Regular Expressi
Scraping,

JSC 370: Data Sc

February 12, 20

Today's goals

- Introduction to Regular Expressions
- Understand the fundamentals of Web Scrapp
- Learn how to use an API

Regular Expressions: What i

A regular expression (shortened as regex or regexp
expression) is a sequence of characters that define

Regular Expressions: Why sh

We can use Regular Expressions for:

- Validating data fields, email address, number
- Searching text in various formats, e.g., addres write an address.
- Replace text, e.g., different spellings, `Storm`
- Remove text, e.g., tags from an HTML text, `<` `George`.

Regular Expressions 101: Me

What makes **regex** special is metacharacters. While w
literals like `dog`, `human`, `1999`, we only make use of
metacharacters:

- `.` Any character except new line
- `^` beginning of the text
- `$` end of the text

Regular Expressions 101: Me

- `[regex]` Match a single character in `regex`
  - `[0123456789]` Any number
  - `[0-9]` Any number in the range 0-9
  - `[a-z]` Lower-case letters
  - `[A-Z]` Upper-case letters
  - `[a-zA-Z]` Lower or upper case letters.
  - `[a-zA-Z0-9]` Any alpha-numeric

Regular Expressions 101: Me

- `[^regex]` Match any except those in `regex`
  - `[^0123456789]` Match any except a nu
  - `[^0-9]` Match anything except in the ra
  - `[^./ ]` any except dot, slash, and spac

Regular Expressions 101: Me
1)

Ranges, e.g., `0-9` or `a-z`, are locale- and implementa
range of lower case letters may vary depending on the
problem, you could use Character classes. Some exam

- `[:lower:]` lower case letters in the current
- `[:upper:]` upper case letters in the current
- `[:alpha:]` upper and lower case letters in
zA-Z]

- `[:digit:]` Digits: 0 1 2 3 4 5 6 7 8 9
- `[:alnum:]` Alpha numeric characters `[:al`

Regular Expressions 101: Me
2)

For example, in the locale `en_US`, the word `Ḧóla` IS N
but IT IS fully matched by `[[:alpha:]]+`.

Other important Metacharacters:

- `\s` white space, equivalent to `[\r\n\t\f\v`
- `|` or (logical or).

Regular Expressions 101: Me
3)

These usually come together with specifying how mar

- `regex?` Zero or one match.
- `regex*` Zero or more matches
- `regex+` One or more matches
- `regex{n,}` At least `n` matches
- `regex{,m}` at most `m` matches

- `regex{n,m}` Between `n` and `m` matches.

Regular Expressions 101: Me
3)

There are other operators that can be very useful,

- `(regex)` Group capture.
- `(?:regex)` Group operation without capture
- `(?=regex)` Look ahead (match)
- `(?!regex)` Look ahead (don't match)
- `(?<=regex)` Look behind (match)

- `(?<!regex)` Look behind (don't match)

Regular Expressions 101: Exa

Here we are extracting the first occurrence of the follo
`stringr::str_extract()`):

regex	Hanna Perez [name]	The 年 year was 1999
.{5}	Hanna	The 年
n{2}	nn	
[0-9]+		1999
\s[a-zA-Z]+\s	Perez	year
\s[[:alpha:]]+\s	Perez	年
[a-zA-Z]+ [a-zA-Z]+	Hanna Perez	year was
([a-zA-Z]+\s?){2}	Hanna Perez	The
([a-zA-Z]+)\1	nn	

regex	Hanna Perez [name]	The 年 year was 1999
(@|#)[a-z0-9]+		

Regular Expressions 101: Exa

1..{5} Match **any character** (except line end) five times.

2.n{2} Match the letter **n** twice.

3.[0-9]+ Match **any number** at least once

4.\s[a-zA-Z]+\s Match a **space**, **any lower or upper ca**

5.\s[[:alpha:]]+\s Same as before but this time .

Regular Expressions 101: Exa

6.[a-zA-Z]+ [a-zA-Z]+ Match two sets of letters separ

7.([a-zA-Z]+\s?){2} Match any lower or upper case let
a white space, twice.

8.([a-zA-Z]+)\1 Match any lower or upper case letter a
same pattern again.

9.(@|#)[a-z0-9]+ Match either the @ or # symbol, fol
letter or number.

10.(?<=#|@)[a-z0-9]+ Match one or more lower case
the @ or # symbol.

11.\ [[a-z]+\ ] Match the symbol [, at least one lower c

Regular Expressions 101: Fur

1.Lookup text: `base::grepl()` , `stringr::str_`

2.Similar to `which()` , which elements are `TRUE ba`
`stringr::str_which()`

3.Replace the first instance: `base::sub()` , `string`

4.Replace all instances: `base::gsub()` , `stringr`

5.Extract text: `base::regmatches()` , `stringr:`
`stringr::str_extract_all()` .

Regular Expressions 101: Fur

For example, like in X (Twitter), let's create a regex that
with the following pattern:

```
(@|#)([[:alnum:]]+)
```

Code	@Hanna Perez [name] #html	The @年 y
str_detect(text, pattern) or grepl(pattern, text)	TRUE	TRUE
str_extract(text, pattern)	@Hanna	@年
str_extract_all(text, pattern)	[@Hanna, #html]	[@年]
str_replace(text, pattern, "\1justinbieber")	@justinbieber Perez [name] #html	The @justi
str_replace_all(text, pattern, "\1justinbieber")	@justinbieber Perez [name] #justinbieber	The @justi

Note: While it is not showing in the table, the group rep
instead of \1 in the code

Data

This week we will use a dataset consisting of medical t
www.mtsamples.com/. See the readme here. The data
columns: "X", "description", "medical_specialty", "sampl
"keywords".

```
fn <- "mtsamples.csv"
if (!file.exists(fn))
  download.file(
    url = "https://github.com/JSC370/JSC370-2024/blob/
    destfile = fn
  )
mtsamples <- fread(fn, sep = ",", header = TRUE)
names(mtsamples)
```

```
## [1] "V1"                "description"       "medical_specialty"
## [4] "sample_name"       "transcription"     "keywords"
```

Regex to Lookup Text: Tumo

Let's search through the "description" using `grepl` lo

```
# How many entries contain the word tumor
mtsamples[grepl("tumor", description, ignore.case = TR
# Generating a column tagging tumor
mtsamples[, tumor_related := grepl("tumor", descriptio
# Taking a look at a few examples
mtsamples[tumor_related == TRUE, .(description)][1:3,]
```

```
## [1] 67
##
##
## 1:                                Transurethral resection of a medium bladder tu
## 2:                                          Transurethral resection of th
## 3:  Cystoscopy, transurethral resection of medium bladder tumor (4.0 cm in diame
```

Notice the `ignore.case = TRUE`. This is equivale

case using `tolower()` before passing the text to the

Regex Lookup text: Pronoun

Now, let's try to guess the pronoun of the patient. To d
he, his, him, they, them, theirs, ze, hir, hirs, she, he
text):

```
mtsamples[, pronoun := str_extract(
  string  = tolower(transcription),
  pattern = "he|his|him|they|them|theirs|ze|hir|hirs|
)]
mtsamples[1:10,pronoun]
mtsamples[, table(pronoun, useNA = "always")]
```

```
##  [1] "his" "his" "his" "ze"  "he"  "he"  "he"  "he"  "he"  "ze"
## pronoun
##   he  him  hir  his  she them   ze <NA>
## 2558    6   14  934   46   13   43   68
```

What is the problem with this approach?

Regex Lookup text: Pronoun

For this we use the following regular expression:

```
(?<=\W|^)(he|his|him|they|them|theirs|
her)(?=\W|$)
```

Bit by bit this is:

- `(?<=regex)` lookback search.
  - `\W` any non alpha numeric character, this `[^[:alnum:]]`, `|` or
  - `^` the beginning of the text

Regex Lookup text: Pronoun

- `he|his|him...` any of these words,
- `(?=regex)` followed by,
  - `\W` any non alpha numeric character, this `[^[:alnum:]]`, `|` or
  - `$` the end of the text.

Regex Lookup text: Pronoun

```
mtsamples[, pronoun := str_extract(
  string  = tolower(transcription),
  pattern = "(?<=\\W|^)(he|his|him|they|them|theirs|z
  )]
mtsamples[1:10, pronoun]
```

```
##  [1] "she" "he"  "he"  NA    NA    "she" "she" NA    NA    NA
```

Regex Lookup text: Pronoun

```
mtsamples[, table(pronoun, useNA = "always")]
```

```
## pronoun
##    he   her   him   his   she them they <NA>
##   767   394    29   361   870    18    67 1176
```

Regex Extract Text: Type of C

- Imagine now that you need to see the types o
- For simplicity, let's assume that, if specified, it
i.e. single word.
- We are interested in the word before cancer, I

Regex Extract Text: Type of C

We can just try to **extract** the phrase `"[some word]` use the following regular expression

```
[[:alnum:]-_]{4,}\s*cancer
```

Where

- `[[:alnum:]-_]{4,}` captures any alphanu `_`. Furthermore, for this match to work there mu
- `\s*` captures 0 or more white-spaces, and
- `cancer` captures the word cancer:

Regex Extract Text: Type of C

```
mtsamples[, cancer_type := str_extract(tolower(keywor
mtsamples[, table(cancer_type)]
```

```
## cancer_type
##        anal cancer     bladder cancer      breast cancer       colon cancer
##                  1                  6                 16                 12
## endometrial cancer  esophageal cancer        lung cancer     ovarian cancer
##                  5                  1                  8                  1
##   papillary cancer    prostate cancer     uterine cancer
##                  2                 14                  4
```

Fundamentals of Web Scrap

What?

Web scraping, web harvesting, or web data extract
extracting data from websites -- Wikipedia

How?

- The `rvest` R package provides various tools
data.

- Under-the-hood, `rvest` is a wrapper of the x
the case of dynamic websites, take a look at se

Web scraping raw HTML: Ex

We would like to capture the table of COVID-19 death
Wikipedia.

```
library(rvest)
library(xml2)

# Reading the HTML table with the function xml2::read
covid <- read_html(
  x = "https://en.wikipedia.org/wiki/COVID-19_pandemi
  )

# Let's the the output
covid
```

```
## {html_document}
## <html class="client-nojs vector-feature-language-in-header-enabled vector-featur
```

```
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ..
## [2] <body class="skin-vector skin-vector-search-vue mediawiki ltr sitedir-ltr ..
```

Web scraping raw HTML: Ex

- We want to get the HTML table that shows up
use the function `xml2::xml_find_all()` ar

- The first will locate the place in the document
expression.

- XPath, XML Path Language, is a query langua
document.

- A nice tutorial can be found here

- Modern Web browsers make it easy to use XP

Live Example! (inspect elements in Google Chrome, M

Safari)

Web scraping with **xml2** and
package

Now that we know what is the path, let's use that and e

```
table <- xml2::xml_find_all(covid, xpath = "/html/body
table <- rvest::html_table(table) # This returns a li
head(table[[1]])
```

```
## # A tibble: 6 × 4
##   Country                `Deaths / million` Deaths    Cases
##   <chr>                  <chr>              <chr>     <chr>
## 1 World[a]               881                7,026,534 774,493,392
## 2 Peru                   6,507              221,583   4,536,733
## 3 Bulgaria               5,703              38,681    1,327,689
## 4 Bosnia and Herzegovina 5,066              16,382    403,565
## 5 Hungary                4,918              49,022    2,229,538
## 6 North Macedonia        4,761              9,968     350,499
```

Web APIs

What?

A Web API is an application programming interface for
browser. -- Wikipedia

Some examples include: twitter API, facebook API, Ge

How?

You can request data, the **GET method**, post data, the
things using the HTTP protocol.

How in R?

We will be using the `httr()` package, which is a wra
in turn provides access to the `curl` library that is use

Web APIs with curl

Structure of a URL (source: "HTTP: The Protocol E
Part 1")

Web APIs with curl

Under-the-hood, the `httr` (and thus `curl`) sends re

```
curl -X GET https://google.com -w "%{content_type}\n%{http_code}\n"
```

A get request (`-X GET`) to `https://google.com`
following: `content_type` and `http_code`:

```
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="https://www.google.com/">here</A>.
</BODY></HTML>
text/html; charset=UTF-8
301
```

We use the `httr` R package to make life easier.

Web API Example 1: Gene O

- We will make use of the Gene Ontology API.

- We want to know what genes (human or not) a
antiviral innate immune response (go term GO:
annotations that have evidence code ECO:000

Web API Example 1: Gene O

```
library(httr)
go_query <- GET(
  url   = "http://api.geneontology.org/",
  path  = "api/bioentity/function/GO:0140374/genes",
  query = list(
    evidence          = "ECO:0000006",
    relationship_type = "involved_in"
  ),
  # May need to pass this option to curl to allow to
  config = config(
    connecttimeout = 60
    )
)
```

We could have also passed the full URL directly...

Web API Example 1: Gene O

Let's take a look at the curl call:

```
curl -X GET "http://api.geneontology.org/api/bioentity/function/GO:0140374/genes?ev
```

What `httr::GET()` does:

```
> go_query$request
## <request>
## GET http://api.geneontology.org/api/bioentity/func
## Output: write_memory
## Options:
## * useragent: libcurl/7.58.0 r-curl/4.3 httr/1.4.1
## * connecttimeout: 60
## * httpget: TRUE
```

```
## Headers:
## * Accept: application/json, text/xml, application/
```

Web API Example 1: Gene O

Let's take a look at the response:

```
## Response [https://api.geneontology.org/api/bioentity/function/GO:0140374/genes?e
##   Date: 2024-02-12 14:34
##   Status: 200
##   Content-Type: application/json
##   Size: 107 kB
```

Remember the codes:

- 1xx: Information message
- 2xx: Success
- 3xx: Redirection

- 4xx: Client error
- 5xx: Server error

Web API Example 1: Gene O

We can extract the results using the `httr::conten`

```
dat <- content(go_query)
dat <- lapply(dat$associations, function(a) {
  data.frame(
    Gene        = a$subject$id,
    taxon_id    = a$subject$taxon$id,
    taxon_label = a$subject$taxon$label
  )
})
dat <- do.call(rbind, dat)
str(dat)
```

```
## 'data.frame':    100 obs. of  3 variables:
##  $ Gene       : chr  "UniProtKB:C3Y0M6" "UniProtKB:A0A287AMJ0" "UniProtKB:A0A287
##  $ taxon_id   : chr  "NCBITaxon:7739" "NCBITaxon:9823" "NCBITaxon:9823" "NCBITax
```

```
##  $ taxon_label: chr  "Branchiostoma floridae" "Sus scrofa" "Sus scrofa" "Ornitho
```

Web API Example 1: Gene O

The structure of the result will depend on the API. In th
so the content function returns a list in R. In other scen
(we will see more in the lab)

Genes experimentally annotated with the function **antiviral innate ir

Gene	taxon_id
UniProtKB:C3Y0M6	NCBITaxon:7739
UniProtKB:A0A287AMJ0	NCBITaxon:9823
UniProtKB:A0A287AKR1	NCBITaxon:9823
UniProtKB:A0A6I8NTG1	NCBITaxon:9258
UniProtKB:C3YWB1	NCBITaxon:7739
UniProtKB:C3YWB0	NCBITaxon:7739

Web API Example 2: Using T

- Sometimes, APIs are not completely open, yo
- The API may require to login (user+password
- In this example, I'm using a token which I obt
- You can find information about the National C Information API here

Web API Example 2: Using T

- The way to pass the token will depend on the
- Some require authentication, others need you query, i.e., directly in the URL.
- In this case, we pass it on the header.

```
stations_api <- GET(
  url    = "https://www.ncdc.noaa.gov",
```

Web API Example 2: Using T

This is equivalent to using the following query

```
curl --header "token: [YOUR TOKEN HERE]" \
  https://www.ncdc.noaa.gov/cdo-web/api/v2/stations?limit=1000
```

Note: This won't run, you need to get your own token

Web API Example 2: Using T

Again, we can recover the data using the `content()`

```
ans <- content(stations_api)
ans$results[[64]]
```

```
## $elevation
## [1] 136.6
##
## $mindate
## [1] "1938-01-01"
##
## $maxdate
## [1] "2013-12-01"
##
## $latitude
## [1] 33.8463
##
## $name
## [1] "CARBON HILL 4 SE, AL US"
##
```

```
## $datacoverage
## [1] 0.8596
##
## $id
## [1] "COOP:011377"
```

Web API Example 3: HHS health rec

Here we use the Department of Health and Human Se
specific health recommendations" (details at health.go

```
health_advice <- GET(
  url  = "https://health.gov/",
  path = "myhealthfinder/api/v3/myhealthfinder.json",
  query = list(
    lang = "en",
    age  = "32",
    sex  = "male",
    tobaccoUse = 0
  ),
  config = c(
    add_headers(accept = "application/json"),
    config(connecttimeout = 60)
```

```
  )
)
```

Web API Example 3: HHS health rec

Let's see the response

```
health_advice
```

```
## Response [https://health.gov/myhealthfinder/api/v3/myhealthfinder.json?lang=en&a
##   Date: 2024-02-12 15:39
##   Status: 200
##   Content-Type: application/json
##   Size: 359 kB
## {
##     "Result": {
##         "Error": "False",
##         "Total": 18,
##         "Query": {
##             "ApiVersion": "3",
##             "ApiType": "myhealthfinder",
##             "TopicId": null,
##             "ToolId": null,
##             "CategoryId": null,
## ...
```

Web API Example 3: HHS health rec

```
# Extracting the content
health_advice_ans <- content(health_advice)

# Getting the titles
txt <- with(health_advice_ans$Result$Resources, c(
  sapply(all$Resource, "[[", "Title"),
  sapply(some$Resource, "[[", "Title"),
  sapply(`You may also be interested in these health
))
cat(txt, sep = "; ")
```

Web API Example 3: HHS health rec

Quit Smoking; Protect Yourself from Seasonal Flu; Hep
Doctor; Talk with Your Doctor About Depression; Get V
(Adults Ages 19 to 49); Get Tested for HIV; Get Your Bl
Alcohol Only in Moderation; Talk with Your Doctor Abo
Weight; Testing for Syphilis: Questions for the Doctor; E
Hepatitis B; Testing for Latent Tuberculosis: Questions
Smoking: Conversation Starters; Manage Stress; Alcoh

Summary

- We learned about regular expressions with th

- We can use regular expressions to detect (`st`
(`str_replace()`), and extract (`str_extrac`

- We looked at web scraping using the **rvest** pa

- We extracted elements from the HTML/XML u
XPath expressions.

Summary

- We also used the `html_table()` function fr
HTML documents.

- We took a quick review on Web APIs and the
(HTTP).

- We used the **httr** R package (wrapper of **curl**)
APIs

- We even showed an example using a token p

- Once we got the responses, we used the `cor`
message of the response.

Detour on CURL options

Sometimes you will need to change the default set of c
the list of options in `curl::curl_options()`. A co
limit before dropping the conection, e.g.:

Using the **Health IT** API from the US government, we c
Adoption and Use by County (see docs here)

The problem is that it usually takes longer to get the da
`connecttimeout` (which corresponds to the flag –
call (see next slide)

Detour on CURL options

```
ans <- httr::GET(
  url    = "https://dashboard.healthit.gov/api/open-a
  query  = list(
    source = "AHA_2008-2015.csv",
    region = "California",
    period = 2015
    ),
  config = config(
    connecttimeout = 60
    )
)
```

Detour on CURL options

```
> ans$request
# <request>
# GET https://dashboard.healthit.gov/api/open-api.php
# Output: write_memory
# Options:
# * useragent: libcurl/7.58.0 r-curl/4.3 httr/1.4.1
# * connecttimeout: 60
# * httpget: TRUE
# Headers:
# * Accept: application/json, text/xml, application/x
```

Regular Expressions: Email v

This is the official regex for email validation implemente

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08
\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?
:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:(2(5[
0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0
-9]|[1-9]?[0-9])|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\
x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])
```

See the corresponding post in StackOverflow