

HW 04 - Strings and Regular Expressions

Stat 133, Spring 2018, Prof. Sanchez

Due date: Fri Apr-13 (before midnight)

The main underlying purpose of this assignment is to work with strings. More specifically, you will practice with some *basic/intermediate* manipulations of strings and regular expressions.

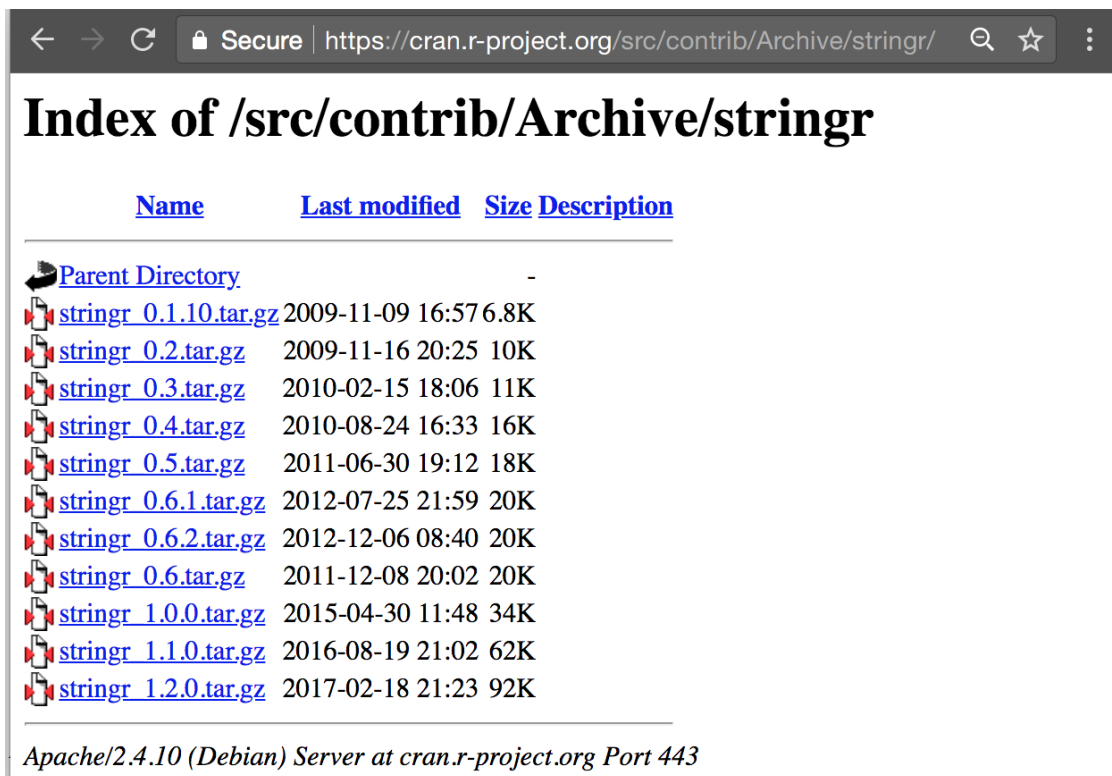
File Structure

After completing this assignment, the file structure of your project should look like this.

```
hw04/  
  README.md  
  data/  
    stringr-archive.csv  
    dplyr-archive.csv  
    ggplot2-archive.csv  
    XML-archive.csv  
    knitr-archive.csv  
  code/  
    archive-functions.R  
    regex-functions.R  
  images/  
    ... # png images  
  report/  
    hw04-first-last.Rmd  
    hw04-first-last.md
```

1) Archive of an R Package

Most R packages live in the **Comprehensive R Archive Network** better known as CRAN. For any package available in CRAN, you can check its associated archive.



Name	Last modified	Size	Description
Parent Directory	-		
stringr_0.1.10.tar.gz	2009-11-09 16:57	6.8K	
stringr_0.2.tar.gz	2009-11-16 20:25	10K	
stringr_0.3.tar.gz	2010-02-15 18:06	11K	
stringr_0.4.tar.gz	2010-08-24 16:33	16K	
stringr_0.5.tar.gz	2011-06-30 19:12	18K	
stringr_0.6.1.tar.gz	2012-07-25 21:59	20K	
stringr_0.6.2.tar.gz	2012-12-06 08:40	20K	
stringr_0.6.tar.gz	2011-12-08 20:02	20K	
stringr_1.0.0.tar.gz	2015-04-30 11:48	34K	
stringr_1.1.0.tar.gz	2016-08-19 21:02	62K	
stringr_1.2.0.tar.gz	2017-02-18 21:23	92K	

Apache/2.4.10 (Debian) Server at cran.r-project.org Port 443

This archive contains the different versions of a package that have been submitted to CRAN. For example, the archive of "stringr" is in the following url:

<http://cran.r-project.org/src/contrib/Archive/stringr>

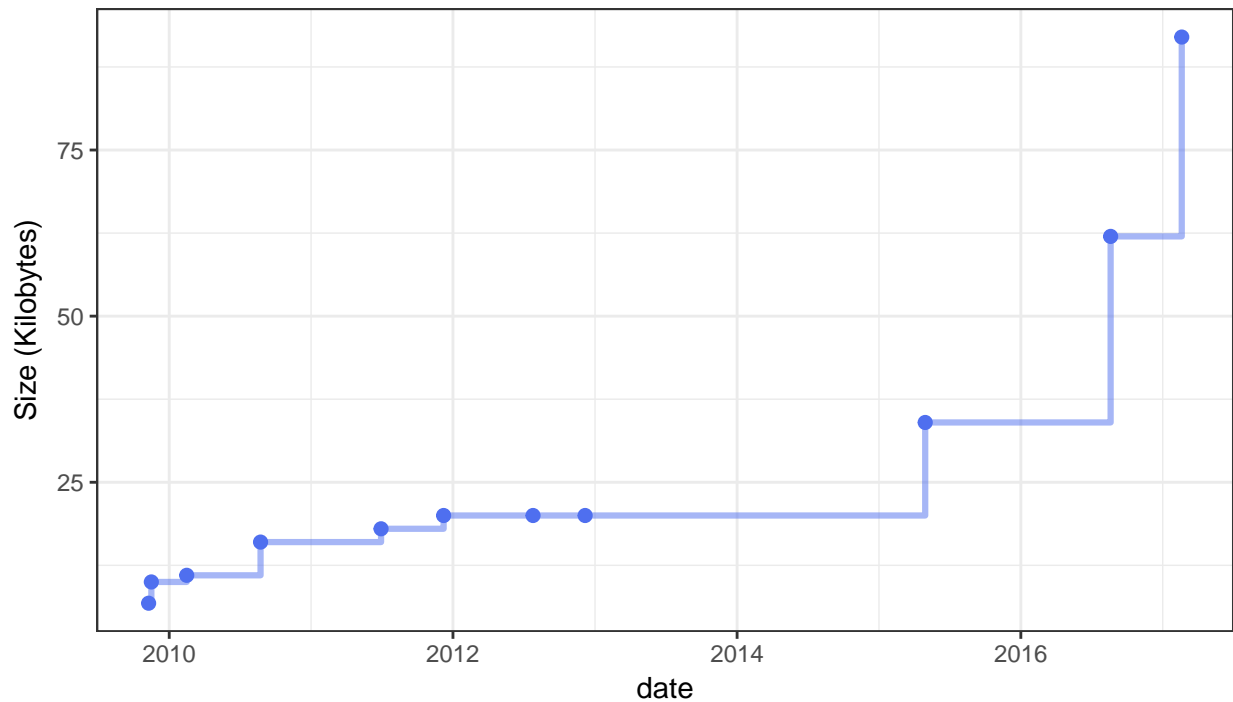
Your mission

Your first mission is to read in the table with the archive information, do some data preparation (e.g. cleaning, transformations, formatting) to produce a clean data frame (few rows shown below):

	name	version	date	size
1	stringr	0.1.10	2009-11-09	6.8
2	stringr	0.2	2009-11-16	10.0
3	stringr	0.3	2010-02-15	11.0
4	stringr	0.4	2010-08-24	16.0
5	stringr	0.5	2011-06-30	18.0

and then use the clean data to visualize the timeline of the version sizes for a given package:

stringr: timeline of version sizes



1.1) Read Archive Data Table

In the R script file `archive-functions.R`, write a function `read_archive()` that takes the name of a package, and returns the data frame from reading the HTML table with the archive data. Here's how you should be able to use `read_archive()`:

```
raw_data <- read_archive('stringr')
```

How do you read in HTML tables in R? One option to read in an html table in R is the function `readHTMLTable()` from the R package "XML" (by Duncan Temple Lang):

```
tbl_html <- readHTMLTable('http://cran.r-project.org/src/contrib/Archive/stringr')
tbl_html
```

```
## $`NULL`
##           Name      Last modified Size Description
## 1          <NA>          <NA> <NA>          <NA>
## 2    Parent Directory                -
## 3 stringr_0.1.10.tar.gz 2009-11-09 16:57 6.8K
## 4   stringr_0.2.tar.gz 2009-11-16 20:25 10K
## 5   stringr_0.3.tar.gz 2010-02-15 18:06 11K
## 6   stringr_0.4.tar.gz 2010-08-24 16:33 16K
## 7   stringr_0.5.tar.gz 2011-06-30 19:12 18K
## 8 stringr_0.6.1.tar.gz 2012-07-25 21:59 20K
## 9 stringr_0.6.2.tar.gz 2012-12-06 08:40 20K
```

```
## 10      stringr_0.6.tar.gz 2011-12-08 20:02 20K
## 11      stringr_1.0.0.tar.gz 2015-04-30 11:48 34K
## 12      stringr_1.1.0.tar.gz 2016-08-19 21:02 62K
## 13      stringr_1.2.0.tar.gz 2017-02-18 21:23 92K
## 14                                     <NA>      <NA> <NA>      <NA>
```

The output of `readHTMLTable()` is a list with as many elements as tables are in the HTML document associated to the provided url. In the case of a CRAN archive, there is only one HTML table.

1.2) Data Cleaning

In the R script file `archive-functions.R`, write a function `clean_archive()` that takes the output of `read_archive()` and returns a “tidy” table (e.g. `data.frame` or `tibble`) with the following four columns:

1. **name:** name of the package (as `character`)
2. **version:** version number (as `character`)
3. **date:** date (yyyy-mm-dd) in `Date` data type (see `?Date`)
4. **size:** size of `tar.gz` file in kilobytes (as `numeric`). *Caution: some packages have size values expressed in MB, these must be converted to KB.*

You should be able to call `clean_archive()` like this:

```
raw_data <- read_archive('stringr')
clean_data <- clean_archive(raw_data)
clean_data
```

```
##      name version      date size
## 1 stringr  0.1.10 2009-11-09  6.8
## 2 stringr    0.2 2009-11-16 10.0
## 3 stringr    0.3 2010-02-15 11.0
## 4 stringr    0.4 2010-08-24 16.0
## 5 stringr    0.5 2011-06-30 18.0
## 6 stringr  0.6.1 2012-07-25 20.0
## 7 stringr  0.6.2 2012-12-06 20.0
## 8 stringr    0.6 2011-12-08 20.0
## 9 stringr  1.0.0 2015-04-30 34.0
## 10 stringr  1.1.0 2016-08-19 62.0
## 11 stringr  1.2.0 2017-02-18 92.0
```

Export the table as a CSV file `"stringr-archive.csv"` to the `data/` folder.

Auxiliary Functions: Although not mandatory, we recommend writing `clean_archive()` as a *high-level* function, that is, a function that calls auxiliary or *low-level* functions.

For example, you could write smaller/simpler functions such as

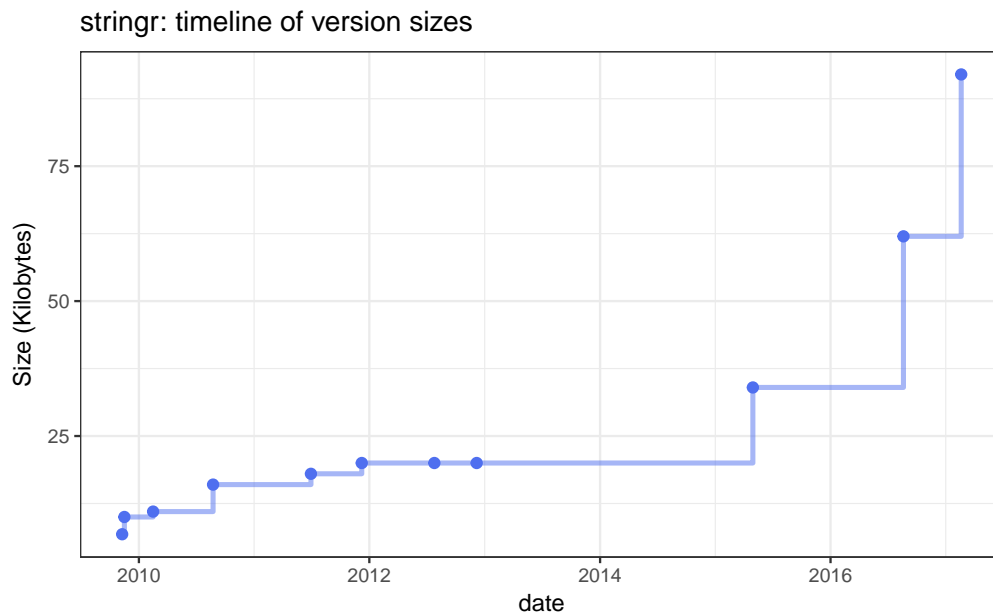
- `version_names()`: extracts the name of the package
- `version_numbers()`: extracts the number of the version
- `version_dates()`: extracts the date of the version
- `version_sizes()`: extracts the size of the version

As a rule of thumb, write short functions with less than 10 lines of code. If you have a function with more than 15 lines, you are doing too much.

1.3) Timeline plot

In the R script file `archive-functions.R`, write a function `plot_archive()` to visualize the timeline with the version sizes of a package. This function takes a clean archive table, and produces a step line chart—see `geom_step()` in `"ggplot2"`. The visual appearance (e.g. color, background, theme, etc) of the plot is up to you. Here's how you should be able to call `plot_archive()`:

```
# step line chart
plot_archive(clean_data)
```



1.4) Archive of "stringr"

So far, here's what to include in your Rmd file. You need to `source()` the functions in `archive-functions.R`, and run the commands:

- `raw_data <- read_archive('stringr')`
- `clean_data <- clean_archive(raw_data)`

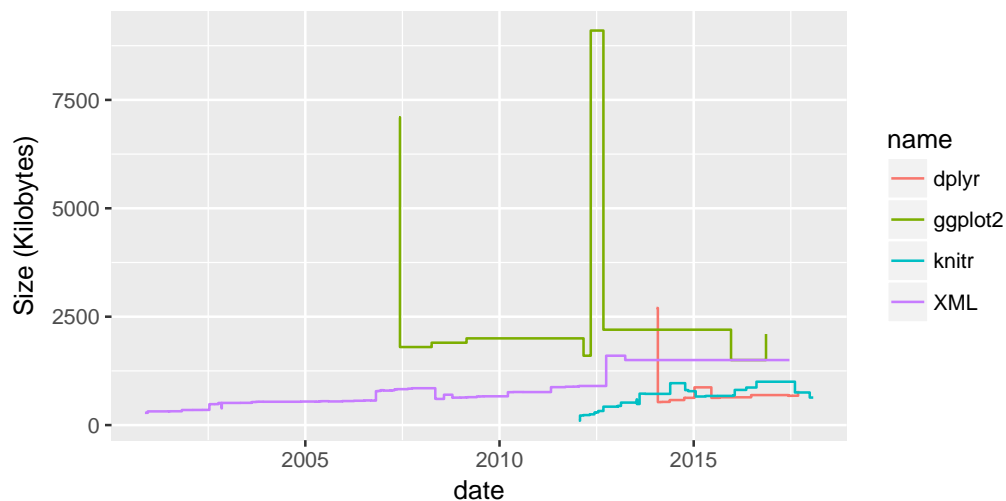
- `plot_archive(clean_data)`

1.5) Archives of "splyr", "ggplot2", "XML", and "knitr"

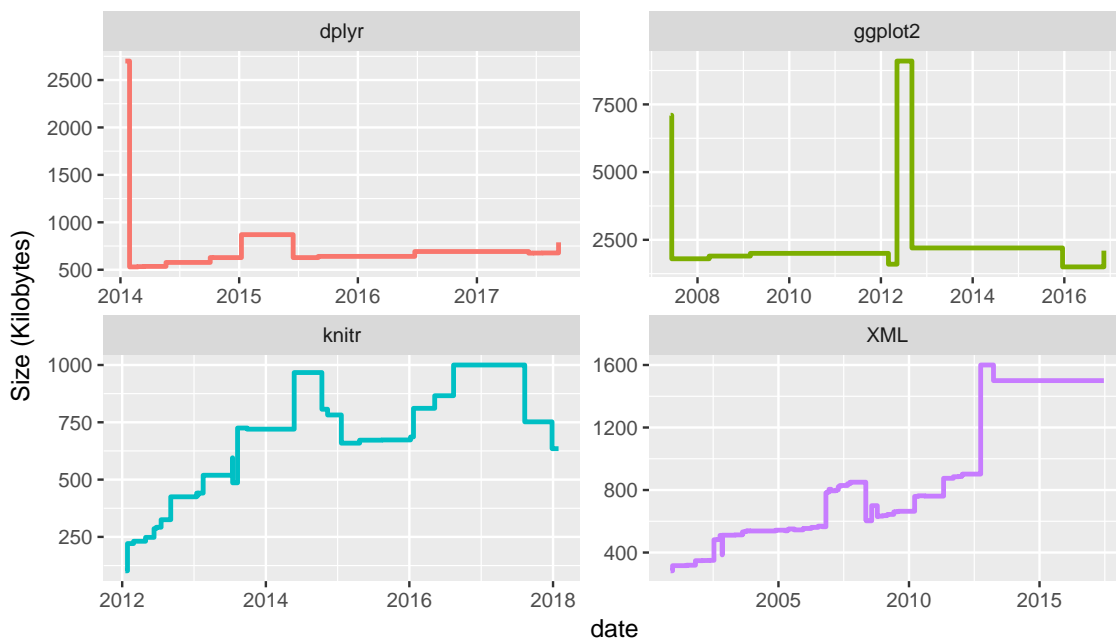
Besides testing your code with "stringr", you will have to get the CRAN archive tables for the packages "ggplot2", "XML", "knitr", and dplyr. Similarly, export each table as a CSV file, e.g. "ggplot2-archive.csv", "xml-archive.csv", etc., to the `data/` folder.

In your Rmd file, combine (or merge) all the data tables in one single data frame and use "ggplot2" to create two step line charts:

Plot all packages in one frame



Plot one package per facet (with free scales)



2) Regex Functions

In the R script file `regex-functions.R`, you will have to write the following functions:

- `split_chars()`
- `num_vowels()`
- `count_vowels()`
- `reverse_chars()`
- `reverse_words()`

In your Rmd file, `source()` the code in `regex-functions.R`, and include commands to run the sample code (see below) for each function.

2.1) Splitting Characters

Create a function `split_chars()` that takes a character string, and splits the content into one single character elements.

Here's an example of `split_chars()`:

```
split_chars('Go Bears!')
```

```
## [1] "G" "o" " " "B" "e" "a" "r" "s" "!"
```

```
split_chars('Expecto Patronum')
```

```
## [1] "E" "x" "p" "e" "c" "t" "o" " " "P" "a" "t" "r" "o" "n" "u" "m"
```

Note that `split_chars()` returns the output in a single vector. Each element is a single character.

2.2) Number of Vowels

Create a function `num_vowels()` that returns the number of vowels in a character vector. In this case, the input is a vector in which each element is a single character.

Here's an example of `num_vowels()`:

```
vec <- c('G', 'o', ' ', 'B', 'e', 'a', 'r', 's', '!')
num_vowels(vec)
```

```
## a e i o u
## 1 1 0 1 0
```

Notice that the output is a numeric vector with five elements. Each element has the name of the corresponding vowel.

2.3) Counting Vowels

Use the functions `split_chars()` and `num_vowels()` to write a function `count_vowels()` that computes the number of vowels of a character string. Make sure that `count_vowels()` counts vowels in both lower and upper case letters.

Here's how `count_vowels()` should be invoked:

```
count_vowels("The quick brown fox jumps over the lazy dog")
```

```
## a e i o u  
## 1 3 1 4 2
```

```
count_vowels("THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG")
```

```
## a e i o u  
## 1 3 1 4 2
```

2.4) Reversing Characters

Write a function `reverse_chars()` that reverses a string by characters. The input is a character vector, and the output is a character vector with the reversed characters.

Here's an example of `reverse_chars()`:

```
reverse_chars("gattaca")
```

```
## [1] "acattag"
```

```
reverse_chars("Lumox Maxima")
```

```
## [1] "amixaM xomuL"
```

2.5) Reversing Sentences by Words

Write a function `reverse_words()` that reverses a string (i.e. a sentence) by words

Test it:

```
reverse_words("sentence! this reverse")
```

```
## [1] "reverse this sentence!"
```

If the string is just one word then there's basically no reversing:

```
reverse_words("string")
```

```
## [1] "string"
```


3) Data “Emotion in Text”

The last part of this assignment involves working with the data file `text-emotion.csv` available in the course github repository. The original source is the data set “Emotion in Text” from the website Crowd Flower Data for Everyone <https://www.crowdflower.com/data-for-everyone/>

The file contains four columns:

- `tweet_id`: tweet identifier
- `sentiment`: class or sentiment label
- `author`: username author of the tweet
- `content`: content of the tweet

You will be working with the column `content` only.

In your Rmd file write R code to do computations in order to answer the following questions:

3.1) Number of characters per tweet

- Count the number of characters in the tweet contents.
- Display a `summary()` of such counts, and plot a histogram of these counts. To plot the histogram, use a bin width of 5 units: 1-5, 6-10, 11-15, 16-20, etc. In other words: the first bin involves tweets between 1 and 5 characters (inclusive), the second bin involves tweets containing between 6 and 10 characters (inclusive), and so on.

It is possible that you find tweets containing more than 140 characters. This has to do with the so-called *predefined XML entities* such as

- `&`; which represents an ampersand `&`
- `"`; which represent duotes `"`
- `<`; which represents less-than symbol `<`
- `>`; which represents greater-than symbol `>`

3.2) Number of Mentions

According to twitter, a “username can only contain alphanumeric characters (letters A-Z, numbers 0-9) with the exception of underscores. . . and it cannot be longer than 15 characters”.

Consider these three hypothetical tweet contents:

- a) `"@stat133 computing with data"`
- b) `"hi @__GSI do we have lab today @10am?"`
- c) `"blah blah &%@#$^&*()etc."`

Tweet a) has one mention: `@stat133`. Tweet b) has one mention, `@__GSI`, although it contains two `@` characters. Tweet c) has no mentions although it contains one `@` character.

You can argue that @10am, in tweet b), could be a username since it contains alphanumeric characters. However, to keep things simple, we will consider that the entire string is @10am?, which has a question mark, and therefore it is not a username.

- Count the number of @ mentions (i.e. mention to another user) in the tweet contents.
- Display such frequencies, and make a barplot of these counts (i.e. number of tweets with 0 mentions, with 1 mention, with 2 mentions, etc).
- Also write code to display the content of the tweet with 10 mentions.

If you want to know more about twitter usernames, visit:

<https://help.twitter.com/en/managing-your-account/twitter-username-rules>

3.3) Hashtags

People use the hashtag symbol # before a relevant keyword or phrase in their tweet to categorize those Tweets and help them show more easily in Twitter search. Hashtags cannot contain spaces or punctuation symbols, and cannot start with or use only numbers.

- Count the number of hashtags in the tweet contents.
- Display such frequencies, and make a barplot of these counts (i.e. number of tweets with 0 hashtags, with 1 hashtag, with 2 hashtags, etc).
- What is the average length of the hashtags?
- What is the most common length (i.e. the mode) of the hashtags?