HW7_IS457_85

Mon Nov 12, 2018

Part 1. Regular Expressions Warmup (12 pt)

Basic Regex

(1) Find words in test vector test_1 which start with lowercase 'e' using grep. What does grep return? (2 pt)

test_1 = c("wireless", "car", "energy", "2020", "elation", "alabaster", "Endoscope")

Ans :

grep("^e",test_1, value = TRUE)

```
[1] "energy"  "elation"
```

grep("^e",test_1)

```
[1] 3 5
```

If the value attribute is set to TRUE **grep()** function returns the elements matched by the pattern else it returns the location of those elements.

(2) Find characters which can be a password with ONLY letters and numbers. (1 pt)

test_2 = c("bb1l9093jak", "jackBlack3", "the password", "!h8p4$$w0rds", "wiblewoble", "ASimpleP4ss", "d0nt_use_this")

Ans :

grep("^[A-zA-Z]+[0-9]",test_2,value = TRUE)

```
[1] "bb1l9093jak"   "jackBlack3"    "ASimpleP4ss"
[4] "d0nt_use_this"
```

grep("^[A-zA-Z]+[0-9]", test_2)

```
[1] 1 2 6 7
```

(3) Find Email addresses of the form letters@letters.xxx (1 pt)
Here "xxx" means any alpha numeric characters with length of 3.
Letters can be any alpha numeric characters of any length. Letters before "@" can also be along with the underscore.

test_3 = c("wolf@gmail.com", "little_red_riding_hood@comcast.net", "spooky woods5@swamp.us", "grandma@is.eaten", "the_ax@sbcglobal.net")

Ans :

grep("^[a-zA-Z0-9+_]+@[a-zA-Z0-9]+\\.[a-zA-Z0-9]{3}$",test_3,value = TRUE)

```
[1] "wolf@gmail.com"
[2] "little_red_riding_hood@comcast.net"
[3] "the_ax@sbcglobal.net"
```

grep("^[a-zA-Z0-9+_]+@[a-zA-Z0-9]+\\.[a-zA-Z0-9]{3}$",test_3)

```
[1] 1 2 5
```

Capture Groups
This is a method to grab specific text within a given match.
This is a very useful technique to get specific bits of text quickly.
We will use a series of steps to extract the domain names from properly formatted email addresses.

(4) Use regexec() to execute a regular expression to find properly formatted email addresses in test_3. Save it as test_3_reg_exec.
This time, we will allow domain names of the form letters.letters. i.e. addresses like 'test.us' are now allowed.(1 pt)

Ans :

test_3_reg_exec = regexec("^[a-zA-Z0-9+_]+@[a-zA-Z0-9]+\\.[a-zA-Z0-9]{3}$",test_3)

test_3_reg_exec

```
[[1]]
[1] 1
attr(,"match.length")
[1] 14
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

(5) What type of object is test_3_reg_exec? What type of information does it contain? (2 pt)

Ans :

typeof(test_3_reg_exec)

```
[1] "list"
```

test_3_re_exec is of type list. It contains information for the match pattern for every element in the test_3 vector. Its length is equal to the length of test_3. It contains information such as whether the current element is matched by the pattern. If it is matched 1 is stored else -1. The information regarding the length of the match is also stored in match.length and also the type of the matched element.

(6) Use regmatches() to get a list of the text matches from test_3_reg_exec. Call this 'reg_match_list'. What is the class of reg_match_list? and what is the format? (4 pt)

Ans :

reg_match_list = regmatches(test_3, test_3_reg_exec)

reg_match_list

```
[[1]]
[1] "wolf@gmail.com"

[[2]]
[1] "little_red_riding_hood@comcast.net"

[[3]]
character(0)

[[4]]
character(0)

[[5]]
[1] "the_ax@sbcglobal.net"
```

For each element in the test vector it stores the element in its list if it matches the pattern else it stores the element of size 0.

(7) Use reg_match_list() to get a vector of matched domain names in Q6. Name this vector 'domain names'. (3 pt)

Ans :

domain_names = character()

```
for (i in 1:length(reg_match_list)) {
  domain_names = c(domain_names, reg_match_list[[i]])
}
domain_names
```

```
[1] "wolf@gmail.com"
[2] "little_red_riding_hood@comcast.net"
[3] "the_ax@sbcglobal.net"
```

Part 2. Aesop's Fables

We will now look at a text file of aesop's fables. We will first need to process the data to get it into a form we can use. We can then look at interesting properties like the number of words in each fable.

(8) Use readLines() to load the aesop fable data from the aesop-fables.txt file you can find in moodle. Name it aesop_data. MAKE SURE to use the encoding 'UTF-8'. (1 pt)

Ans :

aesop_data = readLines("C://Users//komal//Desktop//UIUC Coursework//Intro to Data Science//Data//aesop-fables.txt")

aesop_data

```
[1] "ï»¿The Project Gutenberg EBook of Aesop's Fables, by Aesop"
[2] ""
[3] "This eBook is for the use of anyone anywhere at no cost and with"
[4] "almost no restrictions whatsoever.  You may copy it, give it away or"
[5] "re-use it under the terms of the Project Gutenberg License included"
[6] "with this eBook or online at www.gutenberg.net"
```

(9) What is the format of aesop_data? How is the book formatted? How might we use this formatting to our advantage? (3 pt)

Ans :

The book starts with the introduction of the books, followed by the contents in the book, the actual fables along with lessons and lastly the license for the text. The books follows a consistent structure of whitespaces between the different contents, specially between different fables. This might help us in finding the location for the start of each fables.

(10) Let's take a look of fables using the table of contents. First, find the start point and end point of the table of content using grep() and specific header names in the file. Then subset only those lines which are from the table of contents. Save the fable titles in a character vector. Finally, count the number of non-empty lines in your subset. Print out the number.(5 pt)

Ans :

```
lines = grep("CONTENTS|LIST OF ILLUSTRATIONS", aesop_data)

f = lines[1]+1

l = lines[2]-1

fable_titles = aesop_data[f:l]

fable_titles = fable_titles[fable_titles!=""]

number = length(fable_titles)

number
```

```
[1] 284
```

(11) Separate out all the fables in the file. The process is similar to Q10, find the start point and end point. (3 pt) Call this fable_data. Here do not remove the titles or empty lines.

NOTE: Notice that, in this text file, "AESOP'S FABLES" is sometimes shown as "Ã□SOP'S FABLES", after you find the lines you want to extract information from, make sure you read the text carefully. (if you need to use it, just a simple copy or paste will work).

Ans :

```
start = grep("THE FOX AND THE GRAPES", aesop_data)

start = start[length(start)]
```

end = grep("ILLUSTRATIONS", aesop_data)

end = end[length(end)]

fable_data = aesop_data[start:(end-1)]

length(fable_data)

```
[1] 4995
```

(12) How do you know when a new fable is starting? (1 pt)

Ans :

Each new fable starts with all capital letters title and two blank lines followed by fable text.

(13) We will now transform this data to be a bit easier to work with. Fables always consist of a body which contains consequtive non-empty lines which are the text of the fable. This is sometimes followed by a 'lesson' (summary) statement whose lines are consecutive but indented by four spaces. We will create a list object which contains information about each fable.

Get the start positions of each fable in fable_data (how you answer Q12?). (3 pt)
Hint: Look at the title vector you created in Q10, what does it include (other than letters?)

Ans :

pos = NULL

lpos = NULL

```
for (i in 1:number) {
  str = paste("^",fable_titles[i],"$", sep = "")
  pos = c(pos,grep(str,fable_data))
}
```

lpos = grep("^ {3}", fable_data)

lpos

```
  [1]   26   47   48   65  118  204  221  291  313  350
 [11]  385  528  561  604  605  650  651  672  726  744
 [21]  774  837  856  882  913  927  984 1035 1069 1167
 [31] 1182 1252 1323 1340 1423 1610 1631 1682 1719 1786
 [41] 1866 1971 1986 2004 2024 2093 2110 2153 2224 2271
 [51] 2386 2405 2466 2481 2496 2528 2561 2579 2639 2640
 [61] 2713 2772 2877 2946 2947 3022 3054 3204 3266 3267
 [71] 3286 3307 3475 3496 3516 3517 3535 3552 3568 3648
 [81] 3649 3687 3728 3830 3846 3867 3868 3889 3929 3930
 [91] 4251 4272 4304 4385 4386 4404 4439 4554 4555 4794
[101] 4978
```

(14) Transform the fables into an easy-to-reference format (data structure). First create a new list object named 'fables'. Each element of the list is a sublist that contains two elements ('text' and 'lesson'). For each fable, merge together the separate lines of text into a single character element. That is, one charactor vector (contains all sentences) for that fable. This will be the 'text' element in the sublist for that fable. If the fable has a lesson, extract the statement into a character vector (also remove indentation). This will be the 'lesson' element in the sublist for that fable. (10 pt)

Ans :

```
fables = list()

for (i in 1:length(pos)) {
 if(i!=length(pos)) {
  if(lpos[1]>pos[i+1]) {
   text = paste(fable_data[(pos[i]+1):(pos[i+1]-1)], collapse = " ")
   lesson = NULL
   fables[[i]] = list("text"=text,"lesson" = lesson)
  }
  else {
   text = paste(fable_data[(pos[i]+1):(lpos[1]-1)], collapse = " ")
   lesson = paste(fable_data[(lpos[1]):(pos[i+1]-1)], collapse = " ")
   fables[[i]] = list("text"=text,"lesson" = lesson)
   while(TRUE&&length(lpos)!=0)
   {
    if(lpos[1]<pos[i+1]) {
     lpos = lpos[-1]
    }
    else {
     break
    }
   }
  }
 }
 else {
  if (length(lpos)!=0) {
   text = paste(fable_data[(pos[i]+1):(lpos[1]-1)], collapse = " ")
   lesson = paste(fable_data[(lpos[1]):length(fable_data)], collapse = " ")
   fables[[i]] = list("text"=text,"lesson" = lesson)
  }
  else {
   text = paste(fable_data[(pos[i]+1):length(fable_data)], collapse = " ")
   lesson = NULL
   fables[[i]] = list("text"=text,"lesson" = lesson)
  }
 }
}
```

(15) How many fables have lessons? (2 pt)

Ans :

```
n = NULL
```

```
for (i in 1:(length(fables)-1)) {
  n = c(n, !is.null(fables[[i]]$lesson))
}
sum(n)
```

```
[1] 89
```

Based on the above output 89 fables have lessons.

(16) Add a character count element named 'chars' and a word count element named 'words' to each fable's list. (3 pt) Use the following function to count words:

```
word_count = function(x) {
  return(lengths(gregexpr("\\W+", x)) + 1)  # words separated by space(s)
}
```

Ans :

```
char_count = function(x) {
  return(lengths(gregexpr("\\S", x)) + 1)  # words separated by space(s)
}

word = integer()

char = integer()

for (i in 1:length(fables)) {
  word = c(word, word_count(fables[i]))
  char = c(char, char_count(fables[i]))
}

sum(char)
```

```
[1] 159959
```

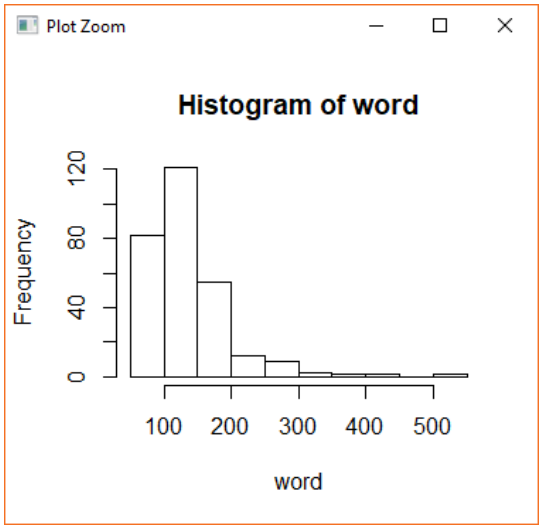
```
sum(word)
```

```
[1] 38438
```

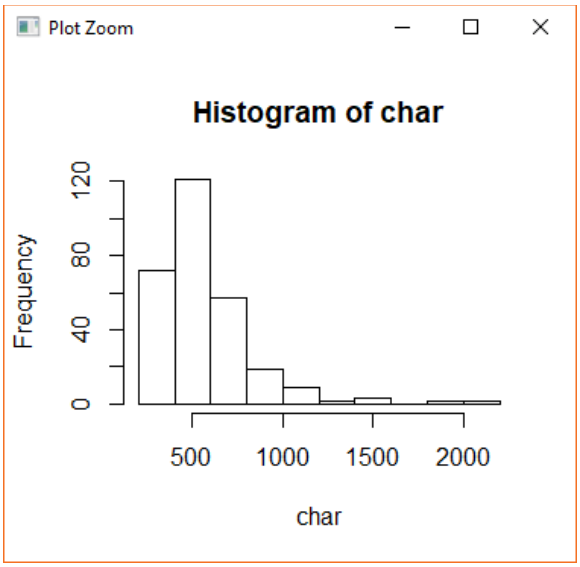
(17) Create separate histograms of the number of characters and words in the fables. (10 pt) Recall the graphics techniques you learned before.

Ans :

```
hist(word)
```



hist(char)



(18) Let's compare the fables with lessons to those without. Extract the text of the fables (from your fables list) into two vectors. One for fables with lessons and one for those without. (4 pt)

Ans :

```
fable_with = vector()

fable_without = vector()

for(i in 1:(length(fables)-1)) {
  if(length(fables[[i]]$lesson)==0){
    fable_without = c(fable_without,fables[[i]]$text)
  }
  else {
    fable_with = c(fable_with,fables[[i]]$text)
  }
}
```

head(fable_with, 1)

```
[1] "  A Man and his Wife had the good fortune to possess a Goose which laid a Gold
en Egg every day. Lucky though they were, they soon began to think they were not ge
tting rich fast enough, and, imagining the bird must be made of gold inside, they d
ecided to kill it in order to secure the whole store of precious metal at once. But
when they cut it open they found it was just like any other goose. Thus, they neith
er got rich all at once, as they had hoped, nor enjoyed any longer the daily additi
on to their wealth. "
```

head(fable_without, 1)

```
[1] "  A hungry Fox saw some fine bunches of Grapes hanging from a vine that was tr
ained along a high trellis, and did his best to reach them by jumping as high as he
could into the air. But it was all in vain, for they were just out of reach: so he
gave up trying, and walked away with an air of dignity and unconcern, remarking, \"
I thought those Grapes were ripe, but I see now they are quite sour.\"    "
```

(19) Remove all non-alphabetic characters (except spaces) and change all characters to lowercase. (3 pt)

Ans :

```
fable_with = tolower(as.character(fable_with))

fable_without = tolower(as.character(fable_without))

fable_with = gsub("[^a-z ]","", fable_with)

fable_without = gsub("[^a-z ]","", fable_without)
```

head(fable_with, 1)

```
[1] "  a man and his wife had the good fortune to possess a goose which laid a gold
en egg every day lucky though they were they soon began to think they were not gett
ing rich fast enough and imagining the bird must be made of gold inside they decide
d to kill it in order to secure the whole store of precious metal at once but when
they cut it open they found it was just like any other goose thus they neither got
rich all at once as they had hoped nor enjoyed any longer the daily addition to the
ir wealth "
```

head(fable_without, 1)

```
[1] "  a hungry fox saw some fine bunches of grapes hanging from a vine that was tr
ained along a high trellis and did his best to reach them by jumping as high as he
could into the air but it was all in vain for they were just out of reach so he gav
e up trying and walked away with an air of dignity and unconcern remarking i though
t those grapes were ripe but i see now they are quite sour    "
```

(20) Split the fables from Q19 by blanks and drop empty words. Save all the split words into a single list for each type of fable. Name them token_with_lessons and token_without_lessons. Print out their lengths. (5 pt)

Ans :

```
token_with_lessons = list()
```

```
token_without_lessons = list()

tokens = character()

for(i in 1:length(fable_with)) {
  tokens = c(tokens, unlist(strsplit(fable_with[i], " ")))
}

tokens = tokens[tokens!=""]

token_with_lessons = list("tokens" = tokens)

tokens = character()

for(i in 1:length(fable_without)) {
  tokens = c(tokens, unlist(strsplit(fable_without[i], " ")))
}

tokens = tokens[tokens!=""]

token_without_lessons = list("tokens" = tokens)

length(token_with_lessons)
```

```
[1] 1
```

```
length(token_with_lessons[[1]])
```

```
[1] 10913
```

```
length(token_without_lessons)
```

```
[1] 1
```

```
length(token_without_lessons[[1]])
```

```
[1] 24884
```

(21) Calculate the token frequency for each type of fable. (2 pt)

Ans :

```
freq_with = table(token_with_lessons)

head(freq_with, 10)
```

```
token_with_lessons
          a  abandoning        able       about
        411           1           5          25
     abroad       abuse    accepted accompanied
          1           1           1           1
accordingly     account
          2           2
```

freq_without = table(token_without_lessons)

head(freq_without, 10)

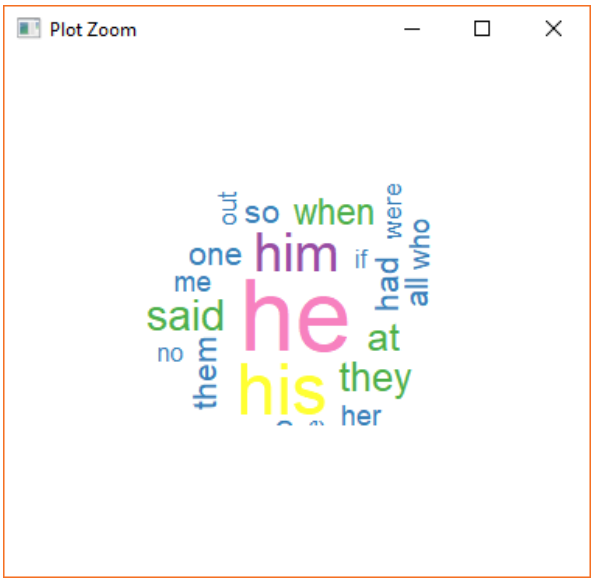
```
        a abandoned     abated       abed       able
      818         2          1          1          4
    abode      about      above    abroad   absence
        1        71          4          1          2
```

(22) Carry out some exploratory analysis of the data and token frequencies. For example, find the words associated fables with lessons. What are distribution patterns for term frequencies? Use wordcloud function in wordcloud package to plot your result. What are your observations? (10 pt)
Hint: you'll want to include important words but not stopwords (we provided a list below) into your plot. What are important words? we have token_with(out)_lessons from Q20, think relative high frequency (use quantile() to help you decide). so, start by creating a table of token frequency; filter out low frequency words and stopwords.

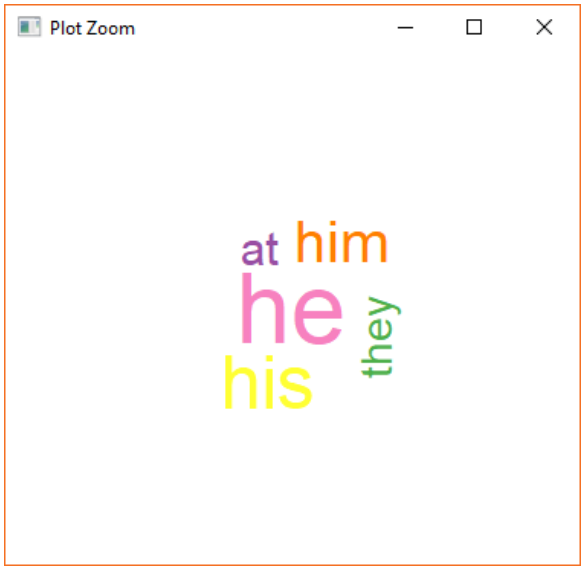
Ans :

```
mask_word = c("by", "as", "a","their", "which", "have", "with", "are", "been", "will", "we", "not",
              "has", "this", "or", "from", "on", "i", "the","is","it","in","my","of","to",
              "and","be","that","for","you","but","its","was")

install.packages("wordcloud"); library(wordcloud); library(RColorBrewer)
plot_word_cloud = function(tokens) {
  tokens_frequency = table(tokens)
  tokens_90 = tokens_frequency[tokens_frequency > quantile(tokens_frequency, 0.9)]
  word_cloud = data.frame( freq = as.numeric(tokens_90), words = names(tokens_90))
  word_cloud = word_cloud[!word_cloud$words %in% mask_word,]
  wordcloud(words = word_cloud$words, freq = word_cloud$freq, max.words = 100, min.freq = 80,
          rot.per = 0.3, random.order = F,colors = brewer.pal(8, "Set1"))
}
plot_word_cloud(token_without_lessons)
```



plot_word_cloud(token_with_lessons)



From the output we can observe that the major words used are the masculine pronouns and also prepositions