Data-607 Assignment 3

Umais Siddiqui

September 16, 2017

Git hub : Github link assignment 3

RPub Link:Rpub link

Chapter 8: Regular Expressions and Essential String Functions

In this assignment we will be exploring the use of regular expressions so that we are able to extract useful information and also be able to manipulate data by making use of other useful string functions.

Question 3: Copy the introductory example. The vector name Stores the exracted names

```
# First we create a variable that stores the raw data
raw.data <- "555-1239Moe Szyslak(636) 555-0113Burns, C. Montgomery555-6542Rev. Timothy Lovejoy555 8904N

# Extract information


name <- unlist(str_extract_all(raw.data, "[[:alpha:]., ]{2,}"))
name
```

```
## [1] "Moe Szyslak"          "Burns, C. Montgomery" "Rev. Timothy Lovejoy"
## [4] "Ned Flanders"         "Simpson, Homer"       "Dr. Julius Hibbert"
```

(a) Use the tools of this chapter to rearrange the vector so that all elements conform to the standard first_name last_name.

```
#In this step we will be spliting the names by using a comma as a delimiter.
names<-str_split(name,"(\\,)")
names
```

```
## [[1]]
## [1] "Moe Szyslak"
##
## [[2]]
## [1] "Burns"          " C. Montgomery"
##
## [[3]]
## [1] "Rev. Timothy Lovejoy"
##
## [[4]]
## [1] "Ned Flanders"
##
## [[5]]
## [1] "Simpson" " Homer"
##
## [[6]]
```

```
## [1] "Dr. Julius Hibbert"
```

In this step we are basically defining a function that will produce the names in the format first_name last_name by using the tools provided in the Chapter.

```
getFirstNamesLastNames=function()
{
#Initializing a Vector to store FirstName LastName
FirstNamesLastNames=c()
#Looping through the names vector to search for names that are not in correct format
for(i in 1:length(names))
{

 len= length(names[[i]])

 #if length of the vector item is greater than 1 then the name needs to be rearranged
  if(len>1)
  {
    fullName=rev(names[[i]])
    #Rearranging the name and massaging the data
    FirstNamesLastNames[i]=str_trim(str_c(str_extract(fullName[1],"[:alpha:]+")," ",fullName[2]))
  }
 else{
   #These names are already in correct order but we are replacing the extraneous characters.
     FirstNamesLastNames[i]=str_trim(str_replace(names[[i]],pattern="[:alpha:]+\\.",replacement=""))
 }
}
# At this point the names are in correct format so we could return them
return(FirstNamesLastNames)
}
```

Now calling the function getFirstNamesLastNames() and displaying the results

```
listOfFirstNameLastNames=getFirstNamesLastNames();
listOfFirstNameLastNames
```

```
## [1] "Moe Szyslak"     "C Burns"         "Timothy Lovejoy" "Ned Flanders"
## [5] "Homer Simpson"   "Julius Hibbert"
```

(b) Construct a logical vector indicating whether a character has a title . (i.e Rev. and Dr.)

Answer: We can make use of the function str_detect from the chapter that will detect the pattern and construct a logical vector and then print out the names of the actors that meet the criteria.

```
#Constructing the logical vector and then printing it out.
IsTitleVector=str_detect(name, "[:alpha:]{2,}\\.")
IsTitleVector
```

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

```
name[IsTitleVector]
```

```
## [1] "Rev. Timothy Lovejoy" "Dr. Julius Hibbert"
```

(c) Construct a logical vector indicating whether a character has a middle name.

Answer: Same as part b we can construct the logical vector using the str_detect function and then print out the logical vector as well as the names that conform to the logical vector.

```
IsInitialVector=str_detect(name, "[:space:][:alpha:]\\.")
IsInitialVector
```

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

```
name[IsInitialVector]
```

```
## [1] "Burns, C. Montgomery"
```

Question 4: Describe the types of strings that conform to the following regular expressions and construct an example that is matched by the regular expression.

(a)[0-9]+\$

In the following regular expression it is expected to match one or more digits and then followed by exactly "$" sign

```
 #Will Conform
 sample <- "12345$"
regex = "[0-9]+\\$"
str_extract(sample, regex)
```

```
## [1] "12345$"
```

```
#Will not conform
sample <- "34567a$"
str_extract(sample, regex)
```

```
## [1] NA
```

(b) "\b[a-z]{1,4}\b"

This regex will extract the first four lower case letters of the word

```
 #Will Conform
 sample <- "abcd efgh"
regex = "\\b[a-z]{1,4}\\b"
str_extract(sample, regex)
```

```
## [1] "abcd"
```

```
 sample <- "abCD efgh"

 str_extract(sample, regex)
```

```
## [1] "efgh"
```

(c)".*?\.txt$" This regular expression will return the .txt and anything that is preceding .txt string. .txt must be the last part of the string.

```
 #Will Conform
 sample <- "Address.txt"
regex = ".*?\\.txt$"
str_extract(sample, regex)
```

```
## [1] "Address.txt"
```

```
 #Will Not Conform
 sample <- "address.txtabcdef"

 str_extract(sample, regex)
```

```
## [1] NA
```

```
 #Will Not Conform
  sample <- "address.csv"

 str_extract(sample, regex)
```

```
## [1] NA
```

(d) \d{2}/\d{2}/\d{4}

This particular regex is looking for 2 digits followed by a / and then 2 digits followed by another / then 4 digits. Basically it looks like the pattern for a date format MM/DD/YYYY or DD/MM/YYYY but any digits will conform and it does not validate a correct date.

```
 #Will Conform
 sample <- "22/10/2014"
regex = "\\d{2}/\\d{2}/\\d{4}"
str_extract(sample, regex)
```

```
## [1] "22/10/2014"
```

```
 #Will Conform
 sample <- "60/60/2014"

 str_extract(sample, regex)
```

```
## [1] "60/60/2014"
```

```
  #Will not Conform
  sample <- "ad/23/2014"

 str_extract(sample, regex)
```

```
## [1] NA
```

(e) "<(.+?)>.+?</\1>"

It is basically extracting the begin an end tags like in an HTML or XML document also using a back reference to make sure that the begin and end tag match. Notice how in the first example it picks up the second tag rather than the first. In the third example there is no match because the begin and end tags do not match.

```
#Conforms
regex = "<(.+?)>.+?</\\1>"
```

```
sample = "<Title>Sometext</head><body>Sometext</body>"
str_extract(sample, regex)
```

```
## [1] "<body>Sometext</body>"
```

```
#coforms
sample = "<html>Sometext</html>"
str_extract(sample, regex)
```

```
## [1] "<html>Sometext</html>"
```

```
#Does not Conform
sample = "<html>Sometext</htm>"
str_extract(sample, regex)
```

```
## [1] NA
```

Question 9: The Following Code hides a secret message . Crack it with R and regular expressions. Hint: Some of the characters are more reealing than others! The Code snippet is also available in the materials at www.r-datacollection.com.

Answer: The code can be cracked by using the regex "[[:upper:].]+" which is basically looking for all uppercase letters in the code and periods. We then use unlist, paste and str_replace_all functions to convert the results retrieved from str_extract_all to a string and we can then dislay the message. So, basically we are making use of tools that were learned in Chapter 8 to crack the code.

```
encryptedCode = "clcopCow1zmstc0d87wnkig7OvdicpNuggvhryn92Gjuwczi8hqrfpRxs5Aj5dwpn0Tanwo Uwisdij7Lj8kpf
regex = "[[:upper:].]+"
#The cracked code is below
str_replace_all(paste(unlist(str_extract_all(encryptedCode, regex)),collapse=""),pattern="[\\.]+",repla
```

```
## [1] "CONGRATULATIONS YOU ARE A SUPERNERD"
```