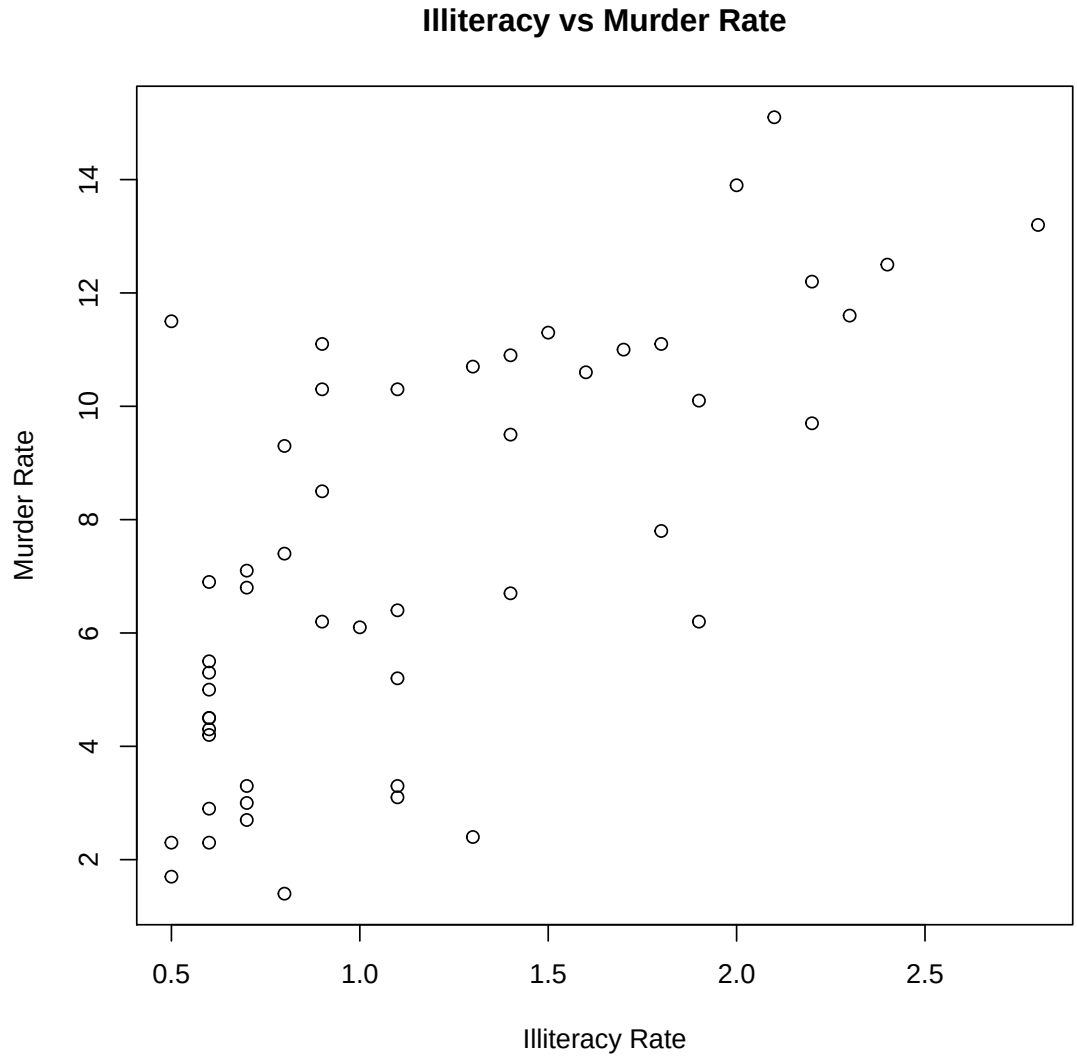


b In the scatterplot, why do the points on the plot appear to line up in vertical columns?

The points appear in columns because of the degree of accuracy that the illiteracy rate is measured and how most of states have a illiteracy rate that is close to the rest of the states. The regions where the illiteracy rate is the same will have varying murder rates causing columns of data.

c The function identify() enables the user to identify an observation in a data set by clicking on a point in a scatterplot. For example, if you have a plot of murder (y) versus illit (x), typing:

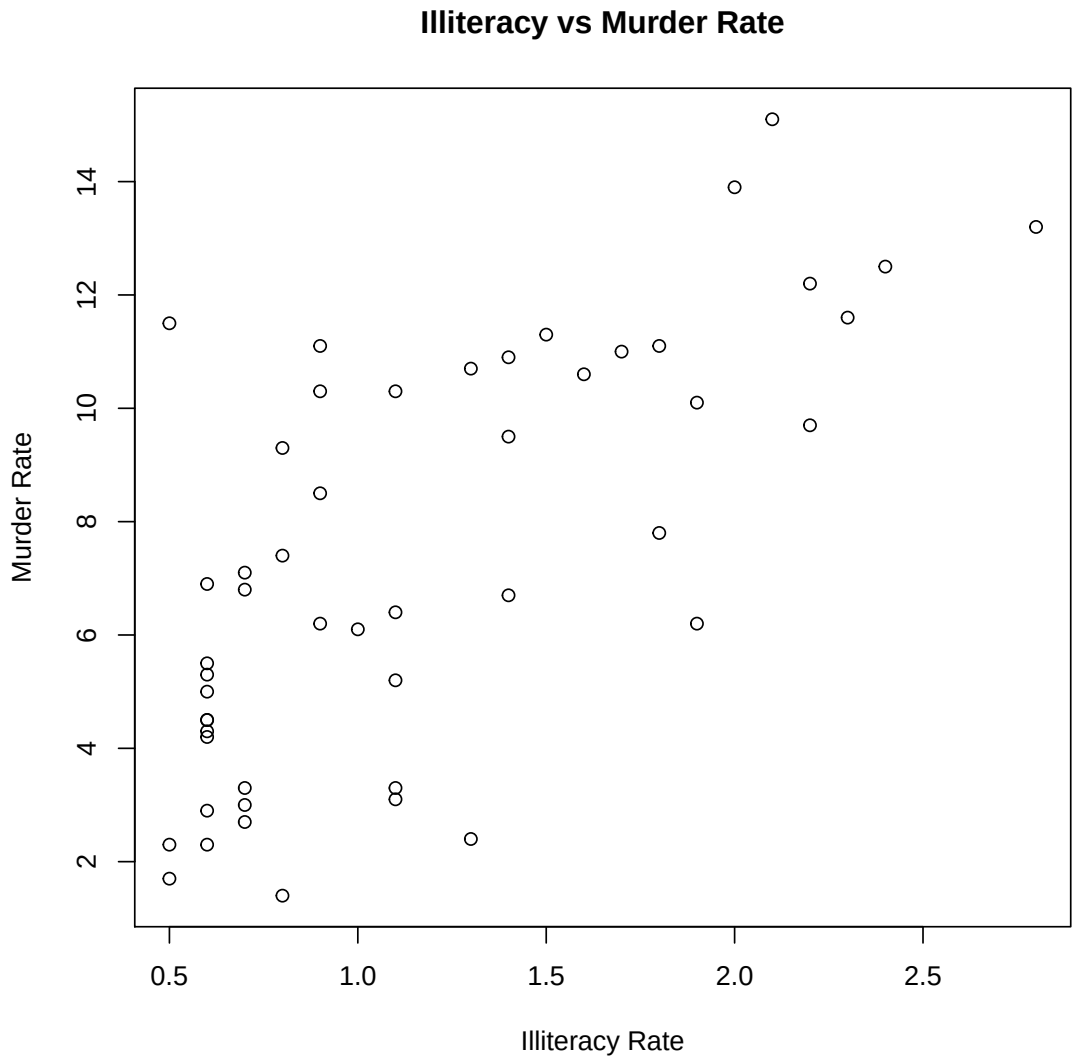
```
plot(x = illit, y = murder, xlab = "Illiteracy Rate", ylab = "Murder Rate",
     main = "Illiteracy vs Murder Rate")

identify(x = illit, y = murder, n = 1)
```




```
## integer(0)
```

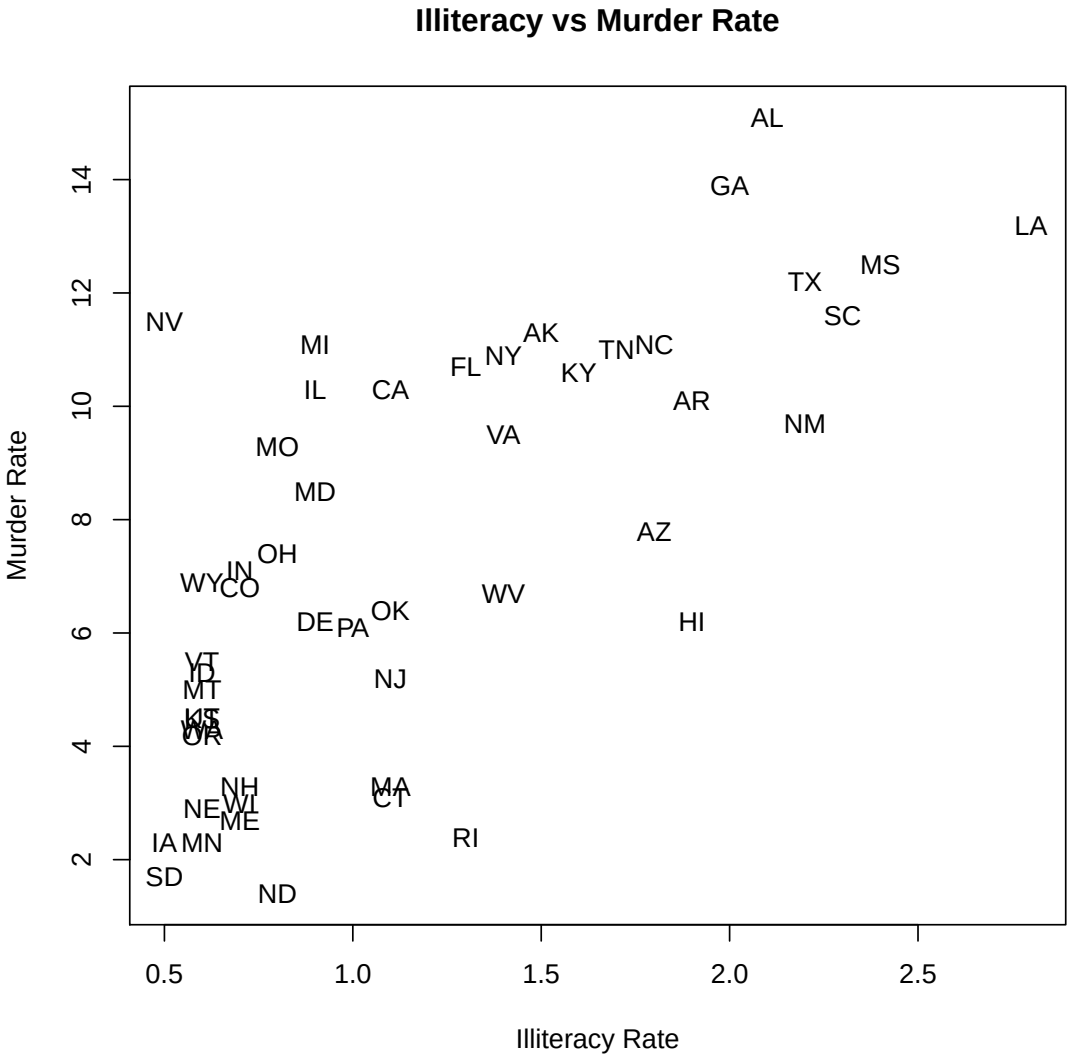
will enable you to click on n = 1 point in the scatterplot. Use the value returned by identify() to produce a new scatterplot of illiteracy and murder rates with any outliers (unusual data points) omitted.

```
plot(x = illit, y = murder, xlab = "Illiteracy Rate", ylab = "Murder Rate",
     main = "Illiteracy vs Murder Rate")
```



d Generate a scatterplot of murder versus illit, but with the state abbreviations labeling the points, instead of the default plotting character.

```
plot(x = illit, y = murder, xlab = "Illiteracy Rate", ylab = "Murder Rate",
     main = "Illiteracy vs Murder Rate", type = "n")
text(illit, murder, states)
```


Illiteracy vs Murder Rate
Murder Rate
Illiteracy Rate
2
4
6
8
10
12
14
0.5
1.0
1.5
2.0
2.5
AL
GA
LA
MS
TX
SC
NV
AK
MI
TN
NC
FL
NY
KY
IL
CA
AR
NM
VA
MO
MD
AZ
OH
IN
WY
CO
WV
DE
PA
OK
HI
VT
ID
MT
NJ
OR
NH
MA
CT
NE
WI
ME
IA
MN
RI
SD
ND