Web APIs

Max Turgeon

SCI 2000–Introduction to Data Science

Lecture Objectives

- Access APIs from R
- Learn basics of JSON
- Compare and contrast web scraping and APIs

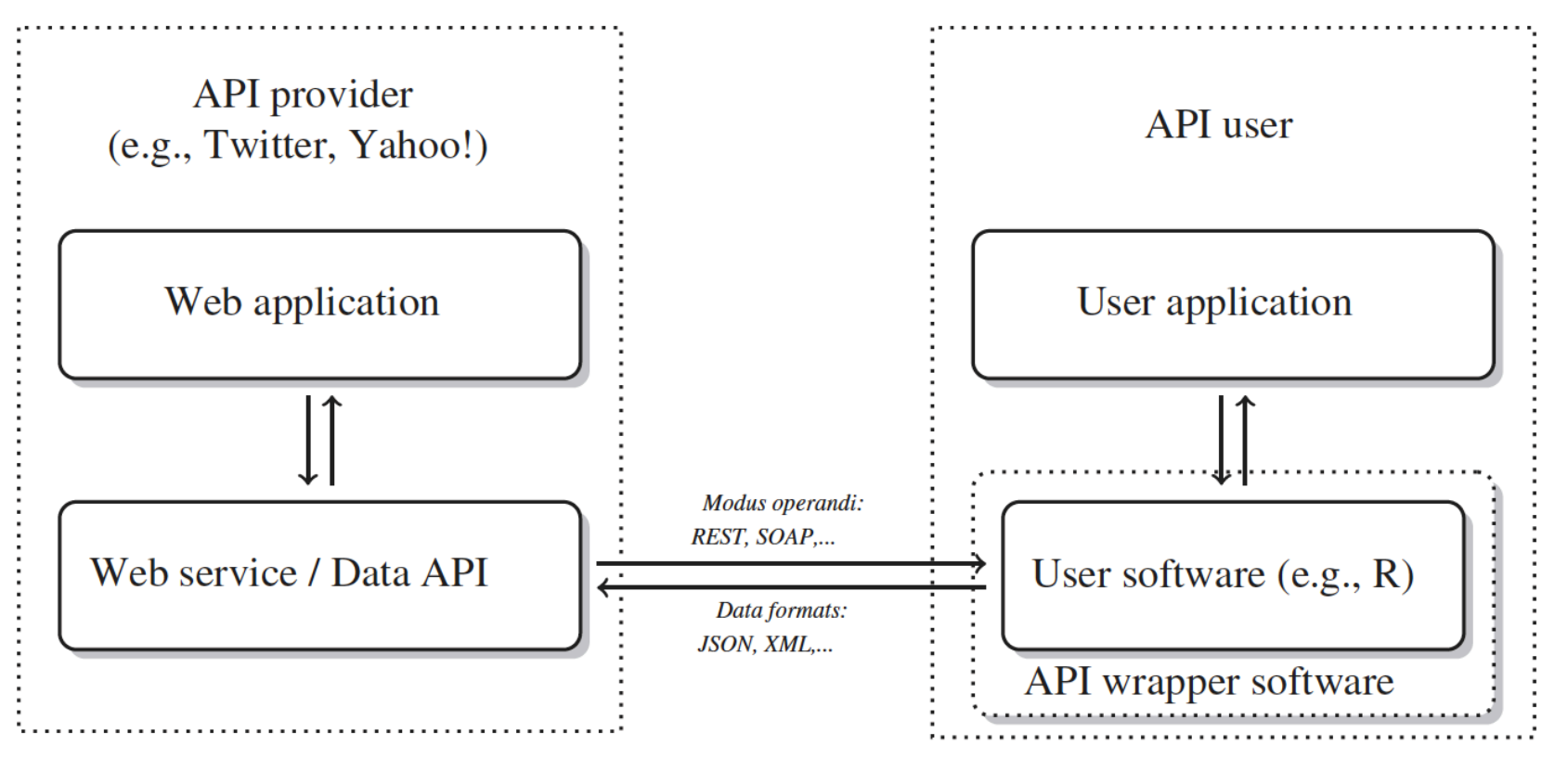
Motivation

- Last week, we talked a bit about the ethics of web scraping.
    - It uses server resources, so we have to be respectful.
- Some web sites provide a way to interact with their data using **APIs**.
- APIs provide a documented way of accessing data, and web sites can control access.

Main definitions

- **API**: Application Programming Interface
  - Allows us to interact with a web application using a programming language.
- **REST**: Representational State Transfer
  - Resources are referenced via URLs and their representations (i.e. data) is transferred via an HTTP request
- An API following the REST standard is called a **RESTful API**.

Why would a website provide an API?



Pros/Cons

- Advantages
    - Documentation on how to access data
    - Structured data
    - Respectful of server-side resources
- Disadvantages
    - May not include the data you want/need
    - Not always free

Example i

- Colorado has a lot of data available:
  `https://data.colorado.gov`
- We will focus on their population projections:
  `https://data.colorado.gov/Demographics/Population-Projections-in-Colorado/q5vp-adf3`
- Looking at the documentation, we see we can specify the county.
    - Using URL parameters!

Example ii

```
library(httr)
# Base URL path
base_url <- paste0("https://data.colorado.gov/",
                   "resource/q5vp-adf3.json?")
full_url <- paste0(base_url, "county=Boulder")

data <- GET(full_url)
status_code(data) # 200 OK
```

```
## [1] 200
```

Example iii

```
# What did we receive?
data$headers$`content-type`
```

```
## [1] "application/json;charset=utf-8"
```

JSON i

- **JSON**: Javascript Object Notation
- It's a common way to share structured data across the web in a human-readable format.
- `Key: value` pairs, grouped using curly braces.
- Main idea: Easy to read and write for both humans and computers.
    - And we have **R** packages to transform them JSON data into `data.frame`s

JSON ii

```
[
  {
    "Id": 0,
    "FirstName": "string",
    "LastName": "string",
    "Name": "string",
    "EmailAddress": "string",
    "TerritoryId": 0
  }
]
```

Example (cont'd) i

```r
library(tidyverse)
library(jsonlite)

data <- fromJSON(content(data, as = "text"))
is.data.frame(data)
```

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

```r
names(data)
```

Example (cont'd) ii

```
## [1] "id" "county" "fipscode" "year"
## [5] "age" "malepopulation" "femalepopulation"
"totalpopulation"
## [9] "datatype"
```

- We can also filter the data using URL parameters
  - `https://dev.socrata.com/docs/queries/`
- **Note**: We need to properly encode the URL first, as it contains spaces.

Example (cont'd) iii

```
# We can filter the data via the URL
full_url <- paste0(base_url, "county=Boulder",
    "&$where=age between 20 and 40")
# Need to encode to proper URL
full_url <- URLencode(full_url)
# Spaces are replaced by %20
str_sub(full_url, 66)
```

```
## [1] "$where=age%20between%2020%20and%2040"
```

Example (cont'd) iv

```
data <- GET(full_url)
status_code(data) # 200 OK
```

```
## [1] 200
```

```
data <- fromJSON(content(data, as = "text"))
range(as.numeric(data$age))
```

```
## [1] 20 40
```

Exercise

Look at the documentation here:

`https://dev.socrata.com/docs/queries/`

Build a URL that will select the following three variables: `year`, `age`, `femalepopulation`; and where `age` is constrained to be between 20 and 40 years old (see previous example).

Use the returned data to plot population projections over time.

Solution i

- Looking at the documentation, we can use `$select=year,age,femalepopulation` as an URL parameter to select the variables we want.

```
full_url <- paste0(base_url, "county=Boulder",
                   "&$where=age between 20 and 40",
                   "&$select=year,age,femalepopulation")

full_url <- URLencode(full_url)
```

Solution ii

```
data <- GET(full_url)
status_code(data) # 200 OK
```

```
## [1] 200
```

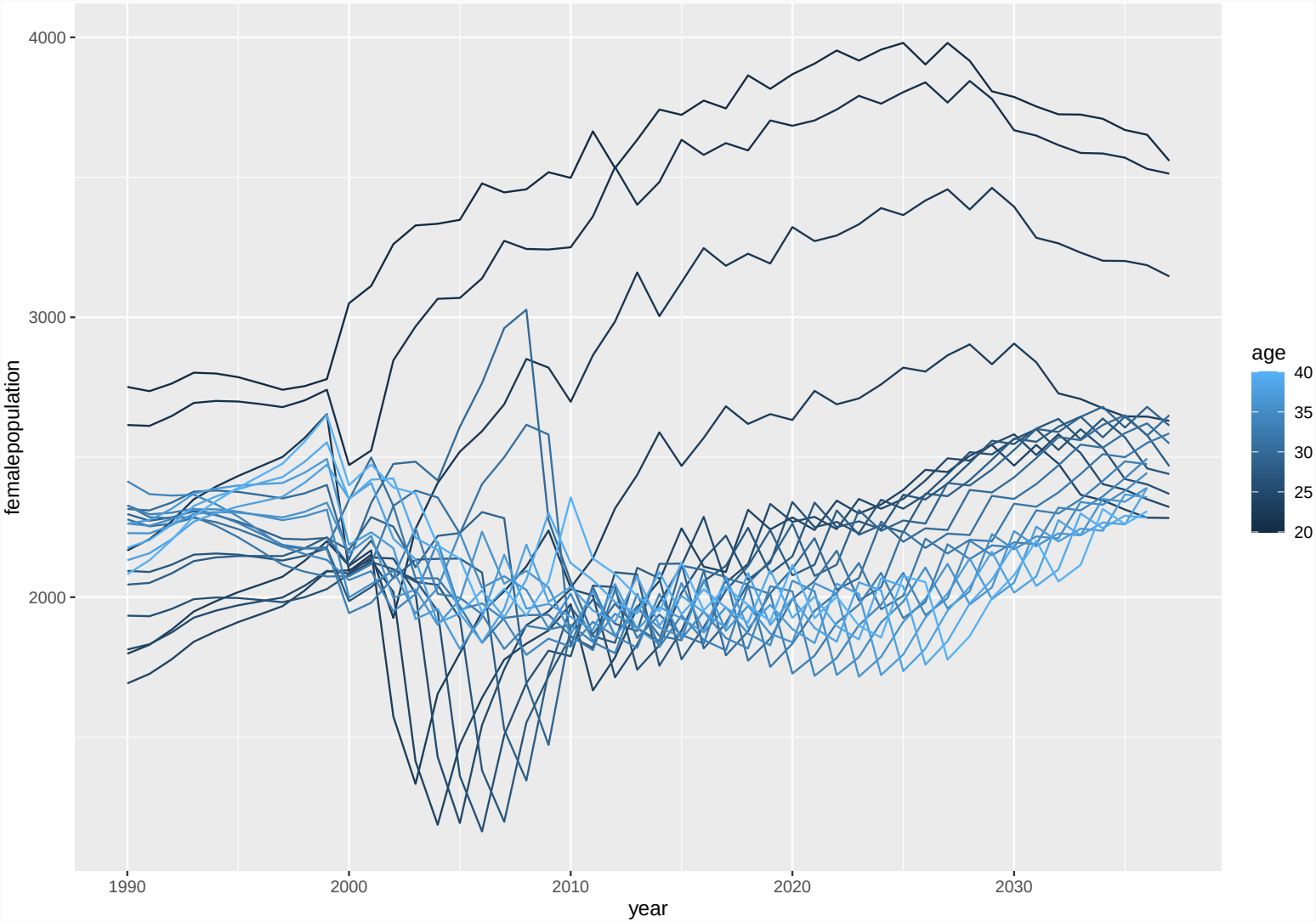
```
data <- fromJSON(content(data, as = "text"))
```

Solution iii

```
library(tidyverse)
# Mutate variables to numeric
data <- data %>%
  mutate(year = as.numeric(year),
         age = as.integer(age),
         femalepopulation = as.numeric(femalepopulation))

data %>%
  ggplot(aes(x = year, y = femalepopulation)) +
  geom_line(aes(group = age, colour = age))
```

Solution iv


4000
3000
2000
femalepopulation
1990
2000
2010
2020
2030
year
age
40
35
30
25
20

Example i

- We will interact with the Winnipeg Transit API.
    - This example is adapted from their web site: `https://api.winnipegtransit.com/home/api/v3/example`
- To access the API you need to register, and you will receive an API key.
- We will start by finding the nearby stops
    - We need to pass longitude (`lon`) and latitude (`lat`) coordinates
    - We need to pass a radius `distance` within which to look for nearby stops.

Example ii

```
# Retrieve API key from environment variable
token <- Sys.getenv("WINNIPEG_TOKEN")

baseurl <- paste0("https://api.winnipegtransit.com/",
                  "v3/stops.json?")

full_url <- paste0(baseurl, "lon=-97.138&lat=49.895&",
                   "distance=250&", "api-key=", token)


data <- GET(full_url)
status_code(data) # 200 OK
```

Example iii

```
## [1] 200
```

```
data <- fromJSON(content(data, as = "text"))
```

```
names(data)
```

```
## [1] "stops"      "query-time"
```

```
glimpse(data$stops)
```

Example iv

```
## Rows: 21
## Columns: 9
## $ key <int> 10763, 10762, 10638, 10627, 10761,
10646, 10644, 10637,~
## $ name <chr> "Eastbound Portage at Main",
"Westbound Portage at Main~
## $ number <int> 10763, 10762, 10638, 10627,
10761, 10646, 10644, 10637,~
## $ direction <chr> "Eastbound", "Westbound",
"Southbound", "Northbound", "~
## $ side <chr> "Farside", "Nearside", "Farside
Opposite", "Nearside", ~
```

Example v

```
## $ street <df[,4]> <data.frame[21 x 4]>
## $ `cross-street` <df[,3]> <data.frame[21 x 3]>
## $ centre <df[,2]> <data.frame[21 x 2]>
## $ distances <df[,1]> <data.frame[21 x 1]>
```

```
pull(data$stops, name)
```

```
## [1] "Eastbound Portage at Main"
## [2] "Westbound Portage at Main"
## [3] "Southbound Main at Pioneer"
## [4] "Northbound Main at Pioneer"
## [5] "Westbound Pioneer at Main"
```

Example vi

```
## [6] "Northbound Fort at Portage"
## [7] "Northbound Fort at Graham North"
## [8] "Southbound Main at Lombard"
## [9] "Eastbound Portage at Fort"
## [10] "Westbound Portage at Westbrook"
## [11] "Eastbound William Stephenson at
Westbrook"
## [12] "Westbound Notre Dame at Albert"
## [13] "Westbound Portage at Garry"
## [14] "Southbound Westbrook at William
Stephenson"
## [15] "Northbound Main at McDermot"
```

Example vii

```
## [16] "Southbound Garry at Portage South"
## [17] "Southbound Garry at Portage North"
## [18] "Northbound Fort at Graham"
## [19] "Eastbound McDermot at Main"
## [20] "Eastbound Graham at Fort (Wpg Square)"
## [21] "Southbound Main at McDermot"
```

- Next, we can pull the stop schedules.
- Each stop has a different endpoint.
    - E.g. for stop 10541, we query `stops/10541`
- We also specify the `max-results-per-route`.

Example viii

```
base_url <- paste0("https://api.winnipegtransit.com/",
                   "v3/stops/10541/schedule.json")
full_url <- paste0(base_url,
                   "?max-results-per-route=2&",
                   "api-key=", token)

data <- GET(full_url)
status_code(data) # 200 OK
```

```
## [1] 200
```

Example ix

```
data <- fromJSON(content(data, as = "text"))
```

```
library(purrr)
data %>%
  pluck("stop-schedule", "route-schedules",
        "scheduled-stops", 1, "times", "arrival")
```

```
##             scheduled           estimated
## 1 2021-03-29T16:15:00 2021-03-29T16:15:00
## 2 2021-03-29T16:21:00 2021-03-29T16:21:00
```

Example x

- **Bonus exercise**: Transform `data` into a table with three columns: Route name, Expected Arrival, and Estimated Arrival.

APIs and R packages

- R packages have been created to interact with most common APIs:
  - `rtweet`: Collecting Twitter data
  - `rnoaa`: NOAA weather data
  - `tradestatistics`: Open Trade international data

Summary

- Web sites use APIs to deliver data as needed, and they sometimes make APIs available to the public.
- APIs often require registration (i.e. using a key) so that they can keep track of your usage.
    - Authentication is sometimes more complex; look at the documentation.
    - Some APIs are not free!
- We should prefer API over scraping whenever possible, as it is more respectful of server resources.