The Strength of Strings

Austin Chiang

November 29, 2017

The Strength of Strings

Introduction

With more and more data becoming accessible to the general public, it becomes imperative to understand how to use the information in order to be able to properly anaylze it. In a class setting, data is often prepared for us and we are able to easily access all of the information to achieve our goal. However, in the real world, most sets of data will require time and effort to pre-process so that we can import them and analyze them with ease. Thus, it is extremely important that we learn how to prepare data and clean them. One thing that we may need to work with is working with strings. Strings usually contain unstructured or semi-structured data so being able to manipulate them will allow us to find patterns and obtain the data we want in an organized manner.

We will be working with the *stringr* package so we will need to load the packages into R explicitly. Make sure to install the package if you dont have it first! Below is all the packages you need to have and call before starting so make sure to install them if you dont already have them!

```
#install.packages('stringr')
#install.packages('htmlwidgets')

library(htmlwidgets)
```

```
## Warning: package 'htmlwidgets' was built under R version 3.4.2
```

```
library(stringr)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.2
```

String Basics

Strings can be easily created by using single or double quotes and there is no difference in using one over the other. However, if you want to have a quote “” inside your string, use `. Below is an example of string construction, try it yourself to see how it works.

```
string <- "Hello World"

string1 <- 'Hello World'

string2 <- 'Hello "world"'
```

Strings can also be stored in character vectors like done below:

```
string_vector <- c("apple", "orange", "banana", "cat", "dance")
string_vector
```

```
## [1] "apple"  "orange" "banana" "cat"    "dance"
```

There are also countless of special characters that help you create the string that you want. For example, “” will create a new line in the string while “” will create a tab. You can view all of the characters by using the help panel with ?’“’.

Stringr

The *stringr* package contains many functions to work with strings and they allow you to manipulate strings so that you are able to view them and understand the dataset that they are a part of. They are very similar and they all have a str_ prefix so they are quite intuitive. Typing the prefix will allow you to see all of the functions available to you. Using the *stringr* package, we are able to do things like figure out the number of characters in a string:

```
str_length("Gaston Sanchez Trujillo")
```

```
## [1] 23
```

The *stringr* package also includes a tool that allows us to combine multiple strings. We do this by using the str_c() function in the package. We use the strings we want to combine as arguments and then include a sep argument that indicates how we want the string to be separated. In this case, we want the strings to be separated by a space so our sep argument is a string that is just a space. The function is also vectorized so we can use it on vectors. Run the code below to see how strings are combined:

```
str_c('Gaston', 'Sanchez', 'Trujillo', sep = ' ')
```

```
## [1] "Gaston Sanchez Trujillo"
```

```
str_c('Hello', c('Gaston', 'Professor Trujillo'), sep = ' ')
```

```
## [1] "Hello Gaston"             "Hello Professor Trujillo"
```

The *stringr* package also includes a way to subset strings through the str_sub() function. We can use str_sub() by inputting start and end arguments that tell the function which positions of the string to return. See how str_sub() is used in the examples below:

```
str_sub("Gaston Sanchez Trujillo", start = 1, end = 6)
```

```
## [1] "Gaston"
```

Regex

Regex or regular expressions are a kind of language that will allow us to describe patterns in strings so that we can manipulate and organize them better. There are many different patterns but it is a very useful tool and will show when you understand them. The simplest forms use str_view(), a function that take in a character vector and regex and shows how they match. We will demonstrate how these functions work below:

```
regex_str <- c("There", "Their", "They're")
str_view(regex_str, "The")
```

There

Their

They're

```
str_view(regex_str, ".h.")
```

There

Their

They're

In the example we used the pattern ".h." as an argument for the str_view function. This is one of the regular expressions that we can use in order to describe a pattern in the strings. The "." in this case represents any single character except for line and paragraph breaks. If you want to know about all of the regular expressions, you can reference this site to learn more: https://help.libreoffice.org/Common/List_of_Regular_Expressions.

More on Stringr

With regex, we have access to a larger array of stringr functions that will allow us to determine patterns in strings and do things from extracting the strings to replacing the matches.

Detection

We can detect whether or not a character vector has a pattern using the str_detect() function. THe function allows us to pass in a character vector and a regular expression to return a logical vector that returns TRUE if the string has the regular expression inside and FALSE otherwise for each string in the vector. For example, if we want to find out which strings in our character vector regex_str had the letter "i" in them, we could use the str_detect to figure it out:

```
str_detect(regex_str, "i") #Only 'Their' has an 'i' in it
```

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

```
str_detect(regex_str, '[aeiou]') #Checks if any string in the character vector has a vowel.
```

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

We can also use the function Str_count(), which takes in the same arguments as str_detect() but instead returns the number of mathes there are in a string rather than whether or not there is a match.:

```
str_count(regex_str, '[aeiou]') #counts number of vowels in each string
```

```
## [1] 2 2 2
```

Extraction

For this portion we will be using a vector of colors that we define below:

```
colors <- c('red', 'blue', 'red', 'green', 'orange', 'red', 'black', 'orange', 'blue', 'maroon')
```

Using str_extract() in the *stringr* package, we are able to pass in a character vector and a regular expression in order to have the function return all the strings that contain the expression. So, if we wanted to extract all of the colors that are red in them, we could use the function like below:

```
str_extract(colors, 'red')
```

```
##  [1] "red" NA    "red" NA    NA    "red" NA    NA    NA    NA
```

However, str_extract() only works with more simple data structures and for data structures like matrices, you will need to use str_extract_all() for it to work.

Splitting

The *stringr* package also has a function that allows us to split up strings into different parts. For example, we could divide a sentence into its words.

```
sentence <- 'Gaston Sanchez Trujillo is the professor for a stats course at UC Berkeley'

str_split(sentence, " ")
```

```
## [[1]]
##  [1] "Gaston"    "Sanchez"   "Trujillo"  "is"        "the"
##  [6] "professor" "for"       "a"         "stats"     "course"
## [11] "at"        "UC"        "Berkeley"
```

We divided the sentence up by the space so that everytime we reached a space, the function separated the next word until the next space appears. Although str_split() is a realtively simple function, it is an extremely useful tool that we can use for manipulating strings.

Replacement

Finally, str_replace)() allows us to replace matches in strings with newer strings. This is an extremely useful tool that allows us to change strings quickly and effiencently. For example, if we realized that in the vector of strings that had the same word spelled wrong every single time, we could use str_replace() to fix all the strings so that there is no spelling errors in the vector. Below is an example of the usage of str_replace():

```
str_replace(sentence, " ", "-")
```

```
## [1] "Gaston-Sanchez Trujillo is the professor for a stats course at UC Berkeley"
```

As you can see, we replaced the spaces in the sentence that we defined above with a dash. However, str_replace() only recognizes the first match and in order to replace all the matches, we must use str_replace_all():

```
str_replace_all(sentence, ' ', '-')
```

```
## [1] "Gaston-Sanchez-Trujillo-is-the-professor-for-a-stats-course-at-UC-Berkeley"
```

Example

For this, we will be using the words vector defined in the *stringr* package and it is simply a vector of words. Let us pretend that these words were the answers by many different people to a question that I had asked in a survey and I want to see how many times each vowel was used.

To do this we can use the str_count() function to count the number of times each vowel appears in a word and sum them all up. We can use a for loop to do this.

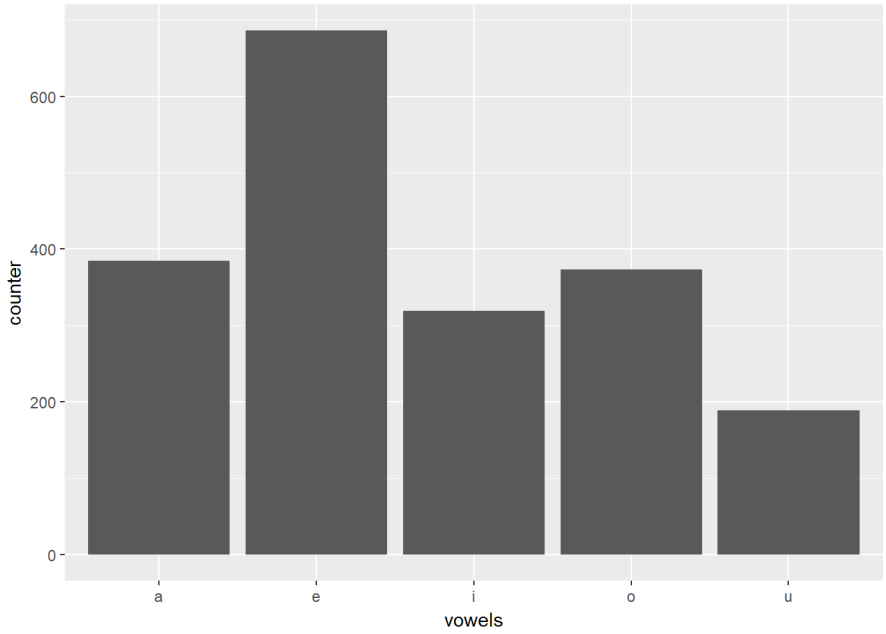
```
vowels <- c("a", "e", "i", "o", "u")
counter <- c()

for(i in 1:length(vowels)){
  counter[i] = sum(str_count(words, vowels[i]))
}
```

We defined a vector vowels and an empty vector counter so that we can hold the values we get from the sum of the str_count() function on the words vector with each vowel. Then, we have achieved our goal of getting the data and we can showcase our findings using plots.

```
vowel_counts <- data.frame(vowels, counter, stringsAsFactors = FALSE)
ggplot(data = vowel_counts, mapping = aes(x = vowels, y = counter)) + geom_bar(stat = "identity")
```

```
## Warning in plyr::split_indices(scale_id, n): '.Random.seed' is not an
## integer vector but of type 'NULL', so ignored
```



New Material

Above is all material covered in Stat 133 at UC Berkeley, however it is in your best interest to expand on this topic. I want to cover other useful functions that are part of the *stringr* package that were not covered in class. One is the apropos() function. This function is an incredibly helpful tool that allows you to view all the objects in the global environment. You can pass in a regular argument to narrow down the search so that you can find an object or function that you forgot about. This is an incredibly useful tool that helps everyone that uses R.

Another function is the dir() function. This function lists all the files in a directory and can take in a regular expression in order to return only the file names that have a match. This is useful because it can help if you want to find all of the files that are of a certain type in your directory if it is big, such as all of your Rmd fles or all of your html files.

Lastly, I want to tell you about the *stringi* package which is a more advanced version of the *stringr* package. It contains many more functions and tools in comparison to the *stringr* package and will allow you to have more flexibility when working with strings. The documentation for the package is in the link provided: https://www.rdocumentation.org/packages/stringi/versions/1.1.6.

Conclusion

Throughout this post, I have covered ways that we can manipulate strings. However, the ultimate goal of this is that you may learn a way to clean up your data so that you can better analyze it. Often, messy data is in the form of strings, answers to a survey, voice recordings converted to strings, or things like locations. We need to be able to manipulate strings in order to organize these messy datasets. I hope that the message that you take away from this is that it is possible to clean up any dataset with the tools provided to us. Thank you.

References

1. https://help.libreoffice.org/Common/List_of_Regular_Expressions
2. https://www.rdocumentation.org/packages/stringi/versions/1.1.6
3. https://www.rstudio.com/wp-content/uploads/2016/09/RegExCheatsheet.pdf
4. https://www.tutorialspoint.com/r/r_strings.htm
5. R for Data Science by Hadley Wickham & Garrett Grolemund
6. https://cran.r-project.org/web/packages/stringi/stringi.pdf
7. https://www.rdocumentation.org/packages/stringi/versions/1.1.6
8. https://engineering.jhu.edu/ams/wp-content/uploads/sites/44/2014/06/canstockphoto6969679.jpg