HW 04 - Solutions

Stat 133, Spring 2018, Prof. Sanchez

Self-grade due date: Fri Apr-20 (before midnight)

General Self-Grading Instructions

- Please read this document (answer key).
- You will have to enter your score and comments in the *Assignments Comments* section of the correspoing assignment on bCourses.
- Enter your own scores and comments for (every part) of every problem in the homework on a simple coarse scale:
  - **0** = Didn't attempt or very very wrong,
  - **2** = Got started and made some progress, but went off in the wrong direction or with no clear direction,
  - **5** = Right direction and got half-way there,
  - **8** = Mostly right but a minor thing missing or wrong,
  - **10** = 100% correct.
- Also, enter your total score (out of an overall score of 110 points).
- If there is a cascading error, use a single deduction (don't double or triple or multiple penalize).
- Note: You must justify every self-grade score with a comment.
- Your self-grades will be due on Fri Apr-20 at 11:59 PM sharp.
- If you don't enter a proper grade by this deadline, you are giving yourself a zero on that assignment.
- Merely doing the homework is not enough, you must do the homework; turn it in on time; read the solutions; do the self-grade; and turn it in on time. Unless all of these steps are done, you will get a zero for that assignment.

File Structure (10 pts)

Give 10 pts for the required filestructure (with valid and complete—non-empty—files)

```
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 (50 pts)

Your `R` script file `archive-functions.R` should contain at least these three functions:

- `read_archive()`
- `clean_archive()`
- `plot_archive()`

and you could also have additional auxiliary functions like:

- `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
- *etc*

Archive Functions (30 pts)

Give 10 pts for each function working as expected. This involves being able to `source()` in your functions in the `Rmd` file, and use them to read, clean and plot the archive table of `"stringr"`

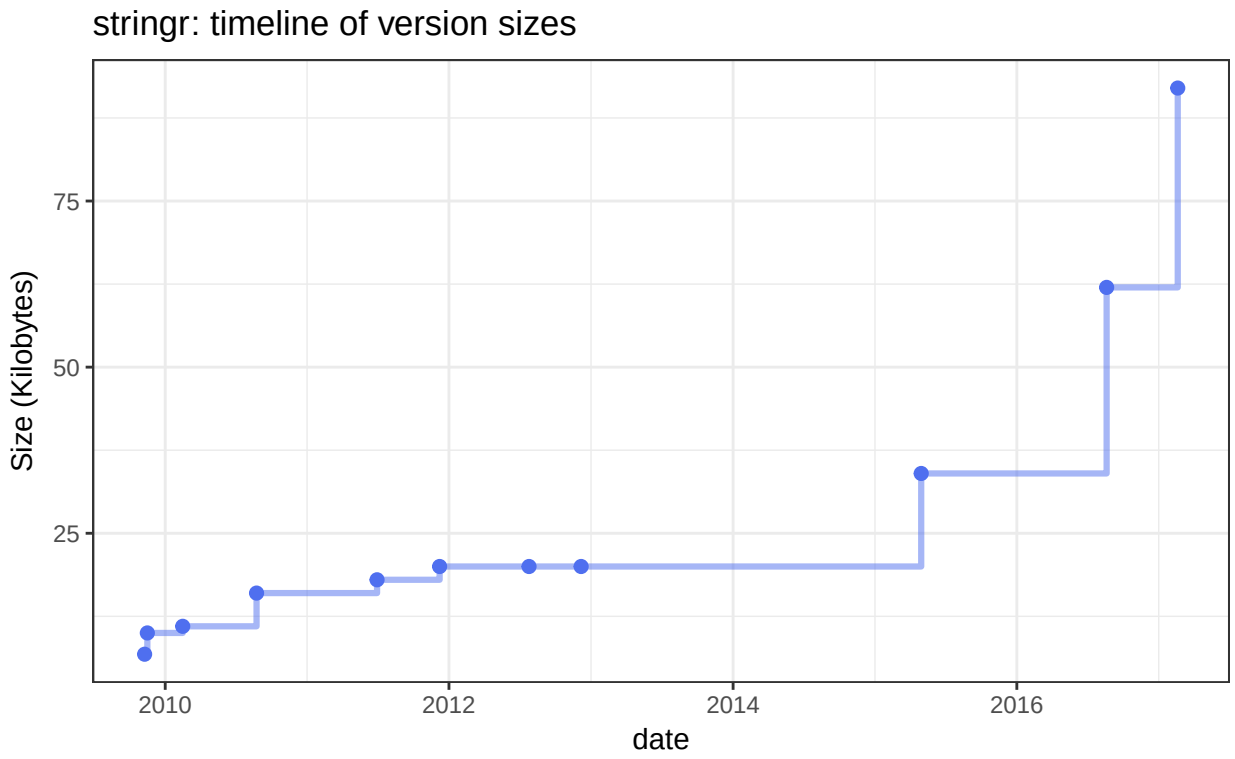
```
raw_data <- read_archive("stringr")
raw_data
```

```
##                      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>
```

```
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
```

```
plot_archive(clean_data)
```



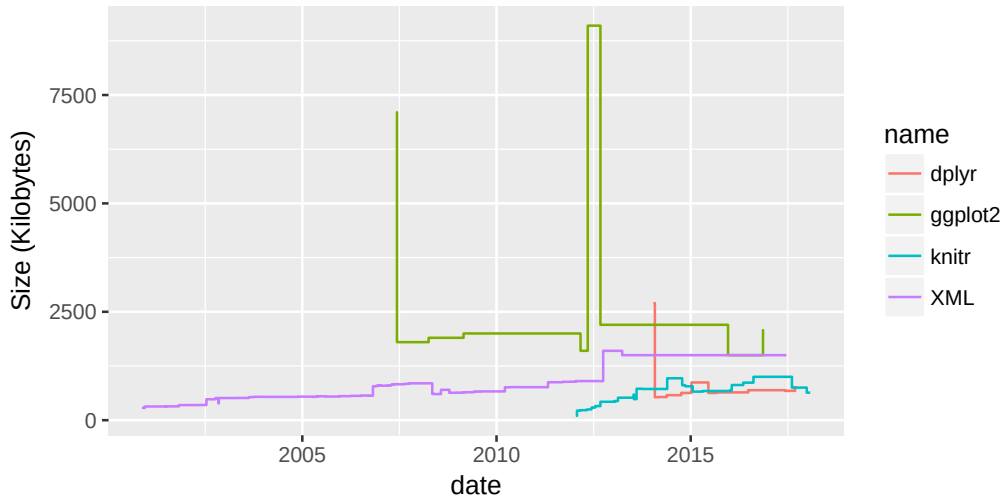
Other archives and plots (20 pts)

Besides testing your code with `"stringr"`, you had to:

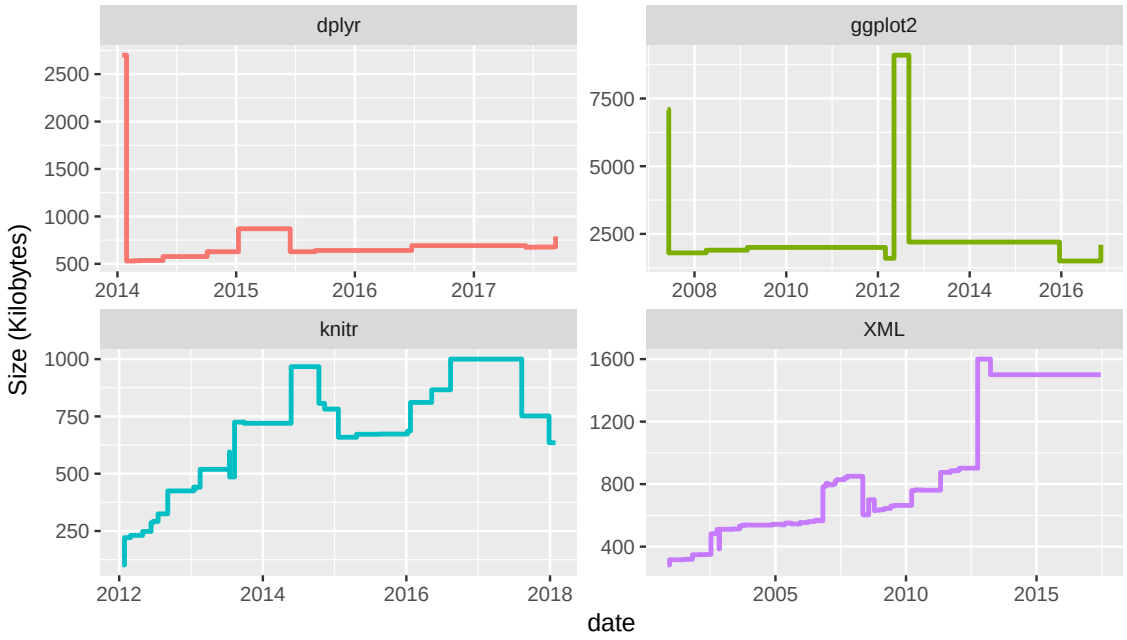
- get the CRAN archive tables for the packages `"ggplot2"`, `"XML"`, `"knitr"`, and `dplyr`.
- export each table as a CSV file, e.g. `"ggplot2-archive.csv"`, `"xml-archive.csv"`, etc., to the `data/` folder.
- in your `Rmd` file, combine (or merge) all the data tables in one single data frame
- and use `"ggplot2"` to create two step line charts

Give 10 pts for successfully getting the additional archive tables and export them to your repo; and another 10 pts for the 2 extra graphs in ggplot2.

Plot all packages in one frame



Plot one package per facet (with free scales)



---

2) Regex Functions (20 pts)

Your `R` script file `regex-functions.R` should contain the following functions:

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

Your `Rmd` file should include commands to run the sample code (see below) for each function.

You can give 6 pts for each function that gives the correct output. This would be the equivalent of 10 pts in the simple coarse scale. Deduct points as necessary.

The solutions contain one possible way to implement the requested functions, although your code will very likely be different. I expect that you compare your solution against mine, and see what parts of your code could be written more compactly and/or efficiently. Or even see if you have a "better" (e.g. clever, more elegant) solution than mine.

2.1) Splitting Characters

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

```
# split_chars()
split_chars <- function(x) {
  unlist(strsplit(x, ''))
}
```

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.

```
# num_vowels()
num_vowels <- function(chars) {
  vowels <- c('a', 'e', 'i', 'o', 'u')
  counts <- rep(0, 5)
  for (v in seq_along(vowels)) {
    counts[v] <- sum(chars == vowels[v])
  }
  names(counts) <- vowels
  counts
}
```

Here's an eaxmple 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.

```
# count_vowels()
count_vowels <- function(string) {
  chars <- split_chars(string)
  chars <- tolower(chars)
  num_vowels(chars)
}
```

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.

```
# reverse_chars()
reverse_chars <- function(string)
{
  string_split <- split_chars(string)
  reversed_split <- string_split[nchar(string):1]
```

```
  return(paste(reversed_split, collapse = ""))
}
```

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

```
# reverse_words()
reverse_words <- function(string)
{
  string_split = strsplit(as.character(string), split="\\s+")
  string_length <- length(string_split[[1]])
  if (string_length == 1) {
    reversed_string <- string_split[[1]]
  } else {
    reversed_split <- string_split[[1]][string_length:1]
    reversed_string <- paste(reversed_split, collapse = " ")
  }
  return(reversed_string)
}
```

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) Tweets (30 pts)

Assume that the data in `text-emotion.csv` has been imported in a data frame called `tweets`.

Your Rmd file should contain code to answer the following questions:

```
tweets <- read.csv('text-emotion.csv', stringsAsFactors = FALSE)
```

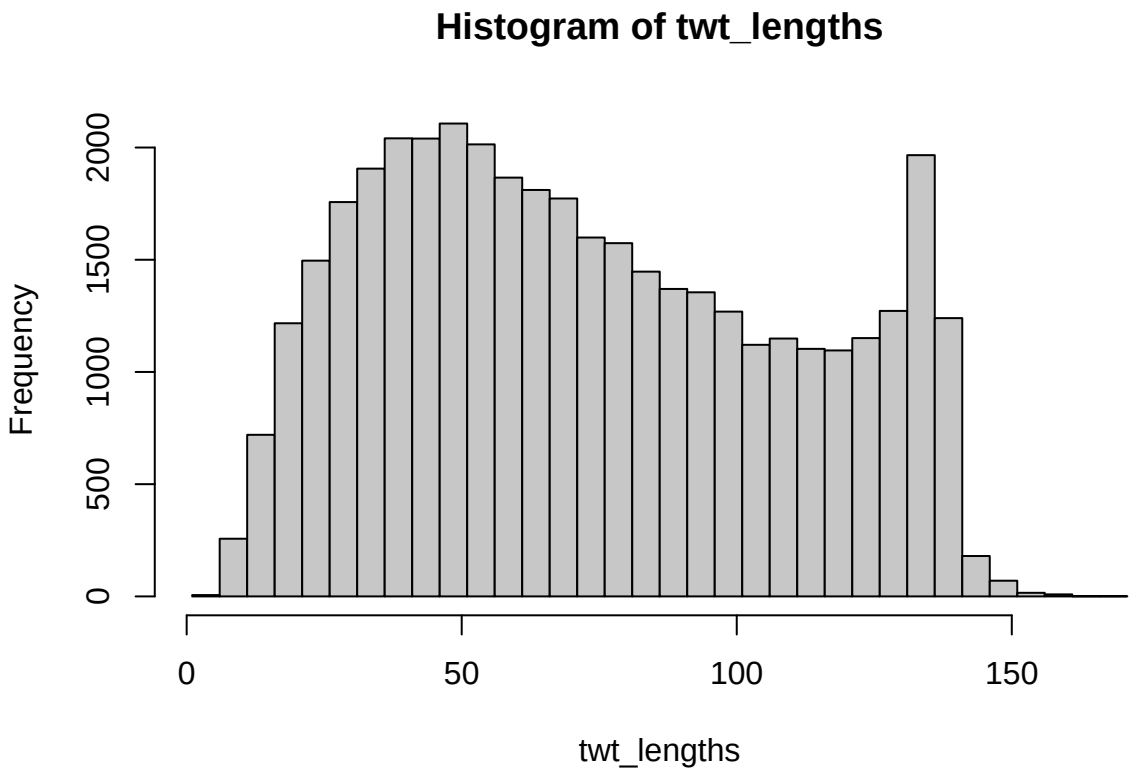
3.1) Number of characters per tweet (10 pts)

```
# Average number of characters of the tweets
twt_lengths <- str_count(tweets$content)
summary(twt_lengths)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00   43.00   69.00   73.41  103.00  167.00
```

You may have different values if you are using Windows

```
# histogram of the number of characters in the tweets
hist(twt_lengths, breaks = seq(1, 175, by = 5),
     include.lowest = TRUE, col = 'gray78')
```



3.2) Number of Mentions (10 pts)

Due to the ambiguities in the definition of a valid @ mention, you can give full credit for a regex pattern that at least matches something like "@\\w+". It is possible that may have

another patters, more or less elaborated, here's a couple of them. Depending on how *strict* your pattern is, you may end up having different answers:

- `"@\\w+"`
- `"@\\w{1,15}+"`
- `"@[a-zA-Z0-9_]{1,15}(\\s|$)"`
- `"(^|[^@\\w])@(\\w{1,15})\\b"`

- Count the number of `@` mentions (i.e. mention to another user) in the tweet contents.

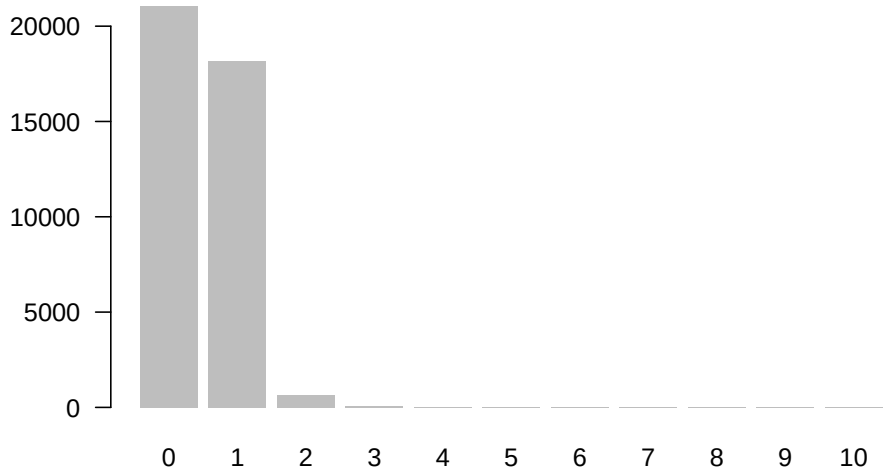
```
# Count @ mentions
twt_mentions <- str_extract_all(tweets$content, '@\\w+')
twt_mention_counts <- sapply(twt_mentions, length)

table(twt_mention_counts)
```

```
## twt_mention_counts
##     0     1     2     3     4     5     6     7     8     9    10
## 21043 18162   649    86    34    16     5     1     2     1     1
```

- 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).

```
barplot(table(twt_mention_counts), las = 1, border = NA, col = 'gray')
```



- Also write code to display the content of the tweet with 10 mentions.

```
tweets$content[twt_mention_counts == 10]
```

```
## [1] "last #ff  @Mel_Diesel @vja4041 @DemonFactory @shawnmcguirt @SEO_Web_Design @Chuc
```

3.3) Hashtags

The last part has to do with matching hashtags. In theory, hashtags cannot contain spaces or punctuation symbols, and cannot start with or use only numbers.

Here's a couple of examples that match #, followed a letter (lower or upper), and then alphanumeric characters. It is possible that you have a slightly different regex pattern that matches the same thing:

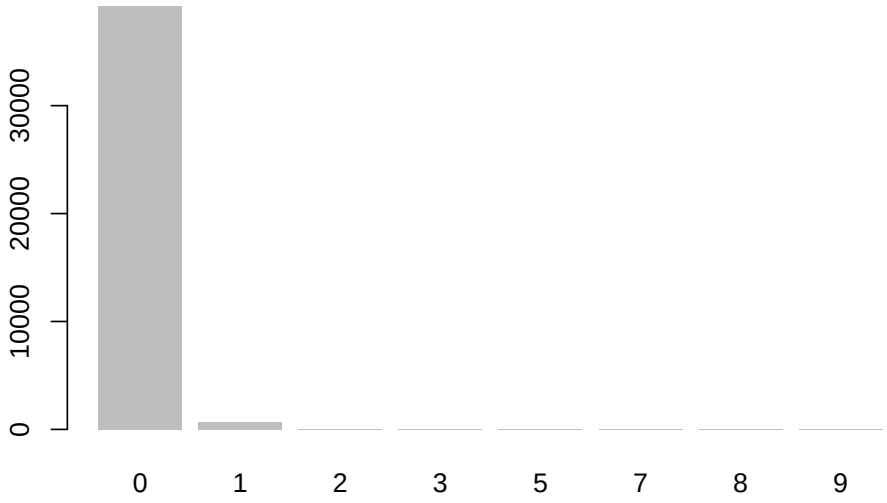
- `"(\\#[a-zA-Z]+[a-zA-Z0-9]*)"`
- `"#[[:alpha:]][[:alnum:]]*"`
- `"#([a-zA-Z][a-zA-Z0-9]*)"`

```
# Count the number of hashtags in the tweet contents.
twt_hashtags <- str_extract_all(tweets$content, '#[[:alpha:]][[:alnum:]]*')
twt_hashtags_counts <- sapply(twt_hashtags, length)
table(twt_hashtags_counts)
```

```
## twt_hashtags_counts
##     0     1     2     3     5     7     8     9
## 39261   650    66    17     1     1     1     3
```

- 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).

```
barplot(table(twt_hashtags_counts), border = NA)
```



- What is the average length of the hashtags?

If you ignore the hash symbol #, the average length of the hashtags is: (this answer contains duplicated hashtags). If you used unique occurrences, you will have a different (and also valid) answer.

```
hashtags <- unlist(twt_hashtags)
hash_length <- str_count(hashtags) - 1
mean(hash_length)
```

```
## [1] 7.625
```

- What is the most common length (i.e. the mode) of the hashtags?

```r
hash_length_tbl <- table(hash_length)
hash_length_tbl
```

```
## hash_length
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 34
##  1 71 74 97 74 59 75 73 97 51 54 63 18 22 27  7  6  3  2  2  2  1  1
```

```r
# mode(s)
names(hash_length_tbl)[hash_length_tbl == max(hash_length_tbl)]
```

```
## [1] "4" "9"
```