Text analysis workshop: stringr package and regex

Casey O'Hara

Overview

This tutorial will walk through an exercise in extracting specific information from untidily formatted blocks of text, i.e. sentences and paragraphs rather than a nice data frame or .csv.

The example comes from a paper I led that examined species ranges from different datasets, and found some discrepancies that resulted from systematic errors. Many of the coral species ranges for IUCN rangemaps extended off the continental shelf into very deep waters; but most corals require shallower water and are dependent upon photosynthesis. So I wanted to examine whether these corals, according to the IUCN's own information, could be found in waters deeper than 200 meters.

Load packages and data

The data are narratives pulled from the IUCN API (http://apiv3.iucnredlist.org/) for coral species, in order to identify their maximum depth. We'll also pull up a set of data on species areas, but mostly just because that data includes scientific names for the corals so we can refer to species rather than ID numbers.

```
library(tidyverse)
# library(stringr)

### original dataset from the manuscript is here:
# data_dir <- 'https://raw.githubusercontent.com/OHI-Science/IUCN-AquaMaps/master/clip_depth'

coral_narrs <- read_csv('data/iucn_narratives.csv')
# head(coral_narrs)
### interested in species_id, habitat
coral_info <- read_csv('data/coral_spp_info.csv')
# head(coral_info)
### info for species mapped in both datasets

### create a dataframe with just ID, scientific name, and habitat
coral_habs_raw <- coral_narrs %>%
  left_join(coral_info, by = 'iucn_sid') %>%
  select(iucn_sid, sciname, habitat)
```

examine a few habitat descriptions

```
coral_habs_raw$habitat[1:2]
```

```
## [1] "This species occurs in shallow, tropical reef environments. It is found only in subtidal turbid
## [2] "<em>Psammocora stellata</em> occurs on shallow wave washed rock, or at depths of 15-20 m depth
```

How can we extract depth information from these descriptions?

In pseudocode, we can think of a process as:

```
coral_habs <- coral_habs_raw %>%
  split into individual sentences %>%
  keep the sentences with numbers in them %>%
  isolate the numbers
```

Intro to stringr functions

Here we'll play a little with some basic stringr functions, and pattern vs. vector of strings. Consider especially how we can use `str_split`, `str_detect`, `str_replace`; later we'll see how to make effective use of `str_extract` as well.

- `str_match`, `str_match_all`
- `str_detect`
- `str_split`
- `str_replace`, `str_replace_all`
- `str_subset`, `str_count`, `str_locate`
- `str_trim`, `tolower`, `toupper`, `tools::toTitleCase`

```
x <- "Everybody's got something to hide except for me and my monkey"
tools::toTitleCase(x)
```

```
## [1] "Everybody's Got Something to Hide Except for Me and My Monkey"
```

```
tolower(x)
```

```
## [1] "everybody's got something to hide except for me and my monkey"
```

```
str_split(x, 'hide'); str_split(x, 't')
```

```
## [[1]]
## [1] "Everybody's got something to " " except for me and my monkey"
```

```
## [[1]]
## [1] "Everybody's go"        " some"                 "hing "
## [4] "o hide excep"          " for me and my monkey"
```

```
str_replace(x, 'except for', 'including')
```

```
## [1] "Everybody's got something to hide including me and my monkey"
```

```
str_replace(x, ' ', '_')
```

```
## [1] "Everybody's_got something to hide except for me and my monkey"
```

```
str_replace_all(x, ' ', '_')
```

```
## [1] "Everybody's_got_something_to_hide_except_for_me_and_my_monkey"
```

```
str_detect(x, 't'); str_detect(x, 'monk') ### is pattern in the string? T/F
```

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

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

```
str_match(x, 't'); str_match_all(x, 'y')  ### return every instance of the pattern in the string
```

```
##      [,1]
## [1,] "t"
```

```
## [[1]]
##      [,1]
```

```
## [1,] "y"
## [2,] "y"
## [3,] "y"
## [4,] "y"
```

```
  ### more useful when using wildcards as a pattern...

str_extract(x, 't'); str_extract_all(x, 'y')  ### return every instance of the pattern in the string
```

```
## [1] "t"
```

```
## [[1]]
## [1] "y" "y" "y" "y"
```

```
  ### more useful when using wildcards as a pattern...

str_locate(x, 't'); str_locate_all(x, 'y')
```

```
##      start end
## [1,]    15  15
```

```
## [[1]]
##      start end
## [1,]     5   5
## [2,]     9   9
## [3,]    54  54
## [4,]    61  61
```

Use `stringr` functions on coral data

First we can use `stringr::str_split()` to break down the habitat column into manageable chunks, i.e. sentences. What is an easily accessible delimiter we can use to separate a paragraph into sentences?

Take 1:

```
coral_habs <- coral_habs_raw %>%
  mutate(hab_cut = str_split(habitat, '.'))


coral_habs$hab_cut[1]
```

```
## [[1]]
##   [1] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
##  [24] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
##  [47] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
##  [70] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
##  [93] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [116] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [139] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [162] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [185] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [208] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [231] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [254] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
```

```
## [277] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [300] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [323] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [346] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [369] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [392] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [415] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [438] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [461] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [484] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [507] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [530] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [553] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [576] "" "" ""
```

Well that didn't work! In a moment we'll see that a period is actually a special character we will later use as a wild card in a "regular expression" or "regex" pattern. Some other characters have special uses as well; so if we want them to be interpreted literally, we need to "escape" them.

Some languages use a single backslash to escape a character (or turn a letter into a special function, e.g. '\n' indicates a line break). In R stringr functions, usually you end up having to use a double backslash (e.g. to get this to render a backslash-n, I had to type an extra backslash that doesn't show up)

Also: why is just separating on a period probably a bad idea? what else could we use?

Take 2:

```
# coral_habs <- coral_habs_raw %>%
#   mutate(hab_cut = str_split(habitat, '\. '))
### Error: '\.' is an unrecognized escape in character string starting "'\."

coral_habs <- coral_habs_raw %>%
  mutate(hab_cut = str_split(habitat, '\\. '))
### creates a cell with a vector of broken up sentences!
```

Use `unnest()` to separate out vector into rows

The str_split function leaves the chopped string in a difficult format - a vector within a dataframe cell. `unnest()` will unpack that vector into individual rows for us.

```
coral_habs <- coral_habs_raw %>%
  mutate(hab_cut = str_split(habitat, '\\. ')) %>%
  unnest(hab_cut)
```

Note the number of observations skyrocketed! Each paragraph was a single observation (for one coral species); now each species description is separated out into rows containing sentences.

Identify numbers and keep the sentences with number in 'em

Without wildcards, we'd have to identify each specific number. This would be annoying. Instead we can use some basic "regular expressions" or "regex" as wild card expressions. We put these in square brackets to create a list of everything we would want to match, e.g. `[aeiou]` would match any instance of lower case vowels.

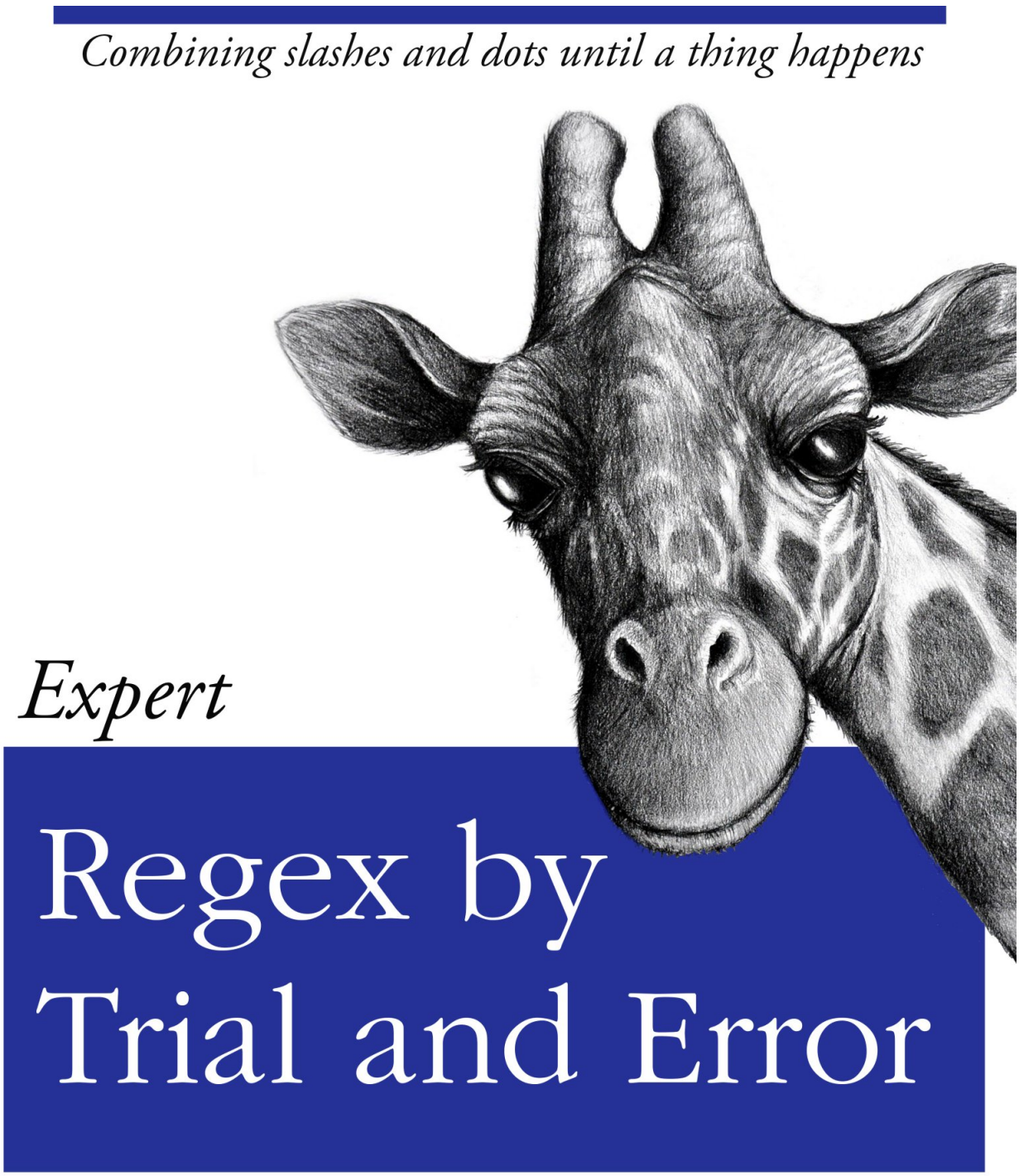
Combining slashes and dots until a thing happens
Expert
Regex by Trial and Error
O RLY?
@ThePracticalDev

Figure 1:

Helpful for testing regex: https://regex101.com/

```
### Without wildcards
coral_habs <- coral_habs_raw %>%
  mutate(hab_cut = str_split(habitat, '\\. ')) %>%
  unnest(hab_cut) %>%
  filter(str_detect(hab_cut, '1') | str_detect(hab_cut, '2'))

### With wildcards
coral_habs <- coral_habs_raw %>%
  mutate(hab_cut = str_split(habitat, '\\. ')) %>%
  unnest(hab_cut) %>%
  filter(str_detect(hab_cut, '[0-9]'))
    ### also works with [3-7], [a-z], [A-Z], [a-z0-9A-Z]
```

But not all numbers are depths

How can we differentiate further to get at depth info?

- exclude years? Knowing a bit about corals, can probably exclude any four-digit numbers; problems with that?
- match pattern of number followed by " m"

```
coral_depth <- coral_habs %>%
  filter(str_detect(hab_cut, '[0-9] m')) %>%
  mutate(depth = str_extract(hab_cut, '[0-9] m'))
```

Why didn't that work???? Only matched the single digit next to the "m"!

We need to use a quantifier:

- `+` means one or more times
- `*` means zero or more times
- `?` means zero or one time
- `{3}` means exactly three times
- `{2,4}` means two to four times; `{2,}` means two or more times

```
years <- coral_habs %>%
  mutate(year = str_extract(hab_cut, '[0-9]{4}'))
### looks for four numbers together

coral_depth <- coral_habs %>%
  filter(str_detect(hab_cut, '[0-9] m')) %>%
  mutate(depth = str_extract(hab_cut, '[0-9]+ m'))
### looks for one or more numbers, followed by ' m'
### Still misses the ranges e.g. "3-30 m" - how to capture?

### let it also capture "-" in the brackets
coral_depth <- coral_habs %>%
  filter(str_detect(hab_cut, '[0-9] m')) %>%
  mutate(depth = str_extract(hab_cut, '[0-9-]+ m'))
```

Also can use a "not" operator inside the brackets:

- `'[^a-z]'` matches "anything that is not a lower case letter"
- BUT: `'^[a-z]'` matches a start of a string, then a lower case letter.
- NOTE: `^` outside brackets means start of a string, inside brackets means "not"

```
### split 'em (using the "not" qualifier), convert to numeric, keep the largest
coral_depth <- coral_habs %>%
  filter(str_detect(hab_cut, '[0-9] m')) %>%
  mutate(depth_char = str_extract(hab_cut, '[0-9-]+ m'),
         depth_num = str_split(depth_char, '[^0-9]')) %>%
  unnest(depth_num)

coral_depth <- coral_depth %>%
  mutate(depth_num = as.numeric(depth_num)) %>%
  filter(!is.na(depth_num)) %>%
  group_by(iucn_sid, sciname) %>%
  mutate(depth_num = max(depth_num),
         n = n()) %>%
  distinct()
```

Note, still some issues in here: some fields show size e.g. 1 m diameter; other fields have slightly different formatting of depth descriptors; so it's important to make sure the filters (a) get everything you want and (b) exclude everything you don't want. We could keep going but we'll move on for now...

Other Examples

start string, end string, and "or" operator

Combining multiple tests using "or", and adding string start and end characters.

```
coral_threats <- coral_narrs %>%
  select(iucn_sid, threats) %>%
  mutate(threats = tolower(threats),
         threats_cut = str_split(threats, '\\. ')) %>%
  unnest(threats_cut) %>%
  filter(str_detect(threats_cut, '^a|s$'))
    ### NOTE: ^ outside brackets is start of a string, but inside brackets it's a negation
```

cleaning up column names in a data frame

Spaces and punctuation in column names can be a hassle, but often when reading in .csvs and Excel files, column names include extra stuff. Use regex and `str_replace` to get rid of these!

```
crappy_colname <- 'Per-capita income ($US) (2015 dollars)'
tolower(crappy_colname) %>%
  str_replace_all('[^a-z0-9]+', '_') %>%
  str_replace('^_|_$', '') ### in case any crap at the start or end
```

```
## [1] "per_capita_income_us_2015_dollars"
```

Lazy vs. greedy evaluation

When using quantifiers in regex patterns, we need to consider lazy vs. greedy evaluation of quantifiers. "Lazy" will find the shortest piece of a string that matches the pattern (gives up as early as it can); "greedy" will match the largest piece of a string that matches the pattern (takes as much as it can get). "Greedy" is the

default behavior, but if we include a question mark after the quantifier we force it to evaluate in the lazy manner.

```
x <- "Everybody's got something to hide except for me and my monkey"
x %>% str_replace('b.+e', '...')
```

```
## [1] "Every...y"
```

```
x %>% str_replace('b.+?e', '...')
```

```
## [1] "Every...thing to hide except for me and my monkey"
```

Lookaround (Lookahead and lookbehind) assertions

A little more advanced - Lookahead and lookbehind assertions are useful to match a pattern led by or followed by another pattern. The lookaround pattern is not included in the match, but helps to find the right neighborhood for the proper match.

```
y <- 'one fish two fish red fish blue fish'
y %>% str_locate('(?<=two) fish')   ### match " fish" immediately preceded by "two"
```

```
##      start end
## [1,]    13  17
```

```
y %>% str_locate('fish (?=blue)')   ### match "fish " immediately followed by "blue"
```

```
##      start end
## [1,]    23  27
```

```
y %>% str_replace_all('(?<=two|blue) fish', '...')
```

```
## [1] "one fish two... red fish blue..."
```

Using regex in `list.files()` to automate file finding

`list.files()` is a ridiculously handy function when working with tons of data sets. At its most basic, it simply lists all the non-hidden files in a given location. But if you have a folder with more folders with more folders with data you want to pull in, you can get fancy with it:

- use `recursive = TRUE` to find files in subdirectories
- use `full.names = TRUE` to catch the entire path to the file (otherwise just gets the basename of the file)
- use `all.files = TRUE` if you need to find hidden files (e.g. `.gitignore`)
- use `pattern = 'whatever'` to only select files whose basename matches the pattern - including regex!

```
list.files(path = 'sample_files')
```

```
## [1] "more"                     "notes1.md"
## [3] "sample_file_Apr2017.jpg" "sample_file_Apr2017.tif"
## [5] "sample_file_Apr2018.jpg" "sample_file_Apr2018.tif"
```

```
list.files(path = 'sample_files', pattern = 'jpg$', full.names = TRUE, recursive = TRUE)
```

```
## [1] "sample_files/sample_file_Apr2017.jpg"
## [2] "sample_files/sample_file_Apr2018.jpg"
```

```
list.files(path = '~/github/text_workshop/sample_files', pattern = '[0-9]{4}',
           full.names = TRUE, recursive = TRUE)
```

```
## [1] "/home/ohara/github/text_workshop/sample_files/more/sample_file_May2018.tif"
## [2] "/home/ohara/github/text_workshop/sample_files/sample_file_Apr2017.jpg"
## [3] "/home/ohara/github/text_workshop/sample_files/sample_file_Apr2017.tif"
## [4] "/home/ohara/github/text_workshop/sample_files/sample_file_Apr2018.jpg"
## [5] "/home/ohara/github/text_workshop/sample_files/sample_file_Apr2018.tif"
```

```
raster_files <- list.files('sample_files', pattern = '^sample.+[0-9]{4}.tif$')
  ### note: should technically be '\\.tif$' - do you see why?

### then create a raster stack from the files in raster_files, or loop
### over them, or whatever you need to do!
```