Homework 4 - Drew Kearny - Due October 20

Name: Drew Kearny netID: 231005935  
Collaborated with:

Your homework **must be submitted in Word or PDF format, created by calling “Knit Word” or “Knit PDF” from RStudio on your R Markdown document.**  
Submission in other formats may receive a grade of 0\*\*. Your responses must be supported by both textual explanations and the code you generate to produce your result. Note that all R code used to produce your results must be shown in your knitted file.

## Q1 Huber loss function

The Huber loss function (or just Huber function, for short) is defined as:

This function is quadratic on the interval [-1,1], and linear outside of this interval. It transitions from quadratic to linear “smoothly”, and looks like this:  
  
It is often used in place of the usual squared error loss for robust estimation. The sample average, —which given a sample minimizes the squared error loss over all choices of —can be inaccurate as an estimate of if the distribution of is heavy-tailed. In such cases, minimizing Huber loss can give a better estimate.

1. Write a function huber() that takes as an input a number , and returns the Huber value , as defined above. Hint: the body of a function is just a block of R code, e.g., in this code you can use if() and else() statements. Check that huber(1) returns 1, and huber(4) returns 7.

huber <- function(x) {  
 if (abs(x) <= 1) {  
 return(x^2)  
 } else {  
 return(2 \* abs(x) - 1)  
 }  
}  
  
huber(1)

## [1] 1

huber(4)

## [1] 7

1. The Huber function can be modified so that the transition from quadratic to linear happens at an arbitrary cutoff value , as in:

* Starting with your solution code to the last question, update your huber() function in the code block below so that it takes two arguments: , a number at which to evaluate the loss, and a number representing the cutoff value. It should now return , as defined above. Check that huber(3, 2) returns 8, and huber(3, 4) returns 9.

huber <- function(x, a) {  
 if (abs(x) <= a) {  
 return(x^2)  
 } else {  
 return(2 \* a \* abs(x) - a^2)  
 }  
}  
  
huber(3, 2)

## [1] 8

huber(3, 4)

## [1] 9

1. Update your huber() function in the code block below so that the default value of the cutoff is 1. Check that huber(3) returns 5.

huber <- function(x, a = 1) {  
 if (abs(x) <= a) {  
 return(x^2)  
 } else {  
 return(2 \* a \* abs(x) - a^2)  
 }  
}  
  
huber(3)

## [1] 5

1. Check that huber(a=1, x=3) returns 5. Check that huber(1, 3) returns 1. Explain why these are different.

These are different because the default order of the huber function is to have the x value first and then the a value so (1, 3) sets x=1 and a=3, so when we write huber(a=1, x=3) we are explicity saying a=1 and x=3 which is obviously different.

huber(a=1, x=3)

## [1] 5

huber(1, 3)

## [1] 1

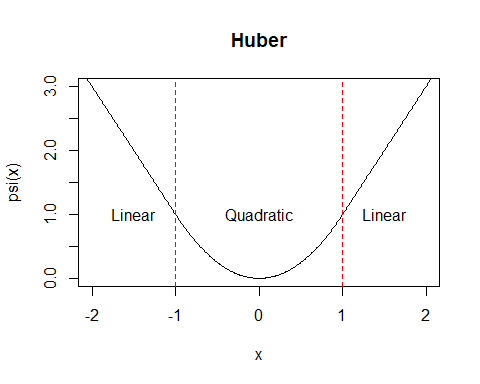
1. Vectorize your huber() function in the code block below, so that the first input can actually be a vector of numbers, and what is returned is a vector whose elements give the Huber evaluated at each of these numbers. Hint: you might try using ifelse(), if you haven’t already, to vectorize nicely. Check that huber(x=1:6, a=3) returns the vector of numbers (1, 4, 9, 15, 21, 27).

huber <- function(x, a = 1) {  
 return(ifelse(abs(x) <= a, x^2, 2 \* a \* abs(x) - a^2))  
}  
  
huber(x=1:6, a=3)

## [1] 1 4 9 15 21 27

1. A plot of the Huber function is displayed at the top of this homework in R. Reproduce this plot with your own plotting code, and the huber() function you wrote above. The axes and title should be just the same, so should the Huber curve (in black), so should be the red dotted lines at the values -1 and 1, and so should the text “Linear”, “Quadratic”, “Linear”.

x <- seq(-3, 3, by = 0.01)  
y <- huber(x, a = 1)  
  
plot(x, y, type = 'l', lwd = 1, col = 'black', xlab = 'x', ylab = 'psi(x)', main = 'Huber', xlim = c(-2, 2), ylim = c(0, 3))  
abline(v = c(-1, 1), col = 'red', lty = 2)  
text(-1.5, 1, 'Linear', col = 'black')  
text(1.5, 1, 'Linear', col = 'black')  
text(0, 1, 'Quadratic', col = 'black')



1. Modify the huber() function so that, as a side effect, it prints the string “Invented by the great Swiss statistician Peter Huber!” to the console. Hint: use cat(). Call your function on an input of your choosing, to demonstrate this side effect.

huber <- function(x, a = 1) {  
 cat("Invented by the great Swiss statistician Peter Huber!\n")  
 return(ifelse(abs(x) <= a, x^2, 2 \* a \* abs(x) - a^2))  
}  
huber(3,1)

## Invented by the great Swiss statistician Peter Huber!

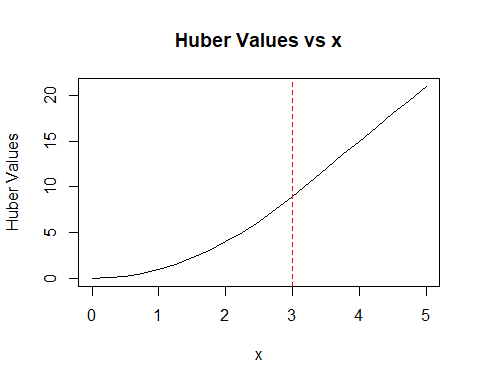
## [1] 5

1. Your instructor computed the Huber function values over a bunch of different values, stored in huber.vals and x.vals, respectively. However, the cutoff was, let’s say, lost. Using huber.vals, x.vals, and the definition of the Huber function, you should be able to figure out the cutoff value , at least roughly. Estimate and explain how you got there. Hint: one way to estimate is to do so visually, using plotting tools; there are other ways too.

Explanation: I created a plot for the x.vals vs huber.vals and looked to see where on the x-axis the line of the graph appeared to turn from a quadratic to linear shape. Then I added a dotted line at x = 3 because that appeared to be where it changed. After drawing the line I can safely approximate that a = 3.

x.vals = seq(0, 5, length=21)  
huber.vals = c(0.0000, 0.0625, 0.2500, 0.5625, 1.0000, 1.5625, 2.2500,  
 3.0625, 4.0000, 5.0625, 6.2500, 7.5625, 9.0000, 10.5000,  
 12.0000, 13.5000, 15.0000, 16.5000, 18.0000, 19.5000,   
 21.0000)

plot(x.vals, huber.vals, type = 'l', col = 'black', xlab = 'x', ylab = 'Huber Values',  
 main = 'Huber Values vs x')  
abline(v = 3, col = 'red', lty = 2)



## Q2 Getting lines of text play-by-play

Recall, as we saw in Week 4, that the complete works of [William Shakespeare](https://en.wikipedia.org/wiki/William_Shakespeare) are available freely from [Project Gutenberg](http://www.gutenberg.org). We’ve put this text file up at <https://raw.githubusercontent.com/schafert/stat404-data/main/shakespeare.txt>.

1. Below is the get.wordtab.from.url() from lecture. Modify this function so that the string vectors lines and words are both included as named components in the returned list. For good practice, update the documentation in comments to reflect your changes. Then call this function on the URL for the Shakespeare’s complete works (with the rest of the arguments at their default values) and save the result as shakespeare.wordobj. Using head(), display the first several elements of (definitely not all of!) the lines, words, and wordtab components of shakespeare.wordobj, just to check that the output makes sense to you.

# get.wordtab.from.url: get a word table from text on the web  
# Inputs:  
# - str.url: string, specifying URL of a web page   
# - split: string, specifying what to split on. Default is the regex pattern  
# "[[:space:]]|[[:punct:]]"  
# - tolower: Boolean, TRUE if words should be converted to lower case before  
# the word table is computed. Default is TRUE  
# - keep.nums: Boolean, TRUE if words containing numbers should be kept in the  
# word table. Default is FALSE  
# Output: list, containing word table, lines, and words.  
  
get.wordtab.from.url = function(str.url, split="[[:space:]]|[[:punct:]]",  
 tolower=TRUE, keep.nums=FALSE) {  
 lines = readLines(str.url)  
 text = paste(lines, collapse=" ")  
 words = strsplit(text, split=split)[[1]]  
 words = words[words != ""]  
   
 # Convert to lower case, if we're asked to  
 if (tolower) words = tolower(words)  
   
 # Get rid of words with numbers, if we're asked to  
 if (!keep.nums)   
 words = grep("[0-9]", words, inv=TRUE, val=TRUE)  
   
 # Compute the word table  
 wordtab = table(words)  
   
 return(list(lines=lines, words=words, wordtab=wordtab,  
 number.unique.words=length(wordtab),  
 number.total.words=sum(wordtab),  
 longest.word=words[which.max(nchar(words))]))  
}

shakespeare.wordobj = get.wordtab.from.url("https://raw.githubusercontent.com/schafert/stat404-data/main/shakespeare.txt")  
  
head(shakespeare.wordobj$lines)

## [1] ""   
## [2] "Project Gutenberg’s The Complete Works of William Shakespeare, by"   
## [3] "William Shakespeare"   
## [4] ""   
## [5] "This eBook is for the use of anyone anywhere in the United States and"  
## [6] "most other parts of the world at no cost and with almost no"

head(shakespeare.wordobj$words)

## [1] "project" "gutenberg" "s" "the" "complete" "works"

head(shakespeare.wordobj$wordtab)

## words  
## a à aaron abaissiez abandon abandoned   
## 16139 1 97 1 10 2

1. Go back and look at Q2 of Homework 3, where you located Shakespeare’s plays in the lines of text for Shakespeare’s complete works. Set shakespeare.lines = shakespeare.wordobj$lines, and then rerun your solution code (or the rerun the official solution code, if you’d like) for Q2 of Homework 3 on the lines of text stored in shakespeare.lines. (Note: you don’t actually need to rerun the code for Q2d or Q2e, since the code for Q2f will accomplish the same task only without encountering NAs). You should end up with two vectors titles.start and titles.end, containing the start and end positions of each of Shakespeare’s plays in shakespeare.lines. Print out titles.start and titles.end to the console.

shakespeare.lines = shakespeare.wordobj$lines  
shakespeare.lines <- shakespeare.lines[shakespeare.lines != ""]  
shakespeare.lines <- trimws(shakespeare.lines)  
  
toc.start <- which(shakespeare.lines == "THE SONNETS")[1]  
toc.end <- which(shakespeare.lines == "VENUS AND ADONIS")[1]  
  
n = toc.end - toc.start + 1  
  
titles <- character(n)  
  
for (i in 1:n) {  
 title\_index <- toc.start + i - 1  
 titles[i] <- shakespeare.lines[title\_index]  
}  
  
titles.start <- integer(n)  
for (i in 1:n) {  
 title <- grep(pattern = titles[i], x = shakespeare.lines)  
 titles.start[i] <- title[2]  
}  
titles.start

## [1] 66 2377 5310 9141 11772 13702 17590 21385 26644 30389  
## [11] 33614 36902 39957 43248 46412 49895 52680 55427 60107 62923  
## [21] 65462 68319 71020 73766 75996 79469 83083 86327 89286 93442  
## [31] 97535 101205 103640 106198 108938 113682 116175 118463 122682 126020  
## [41] 126351 126556 126626 128534

titles.end <- c(titles.start[-1] - 1, length(shakespeare.lines))  
titles.end

## [1] 2376 5309 9140 11771 13701 17589 21384 26643 30388 33613  
## [11] 36901 39956 43247 46411 49894 52679 55426 60106 62922 65461  
## [21] 68318 71019 73765 75995 79468 83082 86326 89285 93441 97534  
## [31] 101204 103639 106197 108937 113681 116174 118462 122681 126019 126350  
## [41] 126555 126625 128533 130094

1. Create a list shakespeare.lines.by.play of length equal to the number of Shakespeare’s plays (a number you should have already computed in the solution to the last question). Using a for() loop, and relying on titles.start and titles.end, extract the appropriate subvector of shakespeare.lines for each of Shakespeare’s plays, and store it as a component of shakespeare.lines.by.play. That is, shakespeare.lines.by.play[[1]] should contain the lines for Shakespeare’s first play, shakespeare.lines.by.play[[2]] should contain the lines for Shakespeare’s second play, and so on. Name the components of shakespeare.lines.by.play according to the titles of the plays.

shakespeare.lines.by.play <- list()  
  
for (i in 1:n) {  
 title <- titles[i]  
 start <- titles.start[i]  
 end <- titles.end[i]   
  
   
 play\_lines <- shakespeare.lines[start:end]  
 shakespeare.lines.by.play[[title]] <- play\_lines  
}  
  
#shakespeare.lines.by.play[[1]] Would print All the lines to the first Play

1. Using one of the apply functions, along with head(), print the first 4 lines of each of Shakespeare’s plays to the console (sorry grader …). This should only require one line of code.

lapply(shakespeare.lines.by.play, head, n = 4)

## $`THE SONNETS`  
## [1] "THE SONNETS"   
## [2] "1"   
## [3] "From fairest creatures we desire increase,"   
## [4] "That thereby beauty’s rose might never die,"  
##   
## $`ALL’S WELL THAT ENDS WELL`  
## [1] "ALL’S WELL THAT ENDS WELL" "Dramatis Personae"   
## [3] "KING OF FRANCE" "THE DUKE OF FLORENCE"   
##   
## $`THE TRAGEDY OF ANTONY AND CLEOPATRA`  
## [1] "THE TRAGEDY OF ANTONY AND CLEOPATRA" "DRAMATIS PERSONAE"   
## [3] "MARK ANTONY, Triumvirs" "OCTAVIUS CAESAR, \""   
##   
## $`AS YOU LIKE IT`  
## [1] "AS YOU LIKE IT"   
## [2] "DRAMATIS PERSONAE."   
## [3] "DUKE, living in exile"   
## [4] "FREDERICK, his brother, and usurper of his dominions"  
##   
## $`THE COMEDY OF ERRORS`  
## [1] "THE COMEDY OF ERRORS" "DRAMATIS PERSONAE"   
## [3] "SOLINUS, Duke of Ephesus" "AEGEON, a merchant of Syracuse"  
##   
## $`THE TRAGEDY OF CORIOLANUS`  
## [1] "THE TRAGEDY OF CORIOLANUS"   
## [2] "Dramatis Personae"   
## [3] "CAIUS MARCIUS, afterwards CAIUS MARCIUS CORIOLANUS"  
## [4] "Generals against the Volscians"   
##   
## $CYMBELINE  
## [1] "CYMBELINE"   
## [2] "Dramatis Personae"   
## [3] "CYMBELINE, King of Britain"   
## [4] "CLOTEN, son to the Queen by a former husband"  
##   
## $`THE TRAGEDY OF HAMLET, PRINCE OF DENMARK`  
## [1] "THE TRAGEDY OF HAMLET, PRINCE OF DENMARK"  
## [2] "by William Shakespeare"   
## [3] "Contents"   
## [4] "ACT I"   
##   
## $`THE FIRST PART OF KING HENRY THE FOURTH`  
## [1] "THE FIRST PART OF KING HENRY THE FOURTH"  
## [2] "by William Shakespeare"   
## [3] "Dramatis Personæ"   
## [4] "KING HENRY the Fourth."   
##   
## $`THE SECOND PART OF KING HENRY THE FOURTH`  
## [1] "THE SECOND PART OF KING HENRY THE FOURTH"  
## [2] "Dramatis Personae"   
## [3] "RUMOUR, the Presenter"   
## [4] "KING HENRY THE FOURTH"   
##   
## $`THE LIFE OF KING HENRY THE FIFTH`  
## [1] "THE LIFE OF KING HENRY THE FIFTH" "DRAMATIS PERSONAE"   
## [3] "CHORUS" "KING HENRY THE FIFTH"   
##   
## $`THE FIRST PART OF HENRY THE SIXTH`  
## [1] "THE FIRST PART OF HENRY THE SIXTH"   
## [2] "Dramatis Personae"   
## [3] "KING HENRY THE SIXTH"   
## [4] "DUKE OF GLOUCESTER, uncle to the King, and Protector"  
##   
## $`THE SECOND PART OF KING HENRY THE SIXTH`  
## [1] "THE SECOND PART OF KING HENRY THE SIXTH"  
## [2] "Dramatis Personae"   
## [3] "KING HENRY THE SIXTH"   
## [4] "HUMPHREY, DUKE OF GLOUCESTER, his uncle"  
##   
## $`THE THIRD PART OF KING HENRY THE SIXTH`  
## [1] "THE THIRD PART OF KING HENRY THE SIXTH"  
## [2] "DRAMATIS PERSONAE"   
## [3] "KING HENRY THE SIXTH"   
## [4] "EDWARD, PRINCE OF WALES, his son"   
##   
## $`KING HENRY THE EIGHTH`  
## [1] "KING HENRY THE EIGHTH"   
## [2] "DRAMATIS PERSONAE"   
## [3] "KING HENRY THE EIGHTH"   
## [4] "CARDINAL WOLSEY CARDINAL CAMPEIUS"  
##   
## $`KING JOHN`  
## [1] "KING JOHN" "DRAMATIS PERSONAE" "KING JOHN"   
## [4] "PRINCE HENRY, his son"  
##   
## $`THE TRAGEDY OF JULIUS CAESAR`  
## [1] "THE TRAGEDY OF JULIUS CAESAR"   
## [2] "Dramatis Personae"   
## [3] "JULIUS CAESAR, Roman statesman and general"   
## [4] "OCTAVIUS, Triumvir after Caesar's death, later Augustus Caesar,"  
##   
## $`THE TRAGEDY OF KING LEAR`  
## [1] "THE TRAGEDY OF KING LEAR" "by William Shakespeare"   
## [3] "Contents" "ACT I"   
##   
## $`LOVE’S LABOUR’S LOST`  
## [1] "LOVE’S LABOUR’S LOST"   
## [2] "Dramatis Personae."   
## [3] "FERDINAND, King of Navarre"   
## [4] "BEROWNE, lord attending on the King"  
##   
## $`THE TRAGEDY OF MACBETH`  
## [1] "THE TRAGEDY OF MACBETH"   
## [2] "Dramatis Personae"   
## [3] "DUNCAN, King of Scotland"   
## [4] "MACBETH, Thane of Glamis and Cawdor, a general in the King's army"  
##   
## $`MEASURE FOR MEASURE`  
## [1] "MEASURE FOR MEASURE" "DRAMATIS PERSONAE" "VINCENTIO, the Duke"  
## [4] "ANGELO, the Deputy"   
##   
## $`THE MERCHANT OF VENICE`  
## [1] "THE MERCHANT OF VENICE"   
## [2] "DRAMATIS PERSONAE"   
## [3] "THE DUKE OF VENICE"   
## [4] "THE PRINCE OF MOROCCO, suitor to Portia"  
##   
## $`THE MERRY WIVES OF WINDSOR`  
## [1] "THE MERRY WIVES OF WINDSOR" "Dramatis Personae"   
## [3] "SIR JOHN FALSTAFF" "FENTON, a young gentleman"   
##   
## $`A MIDSUMMER NIGHT’S DREAM`  
## [1] "A MIDSUMMER NIGHT’S DREAM" "DRAMATIS PERSONAE"   
## [3] "THESEUS, Duke of Athens" "EGEUS, father to Hermia"   
##   
## $`MUCH ADO ABOUT NOTHING`  
## [1] "MUCH ADO ABOUT NOTHING" "by William Shakespeare" "Contents"   
## [4] "ACT I"   
##   
## $`THE TRAGEDY OF OTHELLO, MOOR OF VENICE`  
## [1] "THE TRAGEDY OF OTHELLO, MOOR OF VENICE"   
## [2] "Dramatis Personae"   
## [3] "OTHELLO, the Moor, general of the Venetian forces"  
## [4] "DESDEMONA, his wife"   
##   
## $`PERICLES, PRINCE OF TYRE`  
## [1] "PERICLES, PRINCE OF TYRE" "by William Shakespeare"   
## [3] "Contents" "ACT I"   
##   
## $`KING RICHARD THE SECOND`  
## [1] "KING RICHARD THE SECOND"   
## [2] "DRAMATIS PERSONAE"   
## [3] "KING RICHARD THE SECOND"   
## [4] "JOHN OF GAUNT, Duke of Lancaster - uncle to the King"  
##   
## $`KING RICHARD THE THIRD`  
## [1] "KING RICHARD THE THIRD" "Dramatis Personae" "EDWARD THE FOURTH"   
## [4] "Sons to the King"   
##   
## $`THE TRAGEDY OF ROMEO AND JULIET`  
## [1] "THE TRAGEDY OF ROMEO AND JULIET" "by William Shakespeare"   
## [3] "PERSONS REPRESENTED" "Escalus, Prince of Verona."   
##   
## $`THE TAMING OF THE SHREW`  
## [1] "THE TAMING OF THE SHREW" "by William Shakespeare"   
## [3] "Contents" "INDUCTION"   
##   
## $`THE TEMPEST`  
## [1] "THE TEMPEST" "DRAMATIS PERSONAE" "ALONSO, King of Naples"  
## [4] "SEBASTIAN, his brother"  
##   
## $`THE LIFE OF TIMON OF ATHENS`  
## [1] "THE LIFE OF TIMON OF ATHENS" "DRAMATIS PERSONAE"   
## [3] "TIMON of Athens" "LUCIUS"   
##   
## $`THE TRAGEDY OF TITUS ANDRONICUS`  
## [1] "THE TRAGEDY OF TITUS ANDRONICUS"   
## [2] "Dramatis Personae"   
## [3] "SATURNINUS, son to the late Emperor of Rome, afterwards Emperor"  
## [4] "BASSIANUS, brother to Saturninus"   
##   
## $`THE HISTORY OF TROILUS AND CRESSIDA`  
## [1] "THE HISTORY OF TROILUS AND CRESSIDA" "by William Shakespeare"   
## [3] "Contents" "ACT I"   
##   
## $`TWELFTH NIGHT; OR, WHAT YOU WILL`  
## [1] "TWELFTH NIGHT; OR, WHAT YOU WILL" "DRAMATIS PERSONAE"   
## [3] "ORSINO, Duke of Illyria" "SEBASTIAN, brother of Viola"   
##   
## $`THE TWO GENTLEMEN OF VERONA`  
## [1] "THE TWO GENTLEMEN OF VERONA" "DRAMATIS PERSONAE"   
## [3] "DUKE OF MILAN, father to Silvia" "VALENTINE, one of the two gentlemen"  
##   
## $`THE TWO NOBLE KINSMEN`  
## [1] "THE TWO NOBLE KINSMEN:"   
## [2] "Presented at the Blackfriers by the Kings Maiesties servants, with"  
## [3] "great applause:"   
## [4] "Written by the memorable Worthies of their time;"   
##   
## $`THE WINTER’S TALE`  
## [1] "THE WINTER’S TALE"   
## [2] "Dramatis Personae"   
## [3] "LEONTES, King of Sicilia"   
## [4] "MAMILLIUS, his son, the young Prince of Sicilia"  
##   
## $`A LOVER’S COMPLAINT`  
## [1] "A LOVER’S COMPLAINT"   
## [2] "From off a hill whose concave womb reworded"   
## [3] "A plaintful story from a sist'ring vale,"   
## [4] "My spirits t'attend this double voice accorded,"  
##   
## $`THE PASSIONATE PILGRIM`  
## [1] "THE PASSIONATE PILGRIM"   
## [2] "by William Shakespeare"   
## [3] "I."   
## [4] "Did not the heavenly rhetoric of thine eye,"  
##   
## $`THE PHOENIX AND THE TURTLE`  
## [1] "THE PHOENIX AND THE TURTLE" "by William Shakespeare"   
## [3] "Let the bird of loudest lay," "On the sole Arabian tree,"   
##   
## $`THE RAPE OF LUCRECE`  
## [1] "THE RAPE OF LUCRECE" "by William Shakespeare" "THE RAPE OF LUCRECE"   
## [4] "TO THE"   
##   
## $`VENUS AND ADONIS`  
## [1] "VENUS AND ADONIS"   
## [2] "by William Shakespeare"   
## [3] "\_Vilia miretur vulgus; mihi flavus Apollo"  
## [4] "Pocula Castalia plena ministret aqua.\_"

## Q3 Getting word tables play-by-play

1. Define a function get.wordtab.from.lines() to have the same argument structure as get.wordtab.from.url(), which recall you last updated in Q2a, except that the first argument of get.wordtab.from.lines() should be lines, a string vector passed by the user that contains lines of text to be processed. The body of get.wordtab.from.lines() should be the same as get.wordtab.from.url(), except that lines is passed and does not need to be computed using readlines(). The output of get.wordtab.from.lines() should be the same as get.wordtab.from.url(), except that lines does not need to be returned as a component. For good practice, include documentation for your get.wordtab.from.lines() function in comments.

# get.wordtab.from.lines: Get a word table from a vector of text lines  
# Inputs:  
# - lines: Character vector, containing lines of text to be processed  
# - split: String, specifying what to split on. Default is the regex pattern  
# "[[:space:]]|[[:punct:]]"  
# - tolower: Boolean, TRUE if words should be converted to lowercase before  
# the word table is computed. Default is TRUE  
# - keep.nums: Boolean, TRUE if words containing numbers should be kept in the  
# word table. Default is FALSE  
# Output: list, containing word table, number of unique words, total word count,  
# and the longest word.  
get.wordtab.from.lines = function(lines, split="[[:space:]]|[[:punct:]]", tolower=TRUE, keep.nums=FALSE) {  
 text = paste(lines, collapse=" ")  
 words = strsplit(text, split=split)[[1]]  
 words = words[words != ""]  
   
 # Convert to lower case, if requested  
 if (tolower) words = tolower(words)  
   
 # Get rid of words with numbers, if requested  
 if (!keep.nums)   
 words = grep("[0-9]", words, inv=TRUE, val=TRUE)  
   
 # Compute the word table  
 wordtab = table(words)  
   
 # Return the results as a list  
 return(list(words=words, wordtab=wordtab,  
 number.unique.words=length(wordtab),  
 number.total.words=sum(wordtab),  
 longest.word=words[which.max(nchar(words))]))  
}  
  
shake\_test = get.wordtab.from.url("https://raw.githubusercontent.com/schafert/stat404-data/main/shakespeare.txt")  
  
head(shake\_test$lines)

## [1] ""   
## [2] "Project Gutenberg’s The Complete Works of William Shakespeare, by"   
## [3] "William Shakespeare"   
## [4] ""   
## [5] "This eBook is for the use of anyone anywhere in the United States and"  
## [6] "most other parts of the world at no cost and with almost no"

head(shake\_test$words)

## [1] "project" "gutenberg" "s" "the" "complete" "works"

head(shake\_test$wordtab)

## words  
## a à aaron abaissiez abandon abandoned   
## 16139 1 97 1 10 2

1. Using a for() loop or one of the apply functions (your choice here), run the get.wordtab.from.lines() function on each of the components of shakespeare.lines.by.play, (with the rest of the arguments at their default values). Save the result in a list called shakespeare.wordobj.by.play. That is, shakespeare.wordobj.by.play[[1]] should contain the result of calling this function on the lines for the first play, shakespeare.wordobj.by.play[[2]] should contain the result of calling this function on the lines for the second play, and so on.

shakespeare.wordobj.by.play <- list()  
  
shakespeare.wordobj.by.play <- lapply(shakespeare.lines.by.play, get.wordtab.from.lines)  
  
#shakespeare.wordobj.by.play[[2]] successfully outputs results of second play

1. Using one of the apply functions, compute numeric vectors shakespeare.total.words.by.play and shakespeare.unique.words.by.play, that contain the number of total words and number of unique words, respectively, for each of Shakespeare’s plays. Each vector should only require one line of code to compute. Hint: "[["() is actually a function that allows you to do extract a named component of a list; e.g., try "[["(shakespeare.wordobj, "number.total.words"), and you’ll see this is the same as shakespeare.wordobj[["number.total.words"]]; you should take advantage of this functionality in your apply call. What are the 5 longest plays, in terms of total word count? The 5 shortest plays?

The five longest are “THE TRAGEDY OF HAMLET, PRINCE OF DENMARK”, “KING RICHARD THE THIRD”,“THE TRAGEDY OF CORIOLANUS”,“CYMBELINE”, and “THE TRAGEDY OF KING LEAR”. The five shortest are “THE PHOENIX AND THE TURTLE”,“THE PASSIONATE PILGRIM”, “A LOVER’S COMPLAINT”, “VENUS AND ADONIS”, and “THE RAPE OF LUCRECE”.

shakespeare.total.words.by.play <- sapply(shakespeare.wordobj.by.play, function(play) play[["number.total.words"]])  
  
shakespeare.unique.words.by.play <- sapply(shakespeare.wordobj.by.play, function(play) play[["number.unique.words"]])  
  
longest\_plays <- order(shakespeare.total.words.by.play, decreasing = TRUE)[1:5]  
  
shakespeare.total.words.by.play[longest\_plays]

## THE TRAGEDY OF HAMLET, PRINCE OF DENMARK   
## 32977   
## KING RICHARD THE THIRD   
## 32123   
## THE TRAGEDY OF CORIOLANUS   
## 30210   
## CYMBELINE   
## 29963   
## THE TRAGEDY OF KING LEAR   
## 28686

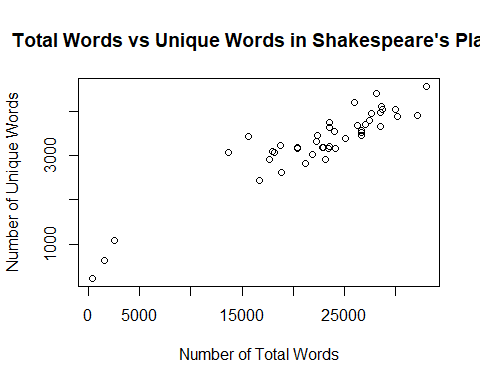
shortest\_plays <- order(shakespeare.total.words.by.play)[1:5]  
  
shakespeare.total.words.by.play[shortest\_plays]

## THE PHOENIX AND THE TURTLE THE PASSIONATE PILGRIM   
## 377 1616   
## A LOVER’S COMPLAINT VENUS AND ADONIS   
## 2608 13634   
## THE RAPE OF LUCRECE   
## 15660

1. Plot the number of unique words versus number of total words, across Shakespeare’s plays. Set the title and label the axes appropriately. Is there a consistent trend you notice?

There is a clear linear trend that as the number of total words in a play increase the number of unique words in a play tends to also increase.

plot(shakespeare.total.words.by.play, shakespeare.unique.words.by.play,   
 xlab = "Number of Total Words",  
 ylab = "Number of Unique Words",  
 main = "Total Words vs Unique Words in Shakespeare's Plays")



1. **Refactoring**: Look back at get.wordtab.from.lines() and get.wordtab.from.url(). Note that they overlap heavily, i.e., their bodies contain a lot of the same code. Redefine get.wordtab.from.url() so that it just calls get.wordtab.from.lines() in its body. Your new get.wordtab.from.url() function should have the same inputs as before, and produce the same output as before. So externally, nothing will have changed; we are just changing the internal structure of get.wordtab.from.url() to clean up our code base (specifically, to avoid code duplication in our case). This is an example of **code refactoring**.

Call your new get.wordtab.from.url() function on the URL for Shakespeare’s complete works, saving the result as shakespeare.wordobj2. Compare some of the components of shakespeare.wordobj2 to those of shakespeare.wordobj (which was computed using the old function definition) to check that your new implementation works as it should.

Output of shakespeare.wordobj2 and shakespeare.wordobj are matching.

get.wordtab.from.url <- function(str.url, split = "[[:space:]]|[[:punct:]]", tolower = TRUE, keep.nums = FALSE) {  
 lines <- readLines(str.url)  
 return(get.wordtab.from.lines(lines, split, tolower, keep.nums))  
}  
  
shakespeare.wordobj2 <- get.wordtab.from.url("https://raw.githubusercontent.com/schafert/stat404-data/main/shakespeare.txt")  
  
head(shakespeare.wordobj2$lines)

## NULL

head(shakespeare.wordobj2$words)

## [1] "project" "gutenberg" "s" "the" "complete" "works"

head(shakespeare.wordobj2$wordtab)

## words  
## a à aaron abaissiez abandon abandoned   
## 16139 1 97 1 10 2

head(shakespeare.wordobj$lines)

## [1] ""   
## [2] "Project Gutenberg’s The Complete Works of William Shakespeare, by"   
## [3] "William Shakespeare"   
## [4] ""   
## [5] "This eBook is for the use of anyone anywhere in the United States and"  
## [6] "most other parts of the world at no cost and with almost no"

head(shakespeare.wordobj$words)

## [1] "project" "gutenberg" "s" "the" "complete" "works"

head(shakespeare.wordobj$wordtab)

## words  
## a à aaron abaissiez abandon abandoned   
## 16139 1 97 1 10 2

## Q4 Basic random number generation

1. Generate the following objects, save them to variables (with names of your choosing), and call head() on those variables.
   * A vector with 1000 standard normal random variables.
   * A vector with 20 draws from .
   * A vector of 2000 characters sampled uniformly from “A”, “G”, “C”, and “T”.
   * A data frame with a column x that contains 100 draws from , and a column y that contains 100 draws of the form . Do this without using explicit iteration.

# A vector with 1000 standard normal random variables  
sdnorm\_random\_vars <- rnorm(1000)  
head(sdnorm\_random\_vars)

## [1] -0.1547754 -1.5444510 1.6880154 0.7260514 -0.5125301 -0.4667456

# A vector with 20 draws from Beta(0.1, 0.1)  
beta\_random\_vars <- rbeta(20, 0.1, 0.1)  
head(beta\_random\_vars)

## [1] 1.281050e-06 3.722403e-07 9.874513e-01 2.411043e-01 8.694589e-09  
## [6] 9.964083e-01

# A vector of 2000 characters sampled uniformly from "A", "G", "C", and "T"  
char\_uniform\_vector <- sample(c("A", "G", "C", "T"), 2000, replace = TRUE)  
head(char\_uniform\_vector)

## [1] "G" "G" "G" "C" "A" "G"

# A data frame with a column x that contains 100 draws from Unif(0, 1), and a column y that contains 100 draws of the form yi ∼ Unif(0, xi)  
x <- runif(100)  
y <- runif(100, max = x)  
data\_frame\_random <- data.frame(x = x, y = y)  
head(data\_frame\_random)

## x y  
## 1 0.09060642 0.013381707  
## 2 0.72467222 0.696844427  
## 3 0.55235026 0.234886123  
## 4 0.67320915 0.222998471  
## 5 0.56267685 0.307646025  
## 6 0.09272344 0.005701009

1. We’ve written a function plot.cum.means() below which plots cumulative sample mean as the sample size increases. The first argument rfun stands for a function which takes one argument n and generates this many random numbers when called as rfun(n). The second argument n.max is an integer which tells the number samples to draw. As a side effect, the function plots the cumulative mean against the number of samples.

# plot.cum.means: plot cumulative sample mean as a function of sample size  
# Inputs:  
# - rfun: function which generates random draws  
# - n.max: number of samples to draw  
# Ouptut: none  
plot.cum.means = function(rfun, n.max) {  
 samples = rfun(n.max)  
 plot(1:n.max, cumsum(samples) / 1:n.max, type = "l")  
}

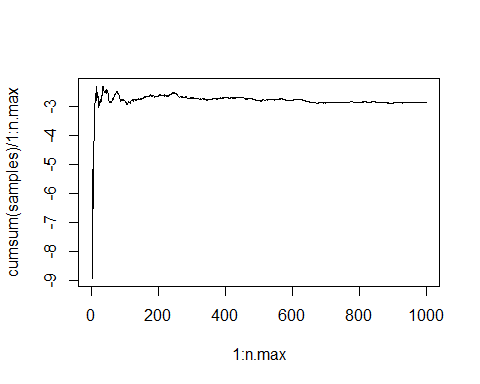
Use this function to make plots for the following distributions, with n.max=1000. Then answer: do the sample means start concentrating around the appropriate value as the sample size increases?

ANSWER: It does appear that in all 3 graphs as the sample size increases, the plots of the sample means begin to start concentrating around the appropriate true mean value,

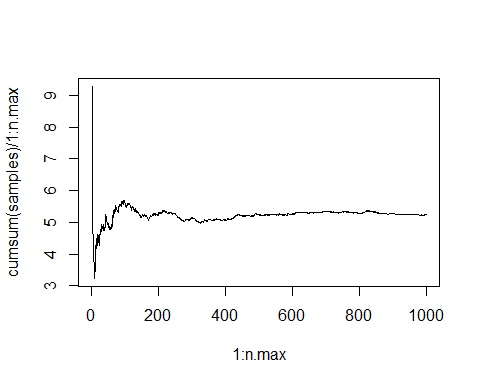
+ $N(-3, 10)$  
+ $\mathrm{Exp}(\mathrm{mean}=5)$   
+ $\mathrm{Beta}(1, 1)$

Hint: for each, you should construct a new single-argument random number generator to pass as the rfun argument to plot.cum.means(), as in function(n) rnorm(n, mean=-3, sd=sqrt(10)) for the first case.

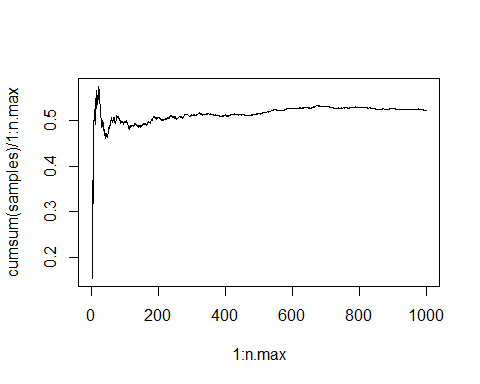
# N(-3, 10)  
normal.rfun <- function(n) rnorm(n, mean = -3, sd = sqrt(10))  
  
plot.cum.means(normal.rfun, n.max = 1000)



# Exp(mean=5)  
exp.rfun <- function(n) rexp(n, rate = 1/5)  
  
plot.cum.means(exp.rfun, n.max = 1000)



# Beta(1, 1)  
beta.rfun <- function(n) rbeta(n, 1, 1)  
  
plot.cum.means(beta.rfun, n.max = 1000)



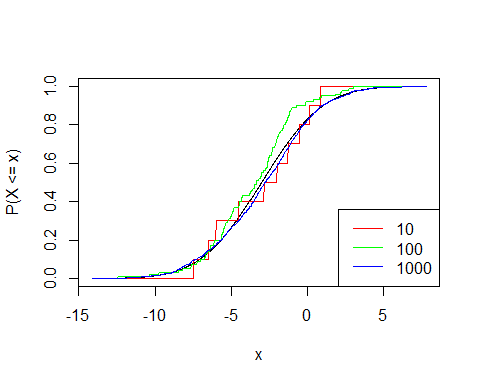
1. For the same distributions as Q4b we will do the following.
   * Generate 10, 100, and 1000 random samples from the distribution.
   * On a single plot, display the ECDFs (empirical cumulative distribution functions) from each set of samples, and the true CDF, with each curve being displayed in a different color.

In order to do this, we’ll write a function plot.ecdf(rfun, pfun, sizes) which takes as its arguments the single-argument random number generating function rfun, the corresponding single-argument conditional density function pfun, and a vector of sample sizes sizes for which to plot the ecdf.

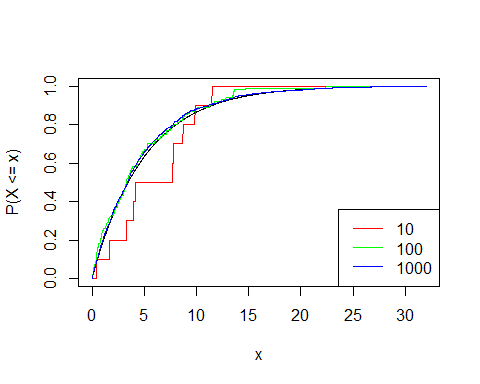
We’ve already started to define plot.ecdf() below, but we’ve left it incomplete. Fill in the definition by editing the lines with “##” and “??”, and then run it on the same distributions as in Q4b. Examine the plots and discuss how the ECDFs converge as the sample size increases. Note: make sure to remove eval=FALSE, after you’ve edited the function, to see the results.

# plot.ecdf: plots ECDFs along with the true CDF, for varying sample sizes  
# Inputs:  
# - rfun: function which generates n random draws, when called as rfun(n)  
# - pfun: function which calculates the true CDF at x, when called as pfun(x)  
# - sizes: a vector of sample sizes  
# Output: none  
  
plot.ecdf = function(rfun, pfun, sizes) {  
 # Draw the random numbers  
 samples = lapply(sizes, rfun)  
  
 # Calculate the grid for the CDF  
 grid.min = min(sapply(samples, min))  
 grid.max = max(sapply(samples, max))  
 grid = seq(grid.min, grid.max, length = 1000)  
  
 # Calculate the ECDFs  
 ecdfs = lapply(samples, ecdf)  
 evals = lapply(ecdfs, function(f) f(grid))  
  
 # Plot the true CDF  
 true\_cdf = pfun(grid)  
 plot(grid, true\_cdf, type = "l", col = "black", xlab = "x", ylab = "P(X <= x)")  
  
 # Plot the ECDFs on top  
 n.sizes = length(sizes)  
 cols = rainbow(n.sizes)  
 for (i in 1:n.sizes) {  
 lines(grid, evals[[i]], col = cols[i])  
 }  
 legend("bottomright", legend = sizes, col = cols, lwd = 1)  
}

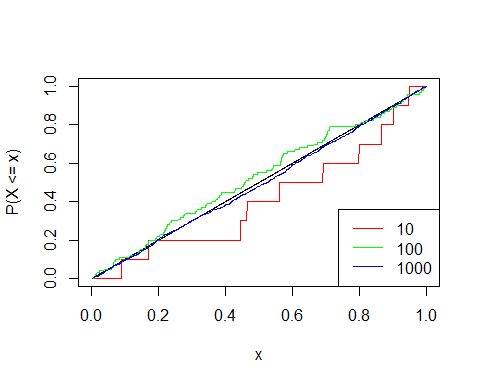
rfun.norm <- function(n) rnorm(n, mean = -3, sd = sqrt(10))  
pfun.norm <- function(x) pnorm(x, mean = -3, sd = sqrt(10))  
  
rfun.exp <- function(n) rexp(n, rate = 1/5)  
pfun.exp <- function(x) pexp(x, rate = 1/5)  
  
rfun.beta <- function(n) rbeta(n, 1, 1)  
pfun.beta <- function(x) pbeta(x, 1, 1)  
  
sizes <- c(10, 100, 1000)  
  
plot.ecdf(rfun.norm, pfun.norm, sizes)



plot.ecdf(rfun.exp, pfun.exp, sizes)



plot.ecdf(rfun.beta, pfun.beta, sizes)

 In the 3 above plots the black line represents the true cdf and we can see that the the higher the sample size is for each distribution the closer the line is too matching the true cdf line. The red line shows each distribution with a sample size 10 and the blue line represents the distribution with sample size 1000, the blue line is super close to exactly following the black line (true cdf) while the red line follows the general direction of the black line but is no where near as close as the blue line and if we run the code over and over the variance of the how the red line is displayed is fairly high.

## Q5 AB testing

A common task in modern data science is to analyze the results of an AB test. AB tests are essentially controlled experiments: we obtain data from two different conditions, such as the different versions of a website we want to show to users, to try to determine which condition gives better results.

1. Write a function to simulate collecting data from an AB test where the responses from the A condition follow a normal distribution with mean a.mean and standard deviation a.sd, whereas responses from the B condition follow a normal distribution with mean b.mean and standard deviation b.sd.

Your function’s signature should be ab.collect(n, a.mean, a.sd, b.mean, b.sd) where n is the number of samples to collect from each condition and the other arguments are as described above. Your function should return a list with two named components a.responses and b.responses which contain the responses for each condition respectively. Try your function out for several values of a.mean, a.sd, b.mean, and b.sd and check that the sample means and standard deviations approximately match the appropriate theoretical values.

The sd’s and means are approximately matching!

ab.collect <- function(n, a.mean, a.sd, b.mean, b.sd) {  
 a.responses <- rnorm(n, mean = a.mean, sd = a.sd)  
 b.responses <- rnorm(n, mean = b.mean, sd = b.sd)  
 return(list(a.responses = a.responses, b.responses = b.responses))  
}  
  
test <- ab.collect(10000, a.mean = 10, a.sd = 1, b.mean = 50, b.sd = 5)  
mean(test$a.responses)

## [1] 10.00366

sd(test$a.responses)

## [1] 1.013728

mean(test$b.responses)

## [1] 49.97913

sd(test$b.responses)

## [1] 5.052913

1. Write a function test.at.end(n, a.mean, a.sd, b.mean, b.sd) which uses your function from Q4a to draw samples of size n and then runs a t-test to determine whether there is a significant difference. We’ll define this as having a p-value at most 0.05. If there is a significant difference, we return either “A” or “B” for whichever condition has the higher mean. If there isn’t no significant difference, we return “Inconclusive”. Hint: recall t.test(), and examine its output on a trial run to figure out how to extract the p-value. Run your function with n=2000, a.mean=100, a.sd=20, b.mean=104, b.sd=10 and display the result.

test.at.end <- function(n, a.mean, a.sd, b.mean, b.sd) {  
 ab.data <- ab.collect(n, a.mean, a.sd, b.mean, b.sd)  
  
 t\_test <- t.test(ab.data$a.responses, ab.data$b.responses)  
 if (t\_test$p.value <= 0.05) {  
 if (mean(ab.data$a.responses) > mean(ab.data$b.responses)) {  
 return("A")  
 } else {  
 return("B")  
 }  
   
 } else {  
 return("Inconclusive")  
 }  
}  
  
# Testing the function with specified values  
test.at.end(n = 2000, a.mean = 100, a.sd = 20, b.mean = 104, b.sd = 10)

## [1] "B"

1. Waiting until you collect all of the samples can take a while. So you instead decide to take the following approach:
   * Every day you collect 100 new observations from each condition.
   * At the end of the day you check whether or not the difference is significant.
   * If the difference is significant you declare the higher response to be the winner.
   * If the difference is not significant you continue onto the next day.
   * As before, if you collect all of the samples without finding a significant difference you’ll declare the result “Inconclusive”.

Note that this kind of sequential sampling is very common in AB testing. Note also the similarity to what we had you do in the drug effect model in lab.

Write a function test.as.you.go(n.per.day, n.days, a.mean, a.sd, b.mean, b.sd) to implement this procedure. Your function should return a list with the winner (or “Inconclusive”), as well and the amount of data you needed to collect.

Run this function on the same example as before with n.per.day=100 and n.days=20 (to match final sample sizes). Do you get the same result? Do you save time collecting data?

ANSWER: I did get the same result of “B” and I saved time because my data collected number of n was lower than before everytime I ran it.

test.as.you.go <- function(n.per.day, n.days, a.mean, a.sd, b.mean, b.sd) {  
 data.a <- numeric(0)  
 data.b <- numeric(0)  
  
 for (day in 1:n.days) {  
 ab\_data\_day <- ab.collect(n.per.day, a.mean, a.sd, b.mean, b.sd)  
   
 data.a <- c(data.a, ab\_data\_day$a.responses)  
 data.b <- c(data.b, ab\_data\_day$b.responses)  
  
 t\_test\_result <- t.test(data.a, data.b)  
  
 if (t\_test\_result$p.value <= 0.05) {  
 if (mean(data.a) > mean(data.b)) {  
 return(list(winner = "A", data\_collected = length(data.a) + length(data.b)))  
 } else {  
 return(list(winner = "B", data\_collected = length(data.b) + length(data.a)))  
 }  
 }  
 }  
  
 return(list(winner = "Inconclusive", data\_collected = length(data.a) + length(data.b)))  
}  
  
  
# Test the function  
test.as.you.go(n.per.day = 100, n.days = 20, a.mean = 100, a.sd = 20, b.mean = 104, b.sd = 10)

## $winner  
## [1] "B"  
##   
## $data\_collected  
## [1] 400

1. In practice, most AB tests won’t have a clear winner; instead both conditions A and B will be roughly equivalent. In this case we want to avoid *false positives*: saying there’s a difference when there isn’t really a difference (with respect to the true distributions). Let’s run a simulation that checks the false positive rate of the two testing regimes.

Setting a.mean = b.mean = 100, a.sd = b.sd = 20, and retaining the number of samples as in the previous examples conduct 1000 AB experiments using each of previous two setups, in test.at.end() and test.as.you.go().

For each, calculate the number of “A” results, “B” results, and “Inconclusive” results. Is this what you would expect to see—recalling that we are declaring significance if the p-value from the t-test is at most 0.05? Does either method of sampling (all-at-once, or as-you-go) perform better than the other, with respect to controlling false positives? **Challenge:** can you explain the behavior you’re seeing, with the sequential sampling?

This is what I would expect to see as most are Inconclusive since we set the mean and sd of both a and b to be the same and declared a significance level of .05 because the lower the significance level the less false positives there will be.

The test.at.end() function does a better job of avoiding false positives because it has much more inconclusive results then the test.as.you.go() function and a false positive in this case would be saying A or B has a significant difference when in fact it doesn’t because we set A and B to have the same distribution.

The sequential sampling with the test.as.you.go() function allows for time to be saved because it can stop early when a significance is detected since this function has data collected and tested each day. So the sequential sampling has the power to detect true differences early on, however if there is no true difference, sequential sampling may still detect a winner due to randomness which can lead to a higher rate of false positives then we will see with the test.at.end() function. Using testing at the end will require a larger sample size and take more time but it is more conservative when it comes to not having false positives, however with this function we don’t have the ability to spot differences early on which may be crucial depending on the scenario.

n.experiments <- 1000  
a.mean <- 100  
a.sd <- 20  
b.mean <- 100  
b.sd <- 20  
  
experiments\_at\_end <- function(n.experiments) {  
 test.result <- rep(NA, n.experiments)  
 for (i in 1:n.experiments) {  
 test.result[i] <- test.at.end(2000, a.mean, a.sd, b.mean, b.sd)  
 }  
 return(test.result)  
}  
  
experiments\_as\_you\_go <- function(n.experiments) {  
 test.result <- rep(NA, n.experiments)  
 for (i in 1:n.experiments) {  
 result <- test.as.you.go(100, 20, a.mean, a.sd, b.mean, b.sd)  
 test.result[i] <- result$winner  
 }  
 return(test.result)  
}  
  
Counts\_experiments\_at\_end <- experiments\_at\_end(n.experiments)  
Counts\_experiments\_as\_you\_go <- experiments\_as\_you\_go(n.experiments)  
  
table(Counts\_experiments\_at\_end)

## Counts\_experiments\_at\_end  
## A B Inconclusive   
## 25 20 955

table(Counts\_experiments\_as\_you\_go)

## Counts\_experiments\_as\_you\_go  
## A B Inconclusive   
## 140 114 746