Lecture 14: String Analysis with Regular Expressions

UNIVERSITY OF SAN FRANCISCO

Abbie M. Popa

BSDS 100 - Intro to Data Science with `R`

Acknowledgements

h\t https://librarycarpentry.github.io/lc-data-intro/04-regular-expressions/index.html for much of the material

String Analysis

- Up until now, when we looked for strings we looked for a fixed match
- There are times when you may wish to look for a pattern match instead
- For example, imagine we have a data frame like this:

Name	Contact_Info
Abbie	apopa@usfca.edu
Jorge	555-555-5555
Purple	peater@usfca

- We want to find only the contact info that contains a correctly formatted e-mail address

Example, cont.

Name	Contact_Info
Abbie	apopa@usfca.edu
Jorge	555-555-5555
Purple	peater@usfca

- We could search for "@", but this would include the Purple People Eater, who has an incorrectly formatted e-mail address
- We could search for "@" and ".", but there might be other incorrect formatting this would still accept
- A better option is to make a pattern we can match

Regular Expressions

- A pattern for string matching is usually called a "regular expression", often abbreviated regex
- We build regular expressions using three tools:
  1. Literal characters, e.g., the letter `A` matches `A` (what we have been doing so far)
  2. Character classes, e.g., `[A-Z]` matches all capital letters from A to Z
  3. Special characters, which can do varied things (we will discuss these further)

Character Classes

- Square brackets can be used to specify a list or range of characters
- `[ABC]` matches A or B or C
- `[A-Z]` matches any upper case letter
- `[A-Za-z]` matches any upper or lower case letter
- `[A-Za-z0-9]` matches any upper case letter, lower case letter, or digit

Example with Character Classes

- Recall we can use `str_detect()` from the `stringr` library to return `TRUE` or `FALSE` if a pattern is detected
- We can also use `str_subset()` to return the actual values that match
- `R` also contains a useful object `letters` that contains all the lowercase letters
- Lets use regular expressions to find all the vowels

Example with Character Classes

- That was all well and good, but not very exciting
- The `stringr` package comes with a dataset "fruit" let's see how many fruit names contain the letters Q and Z (which are very uncommon in English)

Built-In Character Classes

- `\\d` matches any digit (equivalent to `[0-9]`)
- `\\s` matches any whitespace (e.g., space or tab)
- `\\w` matches any "word character" in some languages (not `R`) this means, letters or digits, however in `R` the underscore (_) is also included (possibly others?)

Special Characters - Wild Cards

- Special characters are often "wild cards" that can match many things
- `.` matches any character
- `*` matches the preceding element **0** or more times
  - ab*c matches "ac", "abc", "abbbbc", etc.
- `+` matches the preceding element **1** or more times
  - ab+c matches "abc", "abbbbc", but not "ac"
- `?` matches the preceding element **0 or 1** time
- `{VALUE, VALUE}` match the preceding element the number of times specified by the range

Wild Card Examples

- Parse "k?li.e"
  - This matches "k" zero or one time, "li" must be there, "." matches any character, and "e" must be there
- What about "k+li.e"?
- "o+li.e"?
- "oliv*e"?
- "oliv{1, 3}e"?

Practice

- **Character Classes**: Fill in the blank to find fruits whose names contain (capital or lower case) q, z, j, or x

```
str_subset(fruit, "[QZqz____]")
```

- **Wildcards**: Fill in the blank to fruits whose names contain the pattern any character, a, any chracter

```
str_subset(fruit, "___")
```

Special Characters - Positionals

- Some special characters indicate the position you want the pattern to occur in
- `^` Asserts the position is the start of the line, so what you put after the carat will only match if it's at the beginning of the line
- `$` Asserts the position is at the end of the line, so what you put before the dollar sign will only match if its at the end of the line
- `\\b` asserts a word boundary
  - `foobar` will match `foobar`, `555foobar`, `foobar777`, `555foobar777`
  - `\\bfoobar` will match `foobar` and `foobar777`
  - `foobar\\b` will match `foobar` and `555foobar`
  - `\\bfoobar\\b` will match `foobar` (only)

Positional Examples

- Find the fruits that **begin** with a vowel
- Find the fruits that **end** with a vowel
- Find the fruits that contain "berry"
- Find the fruits where "berry" is it's own word in the fruit name

Other Special Characters

- `()` can be used to make a "group", (if it's easier you can think of this as a word)
  - e.g., I want to find fruits that begin with "black" so I use "(^black)"
- `|` means "or"
  - e.g., I want to find fruits that contain "black" or "blue" so I use "(black)|(blue)"
- \\ is the "escape" character, that means it makes a special character not do it's special thing
  - e.g., to search for a plus sign, I would search for "\\+"
  - **NB!** you only need two backslashes for regex in `R`, in other languages, you only need one backslash

Practice

Using the built-in dataset `words` find all the words that:

- Start with the letter y
- End with the letter m
- Are exactly three letters long (with regex, not with `str_length()`!
- Are 7 or more letters long

Regex and other functions

- For our examples, we have mostly been using `str_subset()`
- A more common use for regex is to find rows in a dataset that match some condition with `str_detect()`
  - We want to find sentences that begin with "The" so we can examine them more closely.
- We can replace patterns that match a regular expression using `str_replace()` and `str_replace_all()`
  - `str_replace()` is **lazy**, will only replace the first match
  - `str_replace_all()` is **greedy**, will replace all matches

An application...

- One really fun application of regular expressions is sentiment analysis
- With sentiment analysis, the data scientist assesses a sentence, product review, or book to see how positive or negative (or more specific emotions) the writer is on the topic
- For more information, see https://www.tidytextmining.com/sentiment.htmlsummary-1 for a tutorial in `R`

Lab

- Let's review our initial question, we want to identify any strings that contain properly formatted e-mail addresses
- We'll make the following assumptions about e-mail addresses
  - They must begin with a letter
  - They must contain an `@`
  - Everything before the `@` must be a word character
  - The `@` must be followed by only letters (and at least one letter) until...
  - It ends with `.com`, `.org`, or `.edu`
- Write a regular expression to match strings that follow this pattern
- Test your regular expression using the test data Rdata file found at

https://github.com/abbiepopa/BSDS100/blob/master/Data/test_email.Rdata?raw=true