Homework 3 - Drew Kearny - Due October 6

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 Computing word counts

The following lines of code are used to create shakespeare.words as in the Lab.

shakespeare.lines <- readLines("https://raw.githubusercontent.com/schafert/stat404-data/main/shakespeare.txt")  
  
shakespeare.lines <- shakespeare.lines[shakespeare.lines != ""]  
  
shakespeare.words <- strsplit(paste(shakespeare.lines,   
 collapse = " "),   
 split=" ")[[1]]

1. Using table(), compute counts for the words in shakespeare.words, and save the result as shakespeare.wordtab. How long is shakespeare.wordtab, and is this equal to the number of unique words? Using named indexing, answer: how many times does the word “thou” appear? The word “rumour”? The word “gloomy”? The word “assassination”?

Shakespeare.wordtab has a length of 76171 which is equal to the number of unique words. ‘thou’ appears 4522 times, ‘rumour’ appears 7 times, ‘gloomy’ appears 3 times, and ‘assasination’ appears once.

shakespeare.wordtab <- table(shakespeare.words)  
length(shakespeare.wordtab)

## [1] 76171

length(unique(shakespeare.words))

## [1] 76171

shakespeare.wordtab["thou"]

## thou   
## 4522

shakespeare.wordtab["rumour"]

## rumour   
## 7

shakespeare.wordtab["gloomy"]

## gloomy   
## 3

shakespeare.wordtab["assassination"]

## assassination   
## 1

1. How many words did Shakespeare use just once? Twice? At least 10 times? More than 100 times?

41842 words once, 10756 words twice, 8187 words at least 10 times, and 975 words more than 100 times.

sum(shakespeare.wordtab == 1)

## [1] 41842

sum(shakespeare.wordtab == 2)

## [1] 10756

sum(shakespeare.wordtab >= 10)

## [1] 8187

sum(shakespeare.wordtab > 100)

## [1] 975

1. Sort shakespeare.wordtab so that its entries (counts) are in decreasing order, and save the result as shakespeare.wordtab.sorted. Print the 25 most commonly used words, along with their counts. What is the most common word? Second and third most common words?

The most common word is a space (” “) then it’s”the”, then “I”, and the third most actual word is “and”.

shakespeare.wordtab.sorted <- sort(shakespeare.wordtab, decreasing = TRUE)  
CommonWords <- head(shakespeare.wordtab.sorted, 25)  
print(CommonWords)

## shakespeare.words  
## the I and to of a my in you is   
## 411073 25378 20629 19806 16966 16718 13657 11443 10519 9591 8335   
## that And not with his be your for have it this   
## 8150 7769 7415 7380 6851 6411 6386 6014 5584 5242 5190   
## me he as   
## 5107 5009 4584

most\_common <- names(CommonWords)[1]  
second\_most\_common <- names(CommonWords)[2]  
third\_most\_common <- names(CommonWords)[3]  
most\_common

## [1] ""

second\_most\_common

## [1] "the"

third\_most\_common

## [1] "I"

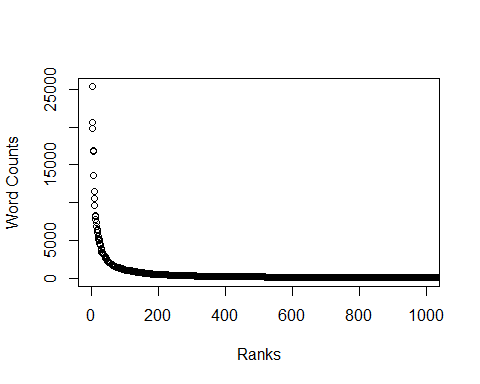
1. What you should have seen in the last question is that the most common word is the empty string ““. This is just an artifact of splitting by spaces, using strsplit(). Redefine shakespeare.words so that all empty strings are deleted from this vector. Then recompute shakespeare.wordtab and shakespeare.wordtab.sorted. Check that you have done this right by printing out the new 25 most commonly used words, and verifying (just visually) that is overlaps with your solution to the last question.

shakespeare.words <- shakespeare.words[shakespeare.words != ""]  
shakespeare.wordtab <- table(shakespeare.words)  
shakespeare.wordtab.sorted <- sort(shakespeare.wordtab, decreasing = TRUE)  
  
CommonWords <- head(shakespeare.wordtab.sorted, 25)  
print(CommonWords)

## shakespeare.words  
## the I and to of a my in you is that And not   
## 25378 20629 19806 16966 16718 13657 11443 10519 9591 8335 8150 7769 7415   
## with his be your for have it this me he as thou   
## 7380 6851 6411 6386 6014 5584 5242 5190 5107 5009 4584 4522

1. As done at the end of the lecture notes, produce a plot of the word counts (y-axis) versus the ranks (x-axis) in shakespeare.wordtab.sorted. Set xlim=c(1,1000) as an argument to plot(); this restricts the plotting window to just the first 1000 ranks, which is helpful here to see the trend more clearly. Be sure the y-axis has numeric tick values.

plot(1:length(shakespeare.wordtab.sorted), shakespeare.wordtab.sorted,   
 type = "p", xlab = "Ranks", ylab = "Word Counts",   
 xlim = c(1, 1000), yaxt = "n")  
axis(2, at = pretty(shakespeare.wordtab.sorted),   
 labels = pretty(shakespeare.wordtab.sorted))



## Q2 Where are Shakespeare’s plays, in this massive text?

1. Let’s go back to shakespeare.lines. Take a look at lines 19 through 23 of this vector: you should see a bunch of spaces preceding the text in lines 21, 22, and 23. Redefine shakespeare.lines by setting it equal to the output of calling the function trimws() on shakespeare.lines. Print out lines 19 through 23 again, and describe what’s happened.

All of the spaces that are in lines 21, 22, and 23 have been cut out and now all 5 lines have no random spaces in the string.

shakespeare.lines[19:23]

## [1] "The Complete Works of William Shakespeare"  
## [2] "by William Shakespeare"   
## [3] " Contents"   
## [4] " THE SONNETS"   
## [5] " ALL’S WELL THAT ENDS WELL"

shakespeare.lines <- trimws(shakespeare.lines)  
  
shakespeare.lines[19:23]

## [1] "The Complete Works of William Shakespeare"  
## [2] "by William Shakespeare"   
## [3] "Contents"   
## [4] "THE SONNETS"   
## [5] "ALL’S WELL THAT ENDS WELL"

1. Visit <https://raw.githubusercontent.com/schafert/stat404-data/main/shakespeare.txt> in your web browser and just skim through this text file. Near the top you’ll see a table of contents. Note that “THE SONNETS” is the first play, and “VENUS AND ADONIS” is the last. Using which(), find the indices of the lines in shakespeare.lines that equal “THE SONNETS”, report the index of the *first* such occurence, and store it as toc.start. Similarly, find the indices of the lines in shakespeare.lines that equal “VENUS AND ADONIS”, report the index of the *first* such occurence, and store it as toc.end.

“THE SONNETS” first appearance is at index 22. “VENUS AND ADONIS” first appearance is at index 65.

toc.start <- which(shakespeare.lines == "THE SONNETS")[1]  
toc.end <- which(shakespeare.lines == "VENUS AND ADONIS")[1]  
toc.start

## [1] 22

toc.end

## [1] 65

1. Define n = toc.end - toc.start + 1, and create an empty string vector of length n called titles. Using a for() loop, populate titles with the titles of Shakespeare’s plays as ordered in the table of contents list, with the first being “THE SONNETS”, and the last being “VENUS AND ADONIS”. Print out the resulting titles vector to the console. Hint: if you define the counter variable i in your for() loop to run between 1 and n, then you will have to index shakespeare.lines carefully to extract the correct titles. Think about the following. When i=1, you want to extract the title of the first play in shakespeare.lines, which is located at index toc.start. When i=2, you want to extract the title of the second play, which is located at index toc.start + 1. And so on.

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

## [1] "THE SONNETS"   
## [2] "ALL’S WELL THAT ENDS WELL"   
## [3] "THE TRAGEDY OF ANTONY AND CLEOPATRA"   
## [4] "AS YOU LIKE IT"   
## [5] "THE COMEDY OF ERRORS"   
## [6] "THE TRAGEDY OF CORIOLANUS"   
## [7] "CYMBELINE"   
## [8] "THE TRAGEDY OF HAMLET, PRINCE OF DENMARK"  
## [9] "THE FIRST PART OF KING HENRY THE FOURTH"   
## [10] "THE SECOND PART OF KING HENRY THE FOURTH"  
## [11] "THE LIFE OF KING HENRY THE FIFTH"   
## [12] "THE FIRST PART OF HENRY THE SIXTH"   
## [13] "THE SECOND PART OF KING HENRY THE SIXTH"   
## [14] "THE THIRD PART OF KING HENRY THE SIXTH"   
## [15] "KING HENRY THE EIGHTH"   
## [16] "KING JOHN"   
## [17] "THE TRAGEDY OF JULIUS CAESAR"   
## [18] "THE TRAGEDY OF KING LEAR"   
## [19] "LOVE’S LABOUR’S LOST"   
## [20] "THE TRAGEDY OF MACBETH"   
## [21] "MEASURE FOR MEASURE"   
## [22] "THE MERCHANT OF VENICE"   
## [23] "THE MERRY WIVES OF WINDSOR"   
## [24] "A MIDSUMMER NIGHT’S DREAM"   
## [25] "MUCH ADO ABOUT NOTHING"   
## [26] "THE TRAGEDY OF OTHELLO, MOOR OF VENICE"   
## [27] "PERICLES, PRINCE OF TYRE"   
## [28] "KING RICHARD THE SECOND"   
## [29] "KING RICHARD THE THIRD"   
## [30] "THE TRAGEDY OF ROMEO AND JULIET"   
## [31] "THE TAMING OF THE SHREW"   
## [32] "THE TEMPEST"   
## [33] "THE LIFE OF TIMON OF ATHENS"   
## [34] "THE TRAGEDY OF TITUS ANDRONICUS"   
## [35] "THE HISTORY OF TROILUS AND CRESSIDA"   
## [36] "TWELFTH NIGHT; OR, WHAT YOU WILL"   
## [37] "THE TWO GENTLEMEN OF VERONA"   
## [38] "THE TWO NOBLE KINSMEN"   
## [39] "THE WINTER’S TALE"   
## [40] "A LOVER’S COMPLAINT"   
## [41] "THE PASSIONATE PILGRIM"   
## [42] "THE PHOENIX AND THE TURTLE"   
## [43] "THE RAPE OF LUCRECE"   
## [44] "VENUS AND ADONIS"

1. Use a for() loop to find out, for each play, the index of the line in shakespeare.lines at which this play begins. It turns out that the *second* occurence of “THE SONNETS” in shakespeare.lines is where this play actually begins (this first ocurrence is in the table of contents), and so on, for each play title. Use your for() loop to fill out an integer vector called titles.start, containing the indices at which each of Shakespeare’s plays begins in shakespeare.lines. Print the resulting vector titles.start to the console.

titles.start <- integer(n)  
for (i in 1:n) {  
 titles.start[i] <- (which(shakespeare.lines == titles[i])[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 NA 122682 126020  
## [41] 126351 126556 126626 128534

1. Define titles.end to be an integer vector of the same length as titles.start, whose first element is the second element in titles.start minus 1, whose second element is the third element in titles.start minus 1, and so on. What this means: we are considering the line before the second play begins to be the last line of the first play, and so on. Define the last element in titles.end to be the length of shakespeare.lines. You can solve this question either with a for() loop, or with proper indexing and vectorization.

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 NA 122681 126019 126350  
## [41] 126555 126625 128533 130094

1. In Q2d, you should have seen that the starting index of Shakespeare’s 38th play “THE TWO NOBLE KINSMEN” was computed to be NA, in the vector titles.start. Why? If you run which(shakespeare.lines == "THE TWO NOBLE KINSMEN") in your console, you will see that there is only one occurence of “THE TWO NOBLE KINSMEN” in shakespeare.lines, and this occurs in the table of contents. So there was no second occurence, hence the resulting NA value.

But now take a look at line 118463 in shakespeare.lines: you will see that it is “THE TWO NOBLE KINSMEN:”, so this is really where the second play starts, but because of colon “:” at the end of the string, this doesn’t exactly match the title “THE TWO NOBLE KINSMEN”, as we were looking for.

which(shakespeare.lines == "THE TWO NOBLE KINSMEN")

## [1] 59

shakespeare.lines[118463]

## [1] "THE TWO NOBLE KINSMEN:"

The advantage of using the grep() function, versus checking for exact equality of strings, is that grep() allows us to match substrings. Specifically, grep() returns the indices of the strings in a vector for which a substring match occurs, e.g.,

grep(pattern="cat",  
 x=c("cat", "canned goods", "batman", "catastrophe", "tomcat"))

## [1] 1 4 5

so we can see that in this example, grep() was able to find substring matches to “cat” in the first, fourth, and fifth strings in the argument x. Redefine titles.start by repeating the logic in your solution to Q5d, but replacing the which() command in the body of your for() loop with an appropriate call to grep(). Also, redefine titles.end by repeating the logic in your solution to Q5e. Print out the new vectors titles.start and titles.end to the console—they should be free of NA values.

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

## Q3 Extracting and analysing a couple of plays

1. Let’s look at two of Shakespeare’s most famous tragedies. Programmatically find the index at which “THE TRAGEDY OF HAMLET, PRINCE OF DENMARK” occurs in the titles vector. Use this to find the indices at which this play starts and ends, in the titles.start and titles.end vectors, respectively. Call the lines of text corresponding to this play shakespeare.lines.hamlet. How many such lines are there? Do the same, but now for the play “THE TRAGEDY OF ROMEO AND JULIET”, and call the lines of text corresponding to this play shakespeare.lines.romeo. How many such lines are there?

“THE TRAGEDY OF HAMLET, PRINCE OF DENMARK” is 5259 lines and “THE TRAGEDY OF ROMEO AND JULIET” is 4093 lines.

hamlet <- which(titles == "THE TRAGEDY OF HAMLET, PRINCE OF DENMARK")  
hamlet\_start <- titles.start[hamlet]  
hamlet\_end <- titles.end[hamlet]  
shakespeare.lines.hamlet <- shakespeare.lines[hamlet\_start:hamlet\_end]  
length(shakespeare.lines.hamlet)

## [1] 5259

romeo <- which(titles == "THE TRAGEDY OF ROMEO AND JULIET")  
romeo\_start <- titles.start[romeo]  
romeo\_end <- titles.end[romeo]  
shakespeare.lines.romeo <- shakespeare.lines[romeo\_start:romeo\_end]  
length(shakespeare.lines.romeo)

## [1] 4093

1. Repeat the following analysis, outlined in the lab and Q2, on shakespeare.lines.hamlet. (This should mostly just involve copying and pasting code as needed.) That is, to be clear:
   * collapse shakespeare.lines.hamlet into one big string, separated by spaces;
   * convert this string into all lower case characters;
   * divide this string into words, by splitting on spaces or on punctuation marks, using split="[[:space:]]|[[:punct:]]" in the call to strsplit();
   * remove all empty words (equal to the empty string ““), and report how many words remain;
   * compute the unique words, report the number of unique words, and plot a histogram of their numbers of characters;
   * report the 5 longest words;
   * compute a word table, and report the 25 most common words and their counts;
   * finally, produce a plot of the word counts verus rank with the same x limits.

There is 32977 words and 5123 unique words in Hamlet.

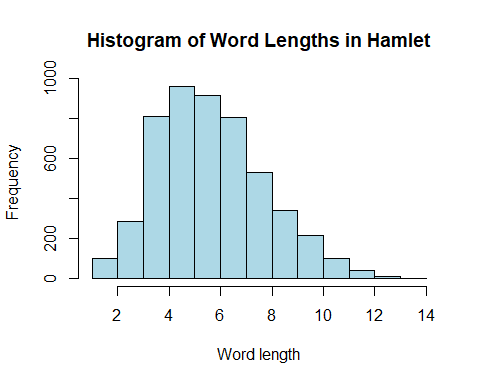
shakespeare.lines.hamlet = paste(shakespeare.lines.hamlet, collapse=" ")  
shakespeare.lines.hamlet = strsplit(shakespeare.lines.hamlet, split="[[:space:]]|[[:punct:]]")[[1]]  
shakespeare.lines.hamlet = shakespeare.lines.hamlet[shakespeare.lines.hamlet != ""]  
shakespeare.lines.hamlet.unique = unique(shakespeare.lines.hamlet)  
  
length(shakespeare.lines.hamlet)

## [1] 32977

length(shakespeare.lines.hamlet.unique)

## [1] 5123

hist(nchar(shakespeare.lines.hamlet.unique), main = "Histogram of Word Lengths in Hamlet", xlab = "Word length", col = "lightblue")



Here Are the 5 longest words:

sorted\_word\_lengths\_hamlet <- shakespeare.lines.hamlet.unique[order(-nchar(shakespeare.lines.hamlet.unique))]  
  
# Get the 5 longest words  
five\_longest\_words\_hamlet <- head(sorted\_word\_lengths\_hamlet, 5)  
  
five\_longest\_words\_hamlet

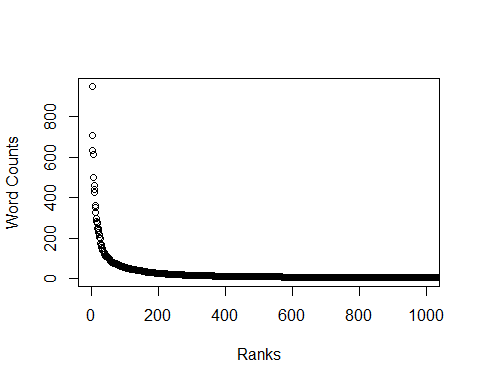
## [1] "transformation" "understanding" "entertainment" "imperfections"   
## [5] "encompassment"

Here are the 25 most common words and their counts.

hamlet.wordtab <- table(shakespeare.lines.hamlet)  
hamlet.wordtab.sorted <- sort(hamlet.wordtab, decreasing = TRUE)  
head(hamlet.wordtab.sorted, 25)

## shakespeare.lines.hamlet  
## the and of to I you a my in HAMLET it   
## 951 706 633 615 613 498 459 443 426 363 354   
## is not d And his that this s me with your   
## 329 299 285 281 278 272 248 246 236 226 220   
## be lord him   
## 204 204 196

plot(1:length(hamlet.wordtab.sorted), hamlet.wordtab.sorted,   
 type = "p", xlab = "Ranks", ylab = "Word Counts",   
 xlim = c(1, 1000), yaxt = "n")  
axis(2, at = pretty(hamlet.wordtab.sorted),   
 labels = pretty(hamlet.wordtab.sorted))



1. Repeat the same task as in the last part, but on shakespeare.lines.romeo. (Again, this should just involve copying and pasting code as needed. P.S. Isn’t this getting tiresome? You’ll be happy when we learn functions, next week!) Comment on any similarities/differences you see in the answers.

There is 26689 words and 4050 unique words in Hamlet.

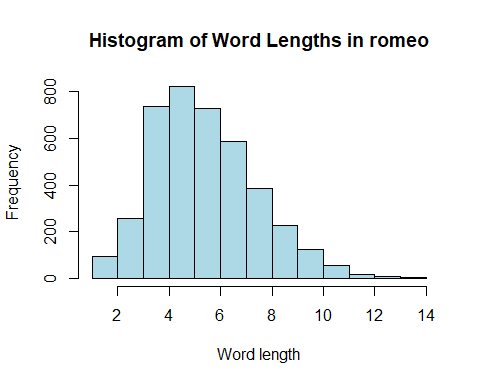
shakespeare.lines.romeo = paste(shakespeare.lines.romeo, collapse=" ")  
shakespeare.lines.romeo = strsplit(shakespeare.lines.romeo, split="[[:space:]]|[[:punct:]]")[[1]]  
shakespeare.lines.romeo = shakespeare.lines.romeo[shakespeare.lines.romeo != ""]  
shakespeare.lines.romeo.unique = unique(shakespeare.lines.romeo)  
  
length(shakespeare.lines.romeo)

## [1] 26689

length(shakespeare.lines.romeo.unique)

## [1] 4050

hist(nchar(shakespeare.lines.romeo.unique), main = "Histogram of Word Lengths in romeo", xlab = "Word length", col = "lightblue")



Here Are the 5 longest words:

sorted\_word\_lengths\_romeo <- shakespeare.lines.romeo.unique[order(-nchar(shakespeare.lines.romeo.unique))]  
  
# Get the 5 longest words  
five\_longest\_words\_romeo <- head(sorted\_word\_lengths\_romeo, 5)  
  
five\_longest\_words\_romeo

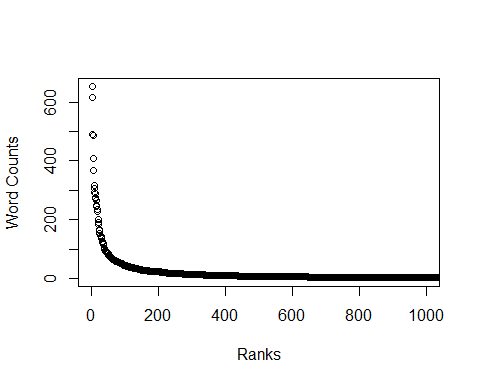
## [1] "distemperature" "unthankfulness" "interchanging" "transgression"   
## [5] "disparagement"

Here are the 25 most common words and their counts.

romeo.wordtab <- table(shakespeare.lines.romeo)  
romeo.wordtab.sorted <- sort(romeo.wordtab, decreasing = TRUE)  
head(romeo.wordtab.sorted, 25)

## shakespeare.lines.romeo  
## I the to and a of my is in s that you me   
## 653 615 490 485 409 367 317 307 293 289 276 274 264   
## d not thou And with be it this for ROMEO O Romeo   
## 249 246 236 229 228 201 192 185 167 164 152 148

plot(1:length(romeo.wordtab.sorted), romeo.wordtab.sorted,   
 type = "p", xlab = "Ranks", ylab = "Word Counts",   
 xlim = c(1, 1000), yaxt = "n")  
axis(2, at = pretty(romeo.wordtab.sorted),   
 labels = pretty(romeo.wordtab.sorted))

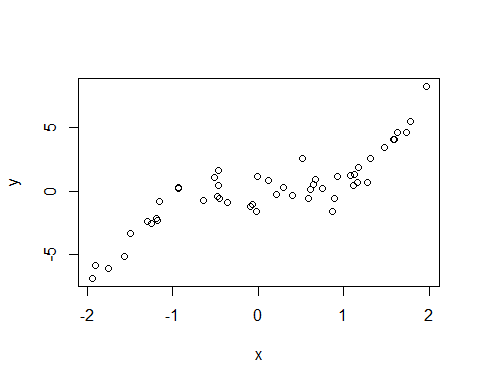
 Most things are different from each other between the two except the graphs look pretty similar and they both share a lot of the same common words like “the” and “I”

## Q4 Plot basics

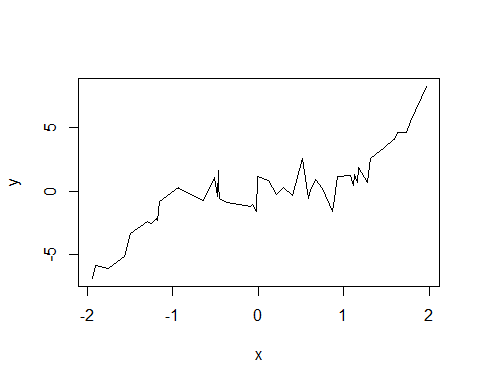
1. Below is some code that is very similar to that from the lecture, but with one key difference. Explain: why does the plot() result with type="p" look normal, but the plot() result with type="l" look abnormal, having crossing lines? Then modify the code below (hint: modify the definition of x), so that the lines on the second plot do not cross.

The plot with type “l” creates a line plot, and since the x values aren’t sorted in order before we graph it creates this abnormal graph. If the sort function is applied then it will fix the issue.

n = 50  
set.seed(0)  
x = sort(runif(n, min=-2, max=2))  
y = x^3 + rnorm(n)  
plot(x, y, type="p")

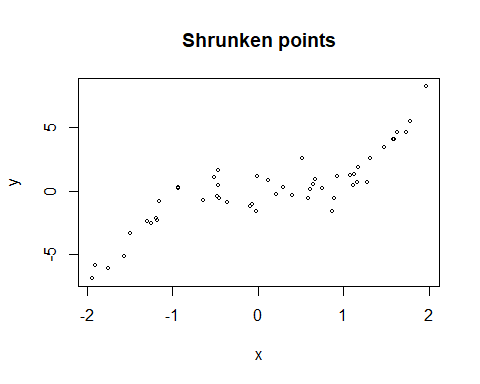


plot(x, y, type="l")

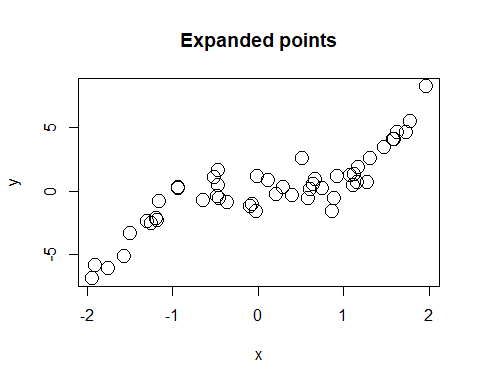


1. The cex argument can used to shrink or expand the size of the points that are drawn. Its default value is 1 (no shrinking or expansion). Values between 0 and 1 will shrink points, and values larger than 1 will expand points. Plot y versus x, first with cex equal to 0.5 and then 2 (so, two separate plots). Give titles “Shrunken points”, and “Expanded points”, to the plots, respectively.

plot(x, y, type="p", cex=0.5, main="Shrunken points")

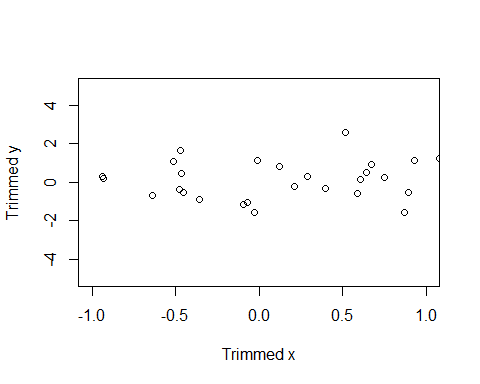


plot(x, y, type="p", cex=2, main="Expanded points")



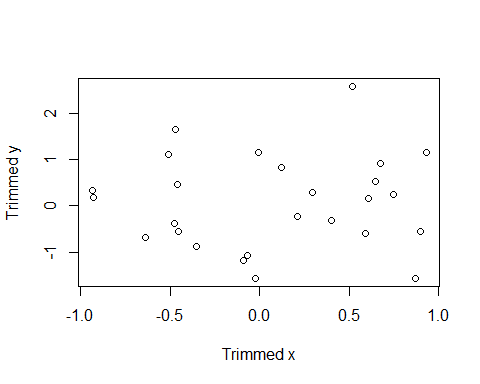
1. The xlim and ylim arguments can be used to change the limits on the x-axis and y-axis, respectively. Each argument takes a vector of length 2, as in xlim = c(-1, 0), to set the x limit to be from -1 to 0. Plot y versus x, with the x limit set to be from -1 to 1, and the y limit set to be from -5 to 5. Assign x and y labels “Trimmed x” and “Trimmed y”, respectively.

plot(x, y, xlim=c(-1, 1), ylim=c(-5, 5), xlab="Trimmed x", ylab="Trimmed y")



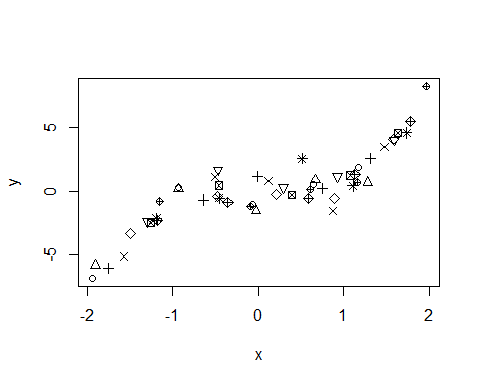
1. Again plot y versus x, only showing points whose x values are between -1 and 1. But this time, define x.trimmed to be the subset of x between -1 and 1, and define y.trimmed to be the corresponding subset of y. Then plot y.trimmed versus x.trimmed without setting xlim and ylim: now you should see that the y limit is (automatically) set as “tight” as possible. Hint: use logical indexing to define x.trimmed, y.trimmed.

index\_trimmed <- x >= -1 & x <= 1  
  
# Create trimmed x and y vectors  
x.trimmed <- x[index\_trimmed]  
y.trimmed <- y[index\_trimmed]  
  
# Plot y.trimmed versus x.trimmed without setting xlim and ylim  
plot(x.trimmed, y.trimmed, xlab="Trimmed x", ylab="Trimmed y")



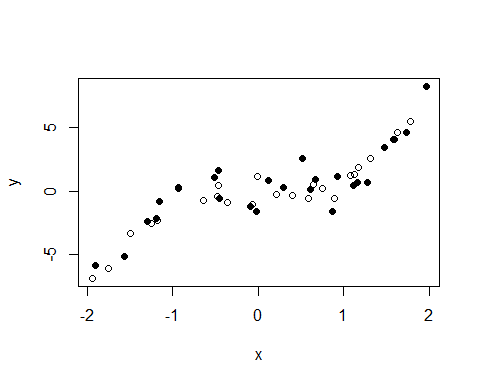
1. The pch argument, recall, controls the point type in the display. In the lecture examples, we set it to a single number. But it can also be a vector of numbers, with one entry per point in the plot. So, e.g.,

plot(x, y, pch = 1:10, xlab = "x", ylab = "y")



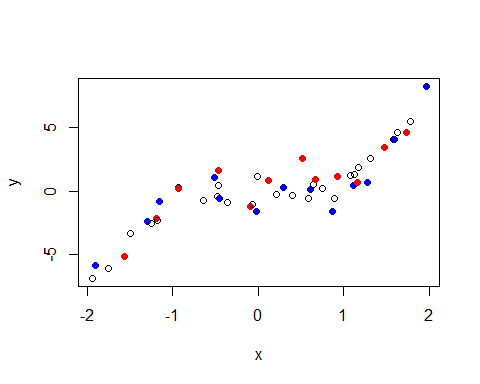
displays the first 10 point types. If pch is a vector whose length is shorter than the total number of points to be plotted, then its entries are recycled, as appropriate. Plot y versus x, with the point type alternating in between an empty circle and a filled circle.

point\_types <- rep(c(1, 19), length.out = n)  
  
plot(x, y, pch = point\_types, xlab = "x", ylab = "y")



1. The col argument, recall, controls the color the points in the display. It operates similar to pch, in the sense that it can be a vector, and if the length of this vector is shorter than the total number of points, then it is recycled appropriately. Plot y versus x, and repeat the following pattern for the displayed points: a black empty circle, a blue filled circle, a black empty circle, a red filled circle.

point\_colors <- rep(c("black", "blue", "black", "red"), length.out = n)  
  
plot(x, y, pch = point\_types, col = point\_colors, xlab = "x", ylab = "y")

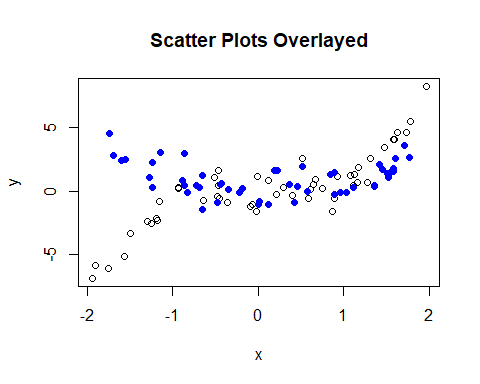


## Q5 Adding to plots

1. Produce a scatter plot of y versus x, and set the title and axes labels as you see fit. Then overlay on top a scatter plot of y2 versus x2, using the points() function, where x2 and y2 are as defined below. In the call to points(), set the pch and col arguments appropriately so that the overlaid points are drawn as filled blue circles.

x2 = sort(runif(n, min=-2, max=2))  
y2 = x^2 + rnorm(n)

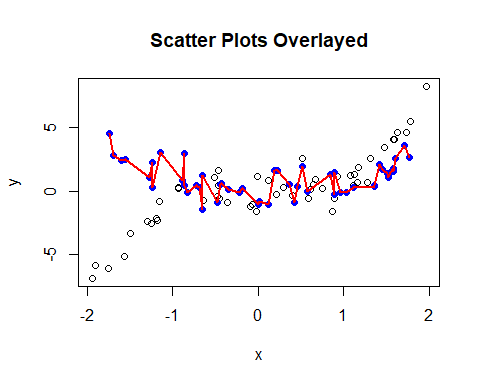
plot(x, y, type = "p", pch = 1, col = "black", xlab = "x", ylab = "y", main = "Scatter Plots Overlayed")  
  
points(x2, y2, pch = 19, col = "blue")



1. Starting with your solution code from the last question, overlay a line plot of y2 versus x2 on top of the plot (which contains empty black circles of y versus x, and filled blue circles of y2 versus x2), using the lines() function. In the call to lines(), set the col and lwd arguments so that the line is drawn in red, with twice the normal thickness. Look carefully at your resulting plot. Does the red line pass overtop of or underneath the blue filled circles? What do you conclude about the way R *layers* these additions to your plot?

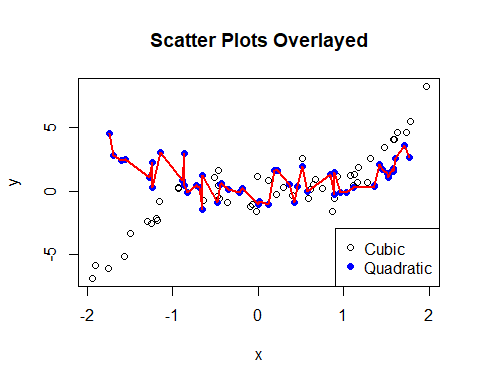
The red line passes over the top of the blue filled circles and it appears that whatever the most recent layer is that you code in is the layer that R codes Overlays on top of whatever the previous layers are.

plot(x, y, type = "p", pch = 1, col = "black", xlab = "x", ylab = "y", main = "Scatter Plots Overlayed")  
points(x2, y2, pch = 19, col = "blue")  
  
lines(x2, y2, col = "red", lwd = 2)



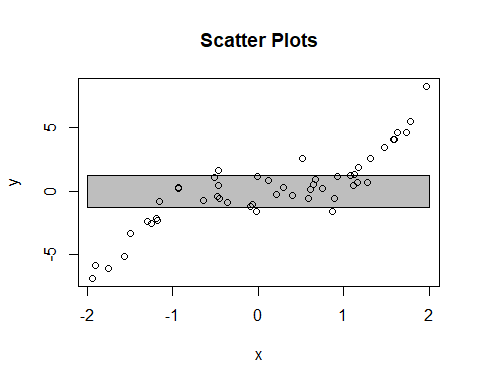
1. Starting with your solution code from the last question, add a legend to the bottom right corner of the the plot using legend(). The legend should display the text: “Cubic” and “Quadratic”, with corresponding symbols: an empty black circle and a filled blue circle, respectively. Hint: it will help to look at the documentation for legend().

plot(x, y, type = "p", pch = 1, col = "black", xlab = "x", ylab = "y", main = "Scatter Plots Overlayed")  
points(x2, y2, pch = 19, col = "blue")  
  
lines(x2, y2, col = "red", lwd = 2)  
legend("bottomright", legend = c("Cubic", "Quadratic"), pch = c(1, 19), col = c("black", "blue"))



1. Produce a plot of y versus x, but with a gray rectangle displayed underneath the points, which has a lower left corner at c(-2, qnorm(0.1)), and an upper right corner at c(2, qnorm(0.9)). Hint: use rect() and consult its documentation. Also, remember how layers work; call plot(), with type="n" or col="white" in order to refrain from drawing any points in the first place, then call rect(), then call points().

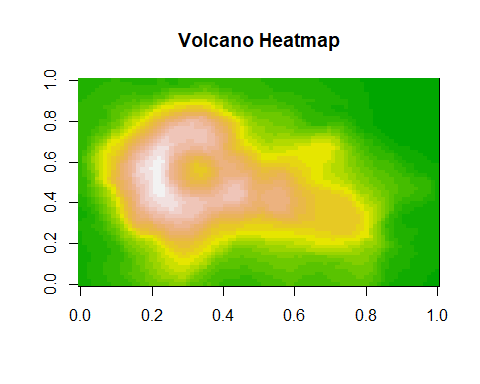
plot(x, y, type = "n", xlab = "x", ylab = "y", main = "Scatter Plots", col = "white")  
  
lower\_left <- c(-2, qnorm(0.1))  
upper\_right <- c(2, qnorm(0.9))  
rect(lower\_left[1], lower\_left[2], upper\_right[1], upper\_right[2], col = "gray")  
  
  
points(x, y, pch = 1, col = "black")



## Q6 Maungawhau volcano and heatmaps

1. The volcano object in R is a matrix of dimension 87 x 61. It is a digitized version of a topographic map of the Maungawhau volcano in Auckland, New Zealand. Plot a heatmap of the volcano using image(), with 25 colors from the terrain color palette.

image(volcano, col = terrain.colors(25), main = "Volcano Heatmap")

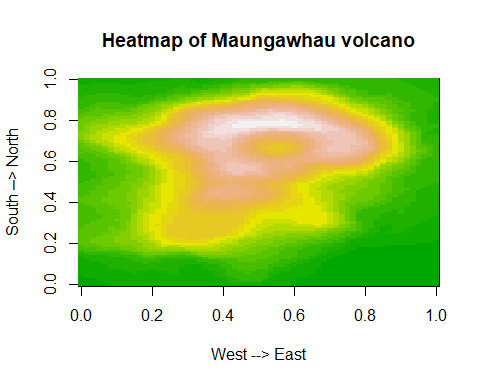


1. Each row of volcano corresponds to a grid line running east to west. Each column of volcano corresponds to a grid line running south to north. Define a matrix volcano.rev by reversing the order of the rows, as well as the order of the columns, of volcano. Therefore, each row volcano.rev should now correspond to a grid line running west to east, and each column of volcano.rev a grid line running north to south.

volcano.rev <- volcano[nrow(volcano):1, ncol(volcano):1]

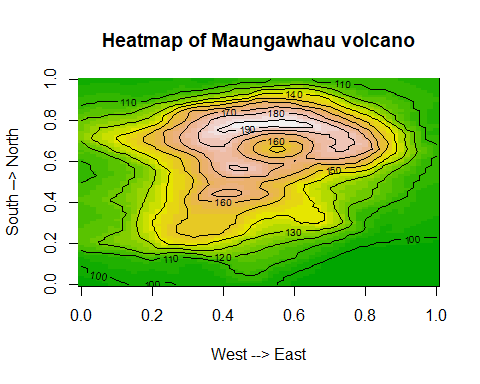
1. If we printed out the matrix volcano.rev to the console, then the elements would follow proper geographic order: left to right means west to east, and top to bottom means north to south. Now, produce a heatmap of the volcano that follows the same geographic order. Hint: recall that the image() function rotates a matrix 90 degrees counterclockwise before displaying it; and recall the function clockwise90() from the lecture, which you can copy and paste into your code here. Label the x-axis “West –> East”, and the y-axis “South –> North”. Title the plot “Heatmap of Maungawhau volcano”.

clockwise90 = function(a) { t(a[nrow(a):1,]) }   
  
volcano.rev <- clockwise90(volcano)  
  
image(volcano.rev, col = terrain.colors(25), main = "Heatmap of Maungawhau volcano", xlab = "West --> East", ylab = "South --> North")



1. Reproduce the previous plot, and now draw contour lines on top of the heatmap.

image(volcano.rev, col = terrain.colors(25), main = "Heatmap of Maungawhau volcano", xlab = "West --> East", ylab = "South --> North")  
contour(volcano.rev, add = TRUE)



1. The function filled.contour() provides an alternative way to create a heatmap with contour lines on top. It uses the same orientation as image() when plotting a matrix. Use filled.contour() to plot a heatmap of the volcano, with (light) contour lines automatically included. Make sure the orientation of the plot matches proper geographic orientation, as in the previous question. Use a color scale of your choosing, and label the axes and title the plot appropriately. It will help to consult the documentation for filled.contour().

filled.contour(volcano.rev, color.palette = topo.colors, main = "Heatmap of Maungawhau volcano", xlab = "West --> East", ylab = "South --> North", plot.axes = { axis(1); axis(2); })

