help— title: 'STAT 206 Lab 7' output: pdf_document name:Casey Kongpanickul —

Due Monday, November 20, 5:00 PM

General instructions for labs: You are encouraged to work in pairs to complete the lab. Labs must be completed as an R Markdown file. Be sure to include your lab partner (if you have one) and your own name in the file. Give the commands to answer each question in its own code block, which will also produce plots that will be automatically embedded in the output file. Each answer must be supported by written statements as well as any code used.

Agenda: Debugging and testing

Testing for Outliers

Identifying outliers in data is an important part of statistical analyses. One simple rule of thumb (due to John Tukey) for finding outliers is based on the quartiles of the data: the first quartile Q_1 is the value $\geq 1/4$ of the data, the second quartile Q_2 or the median is the value $\geq 1/2$ of the data, and the third quartile Q_3 is the value $\geq 3/4$ of the data. The interquartile range, IQR, is $Q_3 - Q_1$. Tukeys rule says that the outliers are values more than 1.5 times the interquartile range from the quartiles – either below $Q_1 - 1.5IQR$, or above $Q_3 + 1.5IQR$. Consider the data values

```
x <- c(2.2, 7.8, -4.4, 0.0, -1.2, 3.9, 4.9, 2.0, -5.7, -7.9, -4.9,  28.7,  4.9)
```

We will use these as part of writing a function to identify outliers according to Tukeys rule. Our function will be called `tukey.outlier`, and will take in a data vector, and return a Boolean vector, `TRUE` for the outlier observations and `FALSE` elsewhere.

1. Calculate the first quartile, the third quartile, and the inter-quartile range of `x`. Some built-in R functions calculate these; you cannot use them, but you could use other functions, like `sort` and `quantile`.

```
#To get the quartiles we first need to sort the data. Next we need the median of the first half and the
#median of the last half for the 1st and 3rd quartiles
#the inter quartile range is q3 - q1
#note we use foor division to separate the bottom half from the top half (works on odd/even length)

mine = sort(x)
mine
```

```
##  [1] -7.9 -5.7 -4.9 -4.4 -1.2  0.0  2.0  2.2  3.9  4.9  4.9  7.8 28.7
```

```
#length(mine)

bot_half = mine[1:(length(mine) %/% 2) ]
#bot_half

if(length(mine) %% 2 == 0) {
  top_half = mine[(length(mine) %/% 2) : length(mine)]

  } else {
    top_half = mine[((length(mine) %/% 2) + 2) : (length(mine))]
  }


if(length(bot_half) %% 2 == 0) {
  q1 = bot_half[length(bot_half) %/% 2]
} else {
```

```
  q1 = median(bot_half)
}


if(length(top_half) %% 2 == 0) {
  q3 = top_half[length(top_half) %/% 2]
} else {
  q3 = median(top_half)
}



IQR = q3 - q1

q1
```

```
## [1] -4.9
```

```
q3
```

```
## [1] 4.9
```

```
IQR
```

```
## [1] 9.8
```

2. Write a function, `quartiles`, which takes a data vector and returns a vector of three components, the first quartile, the third quartile, and the inter-quartile range. Show that it gives the right answers on `x`. (You do not have to write a formal test for quartiles.)

```
#The work has already been done above, Now I'm going to wrap it in a function

quartiles = function(data) {
  sorted = sort(data)
  bot_half = sorted[1:(length(mine) %/% 2) ]
  if(length(sorted) %% 2 == 0) {
  top_half = sorted[(length(sorted) %/% 2) : length(sorted)]

  } else {
    top_half = sorted[((length(sorted) %/% 2) + 2) : (length(sorted))]
  }

if(length(bot_half) %% 2 == 0) {
  q1 = bot_half[length(bot_half) %/% 2]
} else {
  q1 = median(bot_half)
}


if(length(top_half) %% 2 == 0) {
  q3 = top_half[length(top_half) %/% 2]
} else {
  q3 = median(top_half)
}
```

```
IQR = q3 - q1


  return(c(q1 = q1, q3 = q3, IQR = IQR))

}

quartiles(x)
```

```
##   q1   q3  IQR
## -4.9  4.9  9.8
```

3. Which points in `x` are outliers, according to Tukeys rule, if any?

```
outliers = c()
high_pt = q3 * 1.5
low_pt = q1 * 1.5


quartiles(x)
```

```
##   q1   q3  IQR
## -4.9  4.9  9.8
```

```
for(value in x){
  if( value > high_pt) {
    outliers = c(outliers, value)
  }
}

for(value in x){
  if( value < low_pt) {
    outliers = c(outliers, value)
  }
}



outliers
```

```
## [1]  7.8 28.7 -7.9
```

4. Write a function, `test.tukey.outlier`, which tests the function `tukey.outlier` against your answer in the previous question. This function should return `TRUE` if `tukey.outlier` works properly; otherwise, it can either return `FALSE`, or an error message, as you prefer. (You can do the next problem first, if you find that easier.)

```
#doing problem #5 fist

tukey.outlier = function(data) {
  bool = c()
  first = quartiles(x)
  q1 = first[1]
  q3 = first[2]

  outliers = c()
```

```
high_pt = q3 * 1.5
low_pt = q1 * 1.5

for(value in x){
  if( value > high_pt) {
    outliers = c(outliers, value)
  }
}

for(value in x){
  if( value < low_pt) {
    outliers = c(outliers, value)
  }
}

for(value in x) {
  if(value %in% outliers) {
    bool = c(bool, TRUE)
  } else {
    bool = c(bool, FALSE)
  }
}
  return(bool)
}

tukey.outlier(x)
```

```
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [12]  TRUE FALSE
```

5. Write `tukey.outlier`, using your `quartiles` function. The function should take a single data vector, and return a Boolean vector, take in a data vector, and return a Boolean vector, `TRUE` for the outlier observations and `FALSE` elsewhere. Show that it passes `test.tukey.outlier`.

```
#now doing #4

test.tukey.outlier = function(x) {

bool = tukey.outlier(x)

first = quartiles(x)
  q1 = first[1]
  q3 = first[2]

  outliers = c()
high_pt = q3 * 1.5
low_pt = q1 * 1.5

i = 0
checkz = c()

for(value in bool){
  i = i + 1

if(value == TRUE) {
```

```
  checkz = c(checkz, x[i])
}


}

final_check = c()

for(value in checkz) {
  if (low_pt >  value) {
    final_check = c(final_check, 0)
  } else if( value > high_pt) {
    final_check = c(final_check, 0)
  } else {
    final_check = c(final_check, 1)
  }

}

if(sum(final_check == 0)) {
  answer = TRUE
} else {
  answer = FALSE
}

return(answer)

}

test.tukey.outlier(x)
```

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

6. Which data values should be outliers in -x?

```
#It looks like the outliers are the same except with opposite signs which is what I would except

prob_6 = x * -1

sorted_6 = sort(prob_6)

sorted_6
```

```
##  [1] -28.7  -7.8  -4.9  -4.9  -3.9  -2.2  -2.0   0.0   1.2   4.4   4.9
## [12]   5.7   7.9
```

```
q3 = quartiles(sorted_6)[2]
q1 = quartiles(sorted_6)[1]

q3
```

```
##  q3
## 4.4
```

```
q1
```

```
##   q1
## -4.9
```

```
high_pt = q3 * 1.5
low_pt = q1 * 1.5

outliers = c()

for(value in sorted_6){
  if( value > high_pt) {
    outliers = c(outliers, value)
  }
}

for(value in sorted_6){
  if( value < low_pt) {
    outliers = c(outliers, value)
  }
}

outliers
```

```
## [1]   7.9 -28.7  -7.8
```

7. Which data values should be outliers in `100*x`?

```
prob_7 = x * 100


sorted_7 = sort(prob_7)

quartiles(sorted_7)
```

```
##   q1   q3  IQR
## -490  490  980
```

```
q3 = quartiles(sorted_7)[2]
q1 = quartiles(sorted_7)[1]

q3
```

```
##  q3
## 490
```

```
q1
```

```
##   q1
## -490
```

```
high_pt = q3 * 1.5
low_pt = q1 * 1.5

outliers = c()

for(value in sorted_7){
  if( value > high_pt) {
    outliers = c(outliers, value)
  }
```

```
}

for(value in sorted_7){
  if( value < low_pt) {
    outliers = c(outliers, value)
  }
}

outliers
```

```
## [1]  780 2870 -790
```

8. Modify `test.tukey.outlier` to include tests for these cases.

```
test.tukey.outlier(prob_6)
```

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

```
test.tukey.outlier(prob_7)
```

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

9. Show that your `tukey.outlier` function passes the new set of tests, or modify it until it does.

```
tukey.outlier(prob_6)
```

```
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [12]  TRUE FALSE
```

```
prob_6
```

```
##  [1]  -2.2  -7.8   4.4   0.0   1.2  -3.9  -4.9  -2.0   5.7   7.9   4.9
## [12] -28.7  -4.9
```

```
tukey.outlier(prob_7)
```

```
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [12]  TRUE FALSE
```

```
prob_7
```

```
##  [1]  220  780 -440    0 -120  390  490  200 -570 -790 -490 2870  490
```

10. According to Tukeys rule, which points in the next vector are outliers? What is the output of your function? If they differ, explain why.

```
#Everything looks to check out on this one. There are many outliers because q1 and q3 are
#quite close to eachother and relatively small numbers for q1 * 1.5 and q3 * 1.5 are also small

y <- c(11.0, 14.0, 3.5, 52.5, 21.5, 12.7, 16.7, 11.7, 10.8, -9.2, 12.3, 13.8, 11.1)

sorted_y = sort(y)

#sorted_y

q3 = quartiles(sorted_y)[2]
q1 = quartiles(sorted_y)[1]

#q3
#q1
```

```
high_pt = q3 * 1.5
high_pt
```

```
## q3
## 21
```

```
low_pt = q1 * 1.5
low_pt
```

```
##   q1
## 16.2
```

```
outliers = c()

for(value in sorted_y){
  if( value > high_pt) {
    outliers = c(outliers, value)
  }
}

for(value in sorted_y){
  if( value < low_pt) {
    outliers = c(outliers, value)
  }
}


outliers
```

```
##  [1] 21.5 52.5 -9.2  3.5 10.8 11.0 11.1 11.7 12.3 12.7 13.8 14.0
```