Web Scraping With httr and rvest

Outline

Outline

httr for http requests in R

Outline

httr for http requests in R

httr for working with web API's (rtweet)

Outline

httr for http requests in R

httr for working with web API's (rtweet)

rvest for web scraping in R

Outline

httr for http requests in R

httr for working with web API's (rtweet)

rvest for web scraping in R

Useless/Useful Example: pupR

Outline

httr for http requests in R

httr for working with web API's (rtweet)

rvest for web scraping in R

Useless/Useful Example: pupR

Web scraping and cleaning a table from Wikipedia

httr for http

httr manages connections made with web servers via *http (hypertext transfer protocol)*, which drives the web!

httr for http

httr manages connections made with web servers via *http (hypertext transfer protocol)*, which drives the web!

http requests are sent from your computer to a web server that asks the server to perform some task. The requests take the form of verbs in all caps like GET

httr for http

httr manages connections made with web servers via *http (hypertext transfer protocol)*, which drives the web!

http requests are sent from your computer to a web server that asks the server to perform some task. The requests take the form of verbs in all caps like GET

httr provides functions to manage these requests

httr for http

Example: getting info from httpbin.org

httr for http

Example: getting info from httpbin.org

```
GET("http://httpbin.org/get", verbose())
```

httr for http

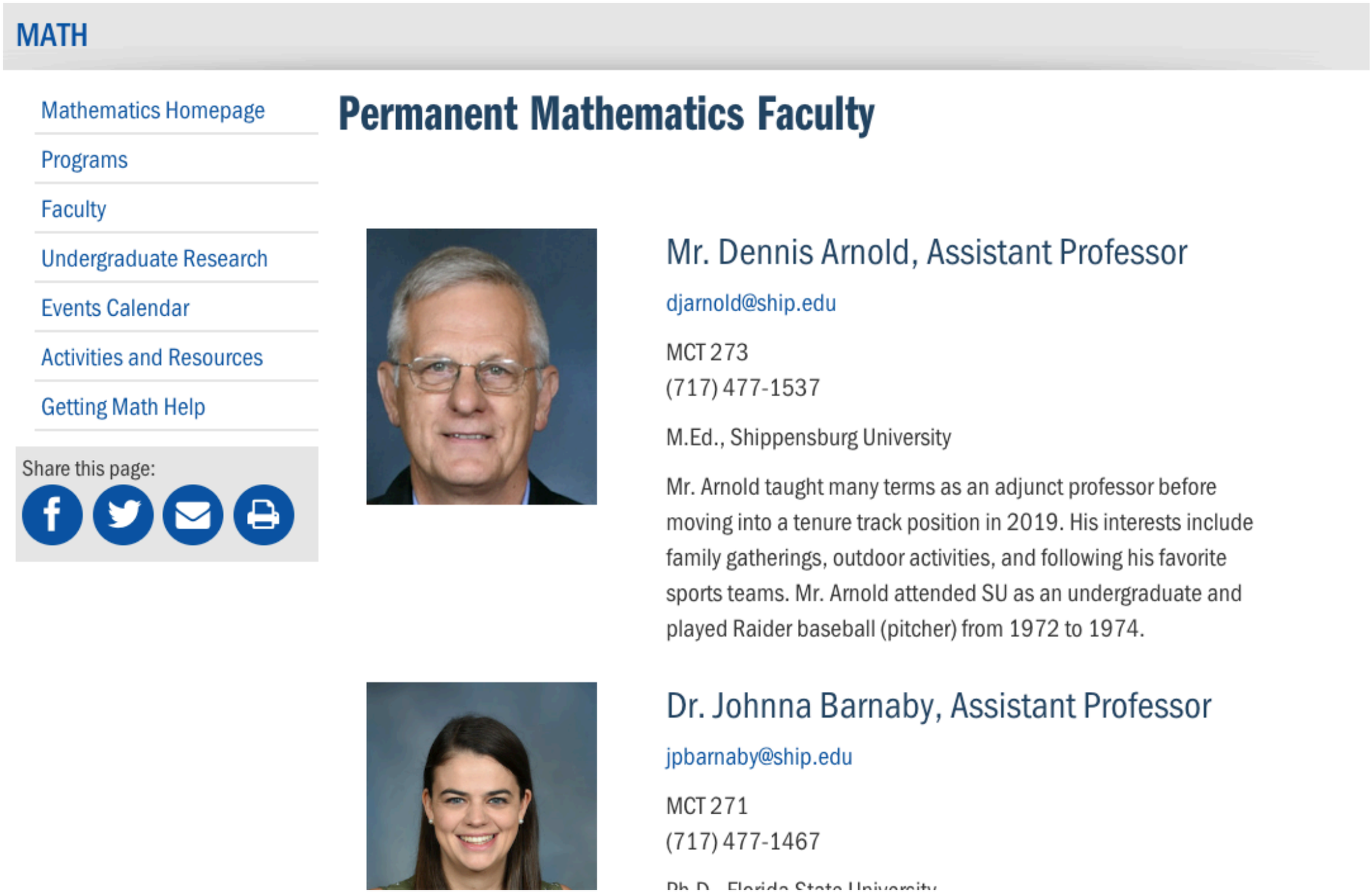
Example: getting info from httpbin.org

```
GET("http://httpbin.org/get", verbose())
```

```
-> GET /get HTTP/1.1
-> Host: httpbin.org
-> User-Agent: libcurl/7.64.1 r-curl/4.3 httr/1.4.1
-> Accept-Encoding: deflate, gzip
-> Accept: application/json, text/xml, application/xml, */*
->
<- HTTP/1.1 200 OK
<- Date: Mon, 02 Mar 2020 16:33:29 GMT
<- Content-Type: application/json
<- Content-Length: 366
<- Connection: keep-alive
<- Server: gunicorn/19.9.0
<- Access-Control-Allow-Origin: *
<- Access-Control-Allow-Credentials: true
<-
```

httr for http

Let's try to scrape emails from the math faculty page!



httr for http

First we need to pull down the content of the website!

httr for http

First we need to pull down the content of the website!

```
response <- GET(“http://www.ship.edu/math/faculty/", verbose())
```

httr for http

First we need to pull down the content of the website!

```
response <- GET("http://www.ship.edu/math/faculty/", verbose())
```

```
-> GET /math/faculty/ HTTP/1.1
-> Host: www.ship.edu
-> User-Agent: libcurl/7.64.1 r-curl/4.3 httr/1.4.1
-> Accept-Encoding: deflate, gzip
-> Cookie: ASP.NET_SessionId=p45neqoqubwlktqyqt2fri5v
-> Accept: application/json, text/xml, application/xml, */*
->
<- HTTP/1.1 200 OK
<- Cache-Control: private
<- Content-Type: text/html; charset=utf-8
<- Server: Microsoft-IIS/10.0
<- X-AspNetMvc-Version: 5.2
<- X-AspNet-Version: 4.0.30319
<- X-Powered-By: ASP.NET
```

httr for http

Looks good! What does it look like? We need content() for that!

httr for http

Looks good! What does it look like? We need content() for that!

```
response %>% content("raw")
```

httr for http

Looks good! What does it look like? We need content() for that!

```
response %>% content("raw")
```

```
[1] 0d 0a 0d 0a 3c 21 44 4f 43 54 59 50 45 20 68 74 6d
[18] 6c 3e 0d 0a 3c 68 74 6d 6c 20 6c 61 6e 67 3d 22 65
[35] 6e 22 3e 0d 0a 3c 68 65 61 64 3e 0d 0a 20 20 20 20
[52] 3c 6d 65 74 61 20 63 68 61 72 73 65 74 3d 22 75 74
[69] 66 2d 38 22 20 2f 3e 0d 0a 20 20 20 20 3c 6d 65 74
```

httr for http

Looks good! What does it look like? We need content() for that!

```
response %>% content("raw")

[1] 0d 0a 0d 0a 3c 21 44 4f 43 54 59 50 45 20 68 74 6d
[18] 6c 3e 0d 0a 3c 68 74 6d 6c 20 6c 61 6e 67 3d 22 65
[35] 6e 22 3e 0d 0a 3c 68 65 61 64 3e 0d 0a 20 20 20 20
[52] 3c 6d 65 74 61 20 63 68 61 72 73 65 74 3d 22 75 74
[69] 66 2d 38 22 20 2f 3e 0d 0a 20 20 20 20 3c 6d 65 74
```

http returns info in raw bytes. Not helpful to us! We can use content(“text”) to get readable text!

httr for http

Hoping for readable stuff!

httr for http

Hoping for readable stuff!

```
response %>% content("text")
```

httr for http

Hoping for readable stuff!

```
response %>% content("text")
```

```
[1] "\r\n\r\n<!DOCTYPE html>\r\n<html lang=\"en\">\r\n<head>\r\n
<meta charset=\"utf-8\" />\r\n    <meta http-equiv=\"X-UA-Compatible\"
content=\"IE=edge\" />\r\n    <meta name=\"viewport\"
content=\"width=device-width, initial-scale=1.0\" />\r\n
<title>Shippensburg University - Faculty</title>\r\n    <link
href=\"http://www.ship.edu/math/faculty/\" rel=\"canonical\">\r\n
<link rel=\"icon\" href=\"/favicon.ico\" />\r\n    \r\n\r\n    <script
src=\"//use.typekit.net/gkg8xxd.js\"></script>\r\n    <script>try
{ Typekit.load(); } catch (e) { }</script>\r\n\r\n    <link
type=\"text/css\" rel=\"stylesheet\" href=\"/Static/css/xyMain.css\"
data-skrollr-stylesheet />\r\n    <link type=\"text/css\"
rel=\"stylesheet\" href=\"/Static/css/xyLevel.css\"/>\r\n …
```

httr for http

Hoping for readable stuff!

```
response %>% content("text")
```

```
[1] "\r\n\r\n<!DOCTYPE html>\r\n<html lang=\"en\">\r\n<head>\r\n
<meta charset=\"utf-8\" />\r\n    <meta http-equiv=\"X-UA-Compatible\"
content=\"IE=edge\" />\r\n    <meta name=\"viewport\"
content=\"width=device-width, initial-scale=1.0\" />\r\n
<title>Shippensburg University - Faculty</title>\r\n    <link
href=\"http://www.ship.edu/math/faculty/\" rel=\"canonical\">\r\n
<link rel=\"icon\" href=\"/favicon.ico\" />\r\n    \r\n\r\n    <script
src=\"//use.typekit.net/gkg8xxd.js\"></script>\r\n    <script>try
{ Typekit.load(); } catch (e) { }</script>\r\n\r\n    <link
type=\"text/css\" rel=\"stylesheet\" href=\"/Static/css/xyMain.css\"
data-skrollr-stylesheet />\r\n    <link type=\"text/css\"
rel=\"stylesheet\" href=\"/Static/css/xyLevel.css\"/>\r\n …
```

A little better! We can now use *stringr* to extract emails!

httr for http

httr for http

```
response %>%
    content("text") %>%
    str_extract_all("\\w+@ship\\.edu") %>%
    flatten_chr() %>%
    unique()
```

httr for http

```
response %>%
    content("text") %>%
    str_extract_all("\\w+@ship\\.edu") %>%
    flatten_chr() %>%
    unique()
```

```
 [1] "djarnold@ship.edu"     "jpbarnaby@ship.edu"
 [3] "lebryant@ship.edu"     "jychoi@ship.edu"
 [5] "deensley@ship.edu"     "dlgochenaur@ship.edu"
 [7] "jehamb@ship.edu"       "GLInnerst@ship.edu"
 [9] "dikenn@ship.edu"       "kgmcgi@ship.edu"
[11] "dtmcni@ship.edu"        "lamelara@ship.edu"
[13] "kjpres@ship.edu"        "msrenault@ship.edu"
[15] "pttaylor@ship.edu"      "nethomas2@ship.edu"
[17] "math@ship.edu"
```

API’s

API (application programming interface) is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software.

API's

API (application programming interface) is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software.

You may need to interface with web API's to get data from servers, etc.

API's

API (application programming interface) is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software.

You may need to interface with web API's to get data from servers, etc.

Example: People love sports data, and you may be able to access the data directly from their API (Like the NFL's data API)

API's

API (application programming interface) is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software.

You may need to interface with web API's to get data from servers, etc.

Example: People love sports data, and you may be able to access the data directly from their API (Like the NFL's data API)

R packages: nflscrapr, nhlapi, lots of baseball ones, nbastatR, etc.

API's

API (application programming interface) is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software.

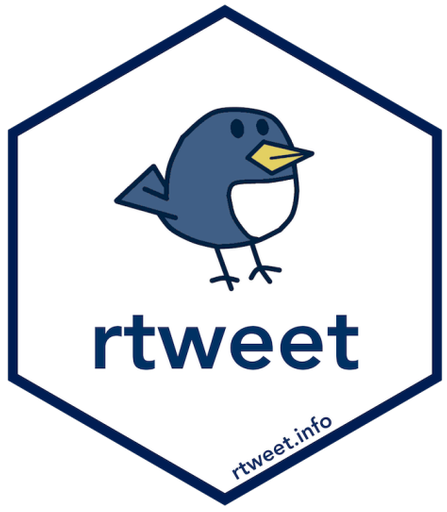
You may need to interface with web API's to get data from servers, etc.

Example: People love sports data, and you may be able to access the data directly from their API (Like the NFL's data API)

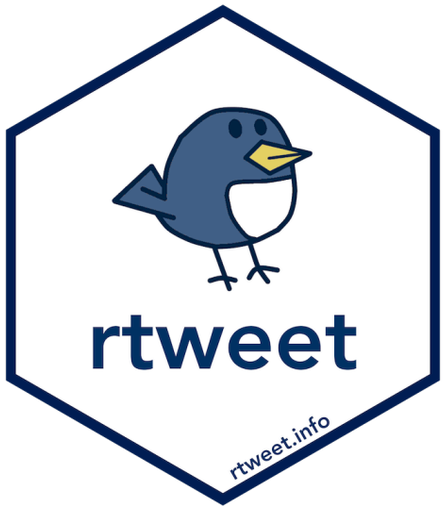
R packages: nflscrapr, nhlapi, lots of baseball ones, nbastatR, etc.

Lets look at something even more fun! Twitter!

rtweet

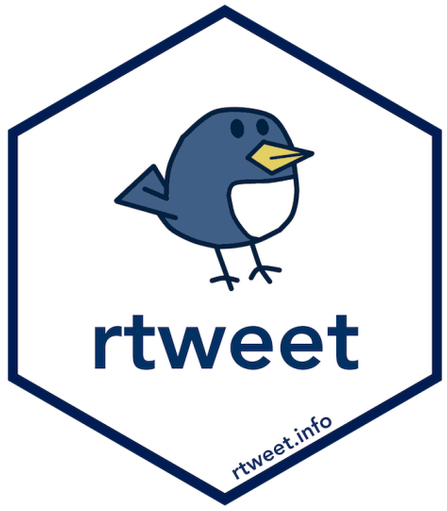


rtweet



rtweet is one of many R packages that help you interact with twitter's web API.

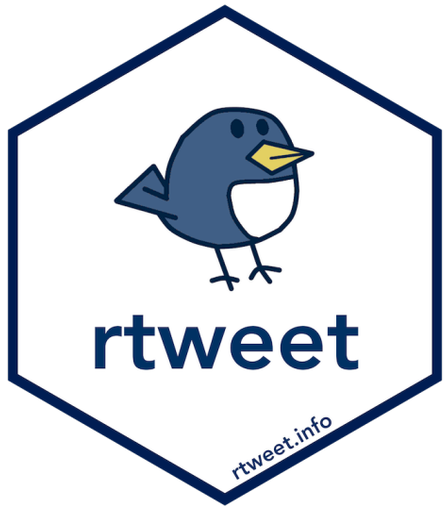
rtweet



rtweet is one of many R packages that help you interact with twitter's web API.

Often, you need to make a developer account in order to interact with the API, but that is not the case with the twitter API any more. You do need to have an account though!

rtweet



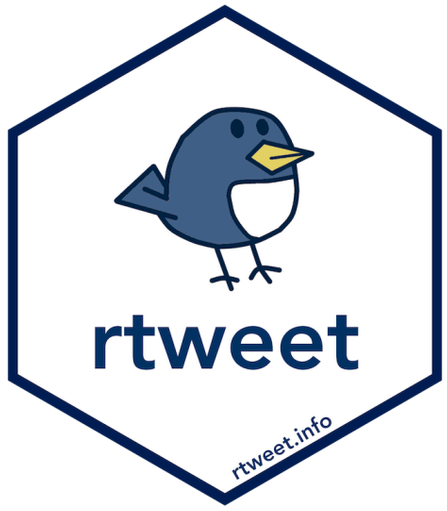
rtweet is one of many R packages that help you interact with twitter's web API.

Often, you need to make a developer account in order to interact with the API, but that is not the case with the twitter API any more. You do need to have an account though!

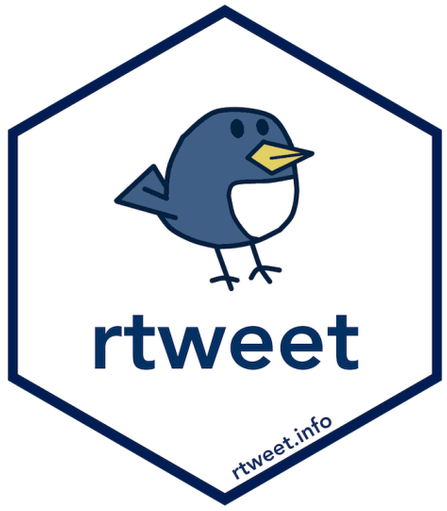
```
rt <- search_tweets(
  "#rstats", n = 18000, include_rts = FALSE
)
```

rtweet

```
rt <- search_tweets(
  "#rstats", n = 18000, include_rts = FALSE
)
```



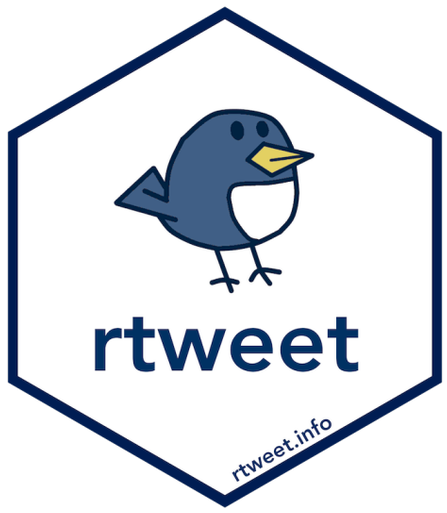
rtweet




```
rt <- search_tweets(
  "#rstats", n = 18000, include_rts = FALSE
)
```

```
rt$text[1]
```

rtweet

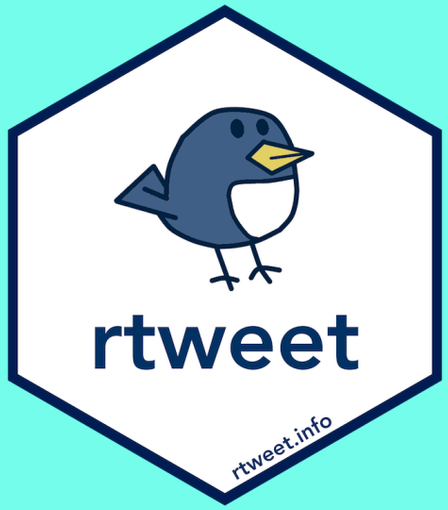



```
rt <- search_tweets(
  "#rstats", n = 18000, include_rts = FALSE
)
```

```
rt$text[1]
```

```
[1] "If anyone might need the below dataset for teaching #rstats
and #dataviz, I've created a (base R) gist to get you up to
speed quickly https://t.co/a82PWZkmHA https://t.co/wqUZq5PzZZ"
```

rtweet



Try to install and load in the rtweet package and take a few minutes to explore! https://rtweet.info/ is a helpful website!

rvest

rvest is a tidyverse-adjacent package for scraping html documents

rvest

rvest is a tidyverse-adjacent package for scraping html documents

html (hypertext markup language) is the standard markup language for creating web content

rvest

rvest is a tidyverse-adjacent package for scraping html documents

html (hypertext markup language) is the standard markup language for creating web content

rvest uses httr to obtain html from servers and knows how to parse and search them properly

rvest

rvest is a tidyverse-adjacent package for scraping html documents

html (hypertext markup language) is the standard markup language for creating web content

rvest uses httr to obtain html from servers and knows how to parse and search them properly

Most web browsers have options to look at the html that generated the webpage, called the page source.

html tags

html marks up webpages with tags, which say what type of thing the object should be

html tags

html marks up webpages with tags, which say what type of thing the object should be

```
<!DOCTYPE html>
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

html tags

html marks up webpages with tags, which say what type of thing the object should be

```
<!DOCTYPE html>
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

We can use rvest to pull out those specific tabs and lots of other stuff!

rvest

The rvest workflow:

rvest

The rvest workflow:

1. read_html() will grab the webpage in an xml format

rvest

The rvest workflow:

1. read_html() will grab the webpage in an xml format

2. html_nodes() will filter out desired content

rvest

The rvest workflow:

1. read_html() will grab the webpage in an xml format

2. html_nodes() will filter out desired content

3. html_text() or html_table() will parse content

rvest

Let’s scrape some emails again!

rvest

Let’s scrape some emails again!

```
(webpage <- read_html(“http://www.ship.edu/math/faculty/"))
```

rvest

Let's scrape some emails again!

```
(webpage <- read_html("http://www.ship.edu/math/faculty/"))
```

```
{html_document}
<html lang="en">
[1] <head>\n<meta http-equiv="Content-Type" content="tex ...
[2] <body class="level">\r\n\r\n    \r\n    \r\n    \r\n ...
```

rvest

Let's scrape some emails again!

```
(webpage <- read_html("http://www.ship.edu/math/faculty/"))
```

```
{html_document}
<html lang="en">
[1] <head>\n<meta http-equiv="Content-Type" content="tex ...
[2] <body class="level">\r\n\r\n    \r\n    \r\n    \r\n ...
```

What kind of nodes should I be looking for?

rvest

What kind of nodes should I be looking for?

rvest

What kind of nodes should I be looking for?

```
webpage %>% html_nodes("a")
```

rvest

What kind of nodes should I be looking for?

```
webpage %>% html_nodes("a")
```

```
{xml_nodeset (82)}
 [1] <a href="#content-area" title="Jump to Main Content ...
 [2] <a href="/" title="Shippensburg University">\r\n    ...
 [3] <a data-target="js-global-header-toggle-container"  ...
 [4] <a href="/academics" title="Academics">Academics</a>
 [5] <a href="/admissions" title="Admissions">Admissions ...
 [6] <a href="/about" title="About">About</a>
 [7] <a href="/" title="Shippensburg University">\r\n    ...
 [8] <a href="/student_life" title="Life At Ship">Life A ...
 [9] <a href="https://www.shipraiders.com/index.aspx" ti ...
[10] <a href="/give" title="Give">Give</a>
[11] <a href="https://portal.ship.edu/" title="Access my ...
[12] <a href="/math/">Math</a>
```

rvest

Can I do better?

rvest

Can I do better?

```
webpage %>% html_nodes("a.email")
```

rvest

Can I do better?

```
webpage %>% html_nodes("a.email")
```

```
{xml_nodeset (17)}
 [1] <a href="mailto:djarnold@ship.edu" class="email">dj ...
 [2] <a href="mailto:jpbarnaby@ship.edu" class="email">j ...
 [3] <a href="mailto:lebryant@ship.edu" class="email">le ...
 [4] <a href="mailto:jychoi@ship.edu" class="email">jych ...
 [5] <a href="mailto:deensley@ship.edu" class="email">de ...
 [6] <a href="mailto:dlgochenaur@ship.edu" class="email" ...
 [7] <a href="mailto:jehamb@ship.edu" class="email">jeha ...
 [8] <a href="mailto:GLInnerst@ship.edu" class="email">G ...
 [9] <a href="mailto:dikenn@ship.edu" class="email">dike ...
[10] <a href="mailto:kgmcgi@ship.edu" class="email">kgmc ...
[11] <a href="mailto:dtmcni@ship.edu" class="email">dtmc ...
[12] <a href="mailto:lamelara@ship.edu" class="email">la ...
```

rvest

Pull it all together!

rvest

Pull it all together!

```
webpage %>% html_nodes("a.email") %>% html_text()
```

rvest

Pull it all together!

```
webpage %>% html_nodes("a.email") %>% html_text()
```

```
 [1] "djarnold@ship.edu"     "jpbarnaby@ship.edu"
 [3] "lebryant@ship.edu"     "jychoi@ship.edu"
 [5] "deensley@ship.edu"     "dlgochenaur@ship.edu"
 [7] "jehamb@ship.edu"       "GLInnerst@ship.edu"
 [9] "dikenn@ship.edu"       "kgmcgi@ship.edu"
[11] "dtmcni@ship.edu"        "lamelara@ship.edu"
[13] "kjpres@ship.edu"        "msrenault@ship.edu"
[15] "pttaylor@ship.edu"      "nethomas2@ship.edu"
[17] "math@ship.edu"
```

rvest

We could also scrape tables, pictures, anything really from a webpage (if of course you are allowed to!!)

rvest

We could also scrape tables, pictures, anything really from a webpage (if of course you are allowed to!!)

An important (and sometimes legal!) issue is if you are allowed to scrape the contents of the website!

rvest

We could also scrape tables, pictures, anything really from a webpage (if of course you are allowed to!!)

An important (and sometimes legal!) issue is if you are allowed to scrape the contents of the website!

Many (all?) websites have a robots.txt file that tell crawlers what they are and aren't allowed to do!

rvest

We could also scrape tables, pictures, anything really from a webpage (if of course you are allowed to!!)

An important (and sometimes legal!) issue is if you are allowed to scrape the contents of the website!

Many (all?) websites have a robots.txt file that tell crawlers what they are and aren't allowed to do!

Example: pupR!

Data Cleaning Example

Let’s bring down a table from Wikipedia and try to clean it so it’s analysis ready!!