Entering and cleaning data #3

Final project

Groups for final project

- Group 1: Chaoyu, Jessica, Erin, Daniel
- Group 2: Heather, Seré, Khum, Sunny
- Group 3: Rachel, Shayna, Sherry, Matthew
- Group 4: Caroline, Jacob, Eric, Burton
- Group 5: Nikki, Collin, Kyle, Molly

Proposed time for final presentations

Friday, December 13, 2:00 pm–4:00 pm

Pulling online data

APIs

API: "Application Program Interface"

An API provides the rules for software applications to interact. In the case of open data APIs, they provide the rules you need to know to write R code to request and pull data from the organization's web server into your R session.

Often, an API can help you avoid downloading all available data, and instead only download the subset you need.

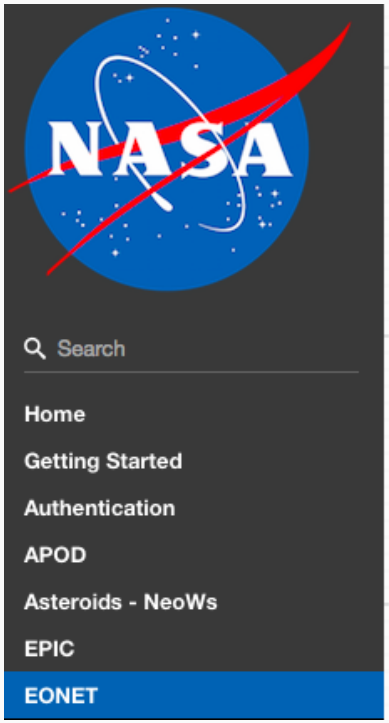
APIs

Strategy for using APIs from R:

- Figure out the API rules for HTTP requests
- Write R code to create a request in the proper format
- Send the request using GET or POST HTTP methods
- Once you get back data from the request, parse it into an easier-to-use format if necessary

API documentation

Start by reading any documentation available for the API. This will often give information on what data is available and how to put together requests.



QUERYING BY DATE(S)

Parameter	Type	Default	Description
date	YYYY-MM-DD	Most Recent Available	Retrieve matadata for all imagery available for a given date.
available_dates	string	All Available Dates	Retrieve a listing of all dates with available imagery.
api_key	string	DEMO_KEY	api.nasa.gov key for expanded usage

EXAMPLE QUERIES

`https://api.nasa.gov/EPIC/api/v1.0/images.php?api_key=DEMO_KEY`

`https://api.nasa.gov/EPIC/api/v1.0/images.php?date=2015-10-31&api_key=DEMO_KEY`

`https://api.nasa.gov/EPIC/api/v1.0/images.php?available_dates&api_key=DEMO_KEY`

More examples and usage tips can be found on the EPIC About Page.

Source: https://api.nasa.gov/api.html#EONET

API key

Many organizations will require you to get an API key and use this key in each of your API requests. This key allows the organization to control API access, including enforcing rate limits per user. API rate limits restrict how often you can request data (e.g., an hourly limit of 1,000 requests per user for NASA APIs).

You should keep this key private. In particular, make sure you do not include it in code that is posted to GitHub.

Example—`riem` package

The `riem` package, developed by Maelle Salmon and an ROpenSci package, is an excellent and straightforward example of how you can use R to pull open data through a web API.

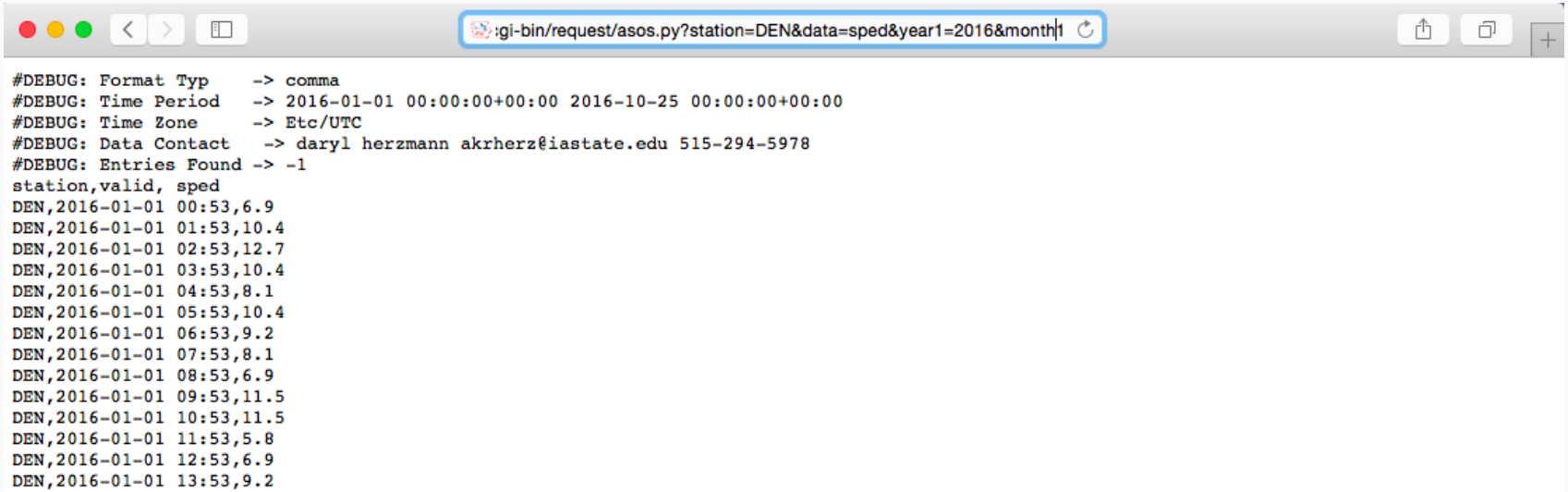
This package allows you to pull weather data from airports around the world directly from the Iowa Environmental Mesonet.

Example—`riem` package

To get a certain set of weather data from the Iowa Environmental Mesonet, you can send an HTTP request specifying a base URL, "https://mesonet.agron.iastate.edu/cgi-bin/request/asos.py/", as well as some parameters describing the subset of dataset you want (e.g., date ranges, weather variables, output format).

Once you know the rules for the names and possible values of these parameters (more on that below), you can submit an HTTP GET request using the `GET` function from the `httr` package.

Example—`riem` package



https://mesonet.agron.iastate.edu/cgi-bin/request/asos.py?station=DEN&data=sknt&year1=2016&month1=6&day1=1&year2=2016&month2=6&day2=30&tz=America%2FDenver&format=comma&latlon=no&direct=no&report_type=1&report_type=2

Using `httr` to get data from a webpage

When you are making an HTTP request using the `GET` or `POST` functions from the `httr` package, you can include the key-value pairs for any query parameters as a list object in the `query` argurment of the function.

```
library(httr)
meso_url <- paste0("https://mesonet.agron.iastate.edu/",
                   "cgi-bin/request/asos.py/")
denver <- GET(url = meso_url,
              query = list(station = "DEN", data = "sped",
                           year1 = "2016", month1 = "6",
                           day1 = "1", year2 = "2016",
                           month2 = "6", day2 = "30",
                           tz = "America/Denver",
                           format = "comma"))
```

Using `httr` to get data from a webpage

The `GET` call will return a special type of list object with elements that include the url you queried and the content of the page at that url:

```
str(denver, max.level = 1, list.len = 6)
```

```
## List of 10
##  $ url        : chr "https://mesonet.agron.iastate.edu/cgi-bi
##  $ status_code: int 200
##  $ headers    :List of 6
##   ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ all_headers:List of 1
##  $ cookies    :'data.frame': 0 obs. of  7 variables:
##  $ content    : raw [1:239447] 23 44 45 42 ...
##   [list output truncated]
##  - attr(*, "class")= chr "response"
```

Using `httr` to get data from a webpage

The `httr` package includes functions to pull out elements of this list object, including:

- `headers`: Pull out the header information
- `content`: Pull out the content returned from the page
- `status_code`: Pull out the status code from the `GET` request (e.g., 200: okay; 404: not found)

Note: For some fun examples of 404 pages, see https://www.creativebloq.com/web-design/best-404-pages-812505

Using `httr` to get data from a webpage

You can use `content` from `httr` to retrieve the contents of the HTTP request we made. For this particular web data, the requested data is a comma-separated file, so you can convert it to a dataframe with `read_csv`:

```
denver %>% content() %>%
  read_csv(skip = 5, na = "M") %>%
  slice(1:3)
```

```
## # A tibble: 3 x 3
##   station valid                sped
##   <chr>   <dttm>              <dbl>
## 1 DEN     2016-06-01 00:00:00   9.2
## 2 DEN     2016-06-01 00:05:00   9.2
## 3 DEN     2016-06-01 00:10:00   6.9
```

Example—`riem` package

The `riem` package wraps up this whole process, so you can call a single function to get in the data you want from the API:

```
library(riem)
denver_2 <- riem_measures(station = "DEN",
                          date_start = "2016-06-01",
                          date_end = "2016-06-30")
denver_2 %>% slice(1:3)
```

```
## # A tibble: 3 x 31
##   station valid                lon   lat  tmpf  dwpf  relh  drct  sknt
##   <chr>   <dttm>             <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 DEN     2016-06-01 00:00:00 -105.  39.8    NA    NA    NA    70     7
## 2 DEN     2016-06-01 00:05:00 -105.  39.8    NA    NA    NA    80     8
## 3 DEN     2016-06-01 00:10:00 -105.  39.8    NA    NA    NA    80     9
## # ... with 22 more variables: p01i <dbl>, alti <dbl>, mslp <dbl>,
## #   vsby <dbl>, gust <dbl>, skyc1 <chr>, skyc2 <chr>, skyc3 <chr>,
## #   skyc4 <chr>, skyl1 <dbl>, skyl2 <dbl>, skyl3 <dbl>, skyl4 <dbl>,
## #   wxcodes <chr>, ice_accretion_1hr <lgl>, ice_accretion_3hr <lgl>,
## #   ice_accretion_6hr <lgl>, peak_wind_gust <dbl>, peak_wind_drct <dbl>,
## #   peak_wind_time <chr>, feel <dbl>, metar <chr>
```

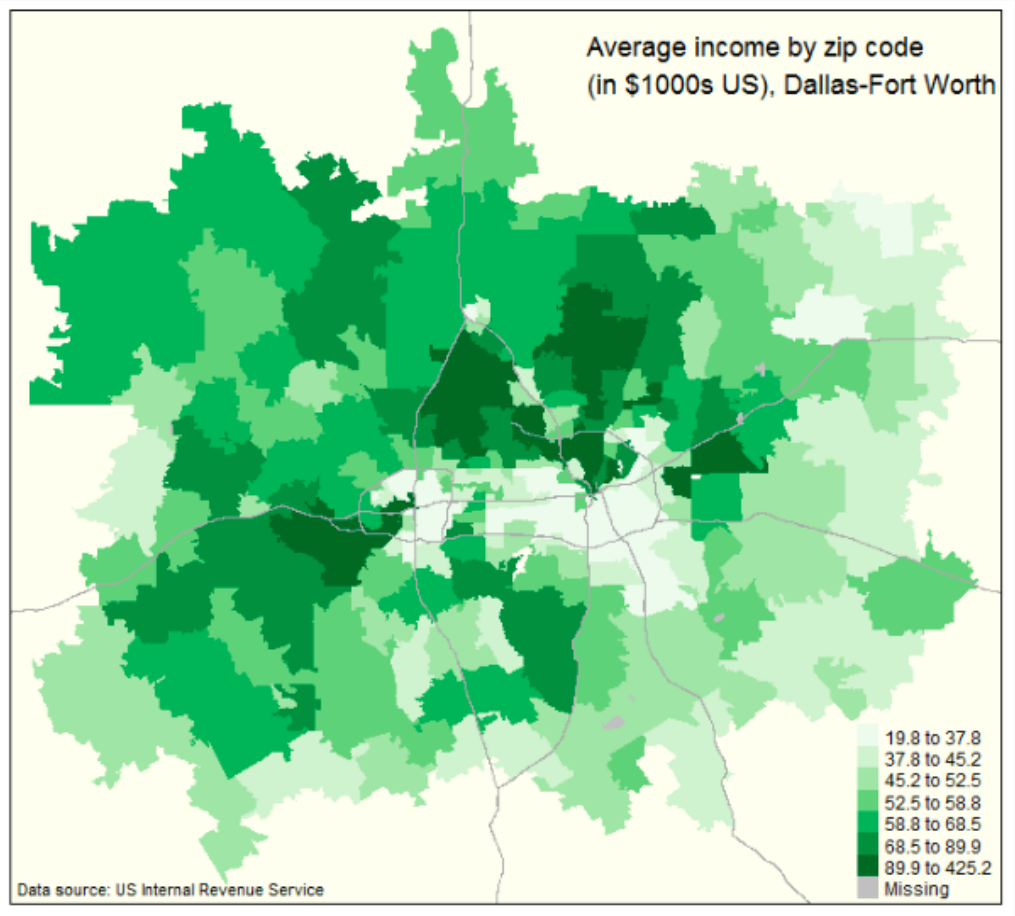
Example R API wrappers

`tigris` package

- Location boundaries
  - States
  - Counties
  - Blocks
  - Tracks
  - School districts
  - Congressional districts
- Roads
  - Primary roads
  - Primary and secondary roads
- Water
  - Area-water
  - Linear-water
  - Coastline
- Other
  - Landmarks
  - Military

`tigris` package

Example from: Kyle Walker. 2016. "tigris: An R Package to Access and Work with Geographic Data from the US Census Bureau". The R Journal.



US Census packages

A number of other R packages also help you access and use data from the U.S. Census:

- `acs`: Download, manipulate, and present American Community Survey and Decennial data from the US Census (see "Working with the American Community Survey in R: A Guide to Using the acs Package", a book available free online through the CSU library)
- `USABoundaries`: Historical and contemporary boundaries of the United States of America
- `idbr`: R interface to the US Census Bureau International Data Base API (e.g., populations of other countries)

rOpenSci

rOpenSci (https://ropensci.org):

> *"At rOpenSci we are creating packages that allow access to data repositories through the R statistical programming environment that is already a familiar part of the workflow of many scientists. Our tools not only facilitate drawing data into an environment where it can readily be manipulated, but also one in which those analyses and methods can be easily shared, replicated, and extended by other researchers."*

rOpenSci

rOpenSci collects a number of packages for tapping into open data for research: https://ropensci.org/packages

Some examples (all descriptions from rOpenSci):

- `AntWeb`: Access data from the world's largest ant database
- `chromer`: Interact with the chromosome counts database (CCDB)
- `gender`: Encodes gender based on names and dates of birth
- `musemeta`: R Client for Scraping Museum Metadata, including The Metropolitan Museum of Art, the Canadian Science & Technology Museum Corporation, the National Gallery of Art, and the Getty Museum, and more to come.
- `rusda`: Interface to some USDA databases
- `webchem`: Retrieve chemical information from many sources. Currently includes: Chemical Identifier Resolver, ChemSpider, PubChem, and Chemical Translation Service.

rnoaa

"Access climate data from NOAA, including temperature and precipitation, as well as sea ice cover data, and extreme weather events"

- Buoy data from the National Buoy Data Center
- Historical Observing Metadata Repository (HOMR))— climate station metadata
- National Climatic Data Center weather station data
- Sea ice data
- International Best Track Archive for Climate Stewardship (IBTrACS)— tropical cyclone tracking data
- Severe Weather Data Inventory (SWDI)

USGS-R Packages

USGS has a very nice collection of R packages that wrap USGS open data APIs: https://owi.usgs.gov/R/

> *"USGS-R is a community of support for users of the R scientific programming language. USGS-R resources include R training materials, R tools for the retrieval and analysis of USGS data, and support for a growing group of USGS-R developers."*

USGS R Packages

USGS R packages include:

- `dataRetrieval`: Obtain water quality sample data, streamflow data, and metadata directly from either the USGS or EPA
- `EGRET`: Analysis of long-term changes in water quality and streamflow, including the water-quality method Weighted Regressions on Time, Discharge, and Season (WRTDS)
- `laketemps`: Lake temperature data package for Global Lake Temperature Collaboration Project
- `lakeattributes`: Common useful lake attribute data
- `soilmoisturetools`: Tools for soil moisture data retrieval and visualization

Other R API wrappers

Here are some examples of other R packages that faciliate use of an API for open data:

- `twitteR`: Twitter
- `Quandl`: Quandl (financial data)
- `RGoogleAnalytics`: Google Analytics
- `WDI`, `wbstats`: World Bank
- `GuardianR`, `rdian`: The Guardian Media Group
- `blsAPI`: Bureau of Labor Statistics
- `rtimes`: New York Times

R and APIs

Find out more about writing API packages with this vignette for the httr package: https://cran.r-project.org/web/packages/httr/vignettes/api-packages.html.

This document includes advice on error handling within R code that accesses data through an open API.

Group work

We'll take a break now to set up the GitHub repositories for your final group project.

Setting up for group project

Get together with your group for the final project, and you will be setting up your GitHub repository to use for that project:

1. Select one person in your group to "host" the repository.
2. Go through all the steps we took for Homework 5 to create a local R Project, put it under git version control, and connect it to a remote GitHub repository. This only needs to be done for the person "hosting" the repository.
3. Once one person in your group has the GitHub repository you'll all use, that person should go to GitHub and make the other group members collaborators. Within the GitHub repository, go to the "Settings" tab and go to "Collaborators". You should be able to find and add other GitHub members. Once you do, your other group members should get email invitations.

(More on next slide)

Setting up for group project

4. For the other group members, accept your email invitation to collaborate on the repository. Now you can "clone" this repository to your own computer, to have a local version to work on. Open a command line, use `cd` to change into the directory where you'd like to save the R project, and then use:

```
git clone git@github.com:geanders/ex_repo.git
```

But replace `geanders` with the GitHub handle for your group member who is hosting the account and `ex_repo` with the repository name for your GitHub repository for the project.

Setting up for group project

5. Now have one of your group members create an RMarkdown document to use for your final project report. That person should create that document on their local version of the repository, save their changes, commit them, and then push them to GitHub. Check online to make sure it went through. Everyone else in the group should then "pull" (use the down arrow in your "Git" box in RStudio) to pull that change to their local version of the repository.
6. Now explore resolving commit conflicts. Have two members of your group open the RMarkdown document and change the "author" input in the YAML of the document to include their own name. They both should save and commit this change locally. Have one of the two push the change to GitHub. The second person should then pull the latest version of the repo from GitHub. There will be commit conflicts (you'll see a filled box for the file in the GitHub page). Open that file and look for the section that starts with "<<<<". Look at the two versions given for that part of the document, decide what you want as the final version, clean up the "<<<", "===", and ">>>" lines, and then commit and push the changes.

Setting up for group project

7. Talk with your group members about how you'd like to share the work for the homework. Go to the "Issues" page for the GitHub repository and create some Issues for the tasks your group will need to do. Try using the `@` notation to reference other people in your group in the message (e.g., `@geanders` would reference me and send me an email about the Issue message).
8. Create one practice Issue that's something like "Test closing an issue with commit". Then have one group member make some change on their local version of the project, commit that change with the message "Close #[x]", where [x] is the number of the test Issue you opened, and then push the changes. Go online to GitHub and you should see that that Issue is now "Closed".