Lecture 4: Regex and PDF Scraping

Set Up

Here are the packages we will be using today:

```r
library(tidyverse)
library(pdftools) ## new!
library(tabulizer) ## new!
```

Regular Expressions

Regular expressions or REGEX are just ways to pattern-match on text. I'll start with a few examples to get us familiarized with REGEX, and then we'll show just how powerful they can be.

First off, regular expressions are used to pattern-match on *characters* or *strings* only! You cannot pattern match on *doubles* or *floats*. Hence, whenever we pattern match, we will want to make sure that the column we are matching on is a character type.

For demonstration purposes, we'll start out with a vector of words and phrases first, just so we're comfortable. We'll also use the `stringr::str_view_all` function to help us out and see our matches in real-time.

Matching on letters

```r
x <- c("Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.",
       "MichaelMichaelTopperTopperTopper")
```

We can match on specific phrases by simply typing out what we see:

```r
str_view_all(x, pattern = "Michael")
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

```r
str_view_all(x, pattern = "michael") ## case sensitive!
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

Notice that we are matching on *every* instance of Michael. Regular expressions are greedy by default (e.g, they will match on as much as they possibly can).

Matching on numbers

We can match on any number we want by using the `\d` regular expression. However, since `\` is a special escape character in R, we actually need to escape the escape character. Hence, to match on any digit, we will use `\\d`

- - matches on any digit (type in `\\d` for it to work)

```r
str_view_all(x, "\\d")
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

Now let's talk about quantifiers. Suppose we only wanted to match on exactly 3 numbers in a row:

- {n} - match on exactly n number of the previous expression
- {n,m} - match on n to m number of previous expression
- – $*$ match on 1 or more of the previous expression

```r
str_view_all(x, "\\d{3}")
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

```r
str_view_all(x, "\\d{4}") ## only matches on 4 numbers in a row
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

```r
str_view_all(x, "\\d+")
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

```r
str_view_all(x, "Michael+")
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

We can use:

- `[a-z]` to match on any letter, or `[A-Z]` to match on any uppercase letter:

```
str_view_all(x, "[a-z]")
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

```
str_view_all(x, "[A-Z]")
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

We can use:

- `\s` to match on any whitespace (written as `\\s`).

```
str_view_all(x, "\\s")
```

```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

Lastly, we can use:

- `.` to match on any character. Yes. I mean any character.

```
str_view_all(x, ".")
```

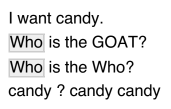
```
Hello! My name is Michael. My phone number is (805) 914-4285. Michael is my name.
MichaelMichaelTopperTopperTopper
```

Anchoring

We can get more flexibility by telling the regex to only match at the beginning of a string or the end of a string.

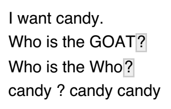
```
x2 <- c("I want candy.", "Who is the GOAT?", "Who is the Who?", "candy ? candy candy")
```

```
str_view_all(x2, "^Who")
```



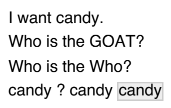
```
I want candy.
Who is the GOAT?
Who is the Who?
candy ? candy candy
```

```r
str_view_all(x2, "\\?$")
```



```
I want candy.
Who is the GOAT?
Who is the Who?
candy ? candy candy
```

```r
str_view_all(x2, "candy$")
```



```
I want candy.
Who is the GOAT?
Who is the Who?
candy ? candy candy
```

Special Characters that you must escape

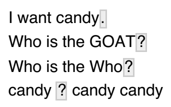
There are a few characters you need to escape:

- If you want to match on a `.` you must do `\\.`
- If you want to match on a `?` you must do `\\?`
- If you want to match on a `(` you must do `\\(`
- If you want to match on a `[` you msut do `\\[`

Using the OR operator

You can also use the OR operator for regular expressions:

```
str_view_all(x2, "\\.|\\?")
```



```
I want candy.
Who is the GOAT?
Who is the Who?
candy ? candy candy
```

Practice:

Extract the email addresses:

```
practice <- c("phone number is: (805) 9140-4285",
              "phone numberrrrr is (805) 402-4356",
              "their number ((805) 422-4555), is in the bag")
```

```
str_extract_all(practice, "\\(\\d\\d\\d\\)\\s\\d\\d\\d-\\d\\d\\d\\d")
```

```
## [[1]]
## character(0)
##
## [[2]]
## [1] "(805) 402-4356"
##
## [[3]]
## [1] "(805) 422-4555"
```

```
practice_1 <- c("email: michaeltopper@ucsb.edu",
                "email: miketopper@gmail.com",
                "email: michaeltopper@umail.ucsb.edu")
```

```
str_extract_all(practice_1, "\\s.{1,}.edu$|\\s.{1,}.com$")
```

```
## [[1]]
## [1] " michaeltopper@ucsb.edu"
##
## [[2]]
## [1] " miketopper@gmail.com"
##
## [[3]]
## [1] " michaeltopper@umail.ucsb.edu"
```

In practice:

In practice we're going to be using the `tidyr::extract` and `tidyr::separate` functions. These will perform regex extracting and splitting on a dataframe rather than just small vector.

```
horror_movies <- readr::read_csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/

horror_movies %>%
  extract(title, into = "year_released", "(\\d{4})") %>%
  relocate(year_released) %>%
  extract(movie_run_time, "movie_run_time", "(\\d{1,})")
```

Going through a PDF:

Now let's focus on scraping a PDF. First off, we're going to use the function: `pdftools::pdf_text`. This function just extracts all the text from a pdf.

```
crime_log <- pdf_text(here::here(paste0("lectures/lecture_4/crime_log.pdf")))
```

```
crime_log
```

One of the tricks to getting a pdf into the correct format is to break everything by lines. In a pdf, the newline character is `\n`. We are going to use the following functions for splitting/cleaning:

- str_split() - split into a vector by some regex.
- str_trim() - trim whitespace
- unlist() - will make a list into a vector
- which() - gives the index of a TRUE statement
- as_tibble() - will turn a vector into a dataframe (tibble)

Brief Refresher

Recall that there are ways to index things in R:

```
x2[1]
x2[3]
```

There are also lists:

```
list1 <- list(x, x2)
list1[1] # notice the difference
list1[[1]] # between both of these
```

Lists are a way to keep dimensionality down. It is important to know how to index lists. There is an important distinction between the double bracket `[[` which gives you the element inside the list, while `[` gives you the element *of* the list. `[` can extract multiple indices, while `[[` can only extract one exact one. ## Scraping

We'll always start off scraping by splitting the pdf:

```r
pdf_trimmed <- crime_log %>%
  str_split("\n") %>% ## splits the pdf by new-lines (\n)
  unlist() %>% ## unlists the list that was creating by splitting
  str_trim() %>%
  str_to_lower()
```

```r
incident_index <- pdf_trimmed %>%
  str_detect("^incident") %>%
  which()
```

```r
pdf_trimmed[incident_index] %>%
  as_tibble() %>%
  extract(value, into = "incident", regex = ".{1,}:(.{1,})report.{1,}") %>%
  mutate(incident = str_trim(incident))
```

Continue on…