

Lecture 6: Data Acquisition - Client-Side Websites and APIs

James Sears*

AFRE 891 SS 24

Michigan State University

*Parts of these slides related to APIs are adapted from [“Data Science for Economists”](#) by Grant McDermott.

Table of Contents

1. **Prologue**
2. **APIs**
3. **Explicit API Endpoint, No Key**
4. **Explicit API Endpoint with Key**
5. **Hidden APIs**
6. **Considerate Web Scraping**

Prologue

Client-Side

Last time, we learned workflows for scraping **server-side** websites.

- Static sites with **rvest**
- Dynamic or interactive sites with **selenium** and **rvest**

Today, we'll turn our attention to **client-side** websites.

Packages for Today

First, a few packages for today:

```
pacman::p_load(lubridate, tidyverse, jsonlite, httr, usethis, fredr, XML,
```

Also recommended: [JSONView](#), a browser extension for Firefox and Chrome that renders JSON output nicely.

APIs

Client-Side Web Sites

Recall that websites built using a **client-side** framework typically involve the following steps:

- Visit a URL containing an **empty template** of static content
- In the process of opening the URL, your browser sends a **request** to the host server
- If the request is valid, the server issues a **response** and populates the page dynamically with the requested data
- The final page you see is thus a mix of static and dynamic content

APIs

An **API (Application Programming Interface)** is a set of rules that carries out all this requesting, responding, and rendering. Some key concepts:

- **Server:** A powerful computer that runs an API.
- **Client:** A program that exchanges data with a server through an API.
- **Protocol:** The "etiquette" underlying how computers talk to each other (e.g. HTTP).
- **Methods:** The "verbs" that clients use to talk with a server.
 - The main one that we'll be using is GET (i.e. ask a server to retrieve information).
 - Other common methods are POST, PUT and DELETE.

APIs

An **API (Application Programming Interface)** is a set of rules that carries out all this requesting, responding, and rendering. Some key concepts:

- **Request:** What the client asks of the server (see Methods above).
- **Response:** The server's response. This includes:
 - Status Code (e.g. "404" if not found, or "200" if successful).
 - Header (i.e. meta-information about the response).
 - Body (i.e. the actual content that we're interested in).

API Endpoints


With web APIs, we can access information *directly* from the API database if we can specify the correct URL, known as an **API endpoint**.

- Similar to normal URLs, but the resulting page is... visually unappealing

← → ↺ 🏠 [api.census.gov/data/2022/acs/acs1/subject?get=NAME,group\(S0101\)&for=us:1&key=5bc2880e32458e32f01df20e05b627ce11ec335e](https://api.census.gov/data/2022/acs/acs1/subject?get=NAME,group(S0101)&for=us:1&key=5bc2880e32458e32f01df20e05b627ce11ec335e)

[illegible]

API Endpoints

The reason the site looks like hot  is that its structured and formatted in a very particular way (that can be read directly into R!).

Usually these pages are formatted in one of two ways: XML or JSON

- We don't need too worry too much about the particular syntax of either

In general there are **three types** of API Endpoints:

1. **Explicit** API Endpoint, **no API Key**
2. **Explicit** API Endpoint, **API Key Required**
3. **Hidden** API Endpoints

Let's work through an example workflow for each. But first...

Some APIs that are free & legal

Some fun APIs:

- [TikTok \(unofficial\)](#)
- [Twitter](#)
- [Spotify](#)
- [YouTube](#)

This list of [Public APIs](#) has many more.

R API Packages

Many R packages have been built to make working with APIs easier. Some examples:

- **tidycensus** for [American Community Survey microdata and other Census Bureau data](#)
- **fredr** for [more easily doing what we will do today](#)
- **Quandl** for [Nasdaq and other financial information](#)
- **rtweet** for [Twitter data](#)
- **rnoaa** for [NOAA weather data](#)
- **prism** for [PRISM gridded climate data](#) (we'll work with this later on)
- **TADA** for [Water Quality Portal Data \(EPA, USGS\)](#)
- **rnassqs** for USDA's [QuickStats API](#) (agricultural data)
- **osmdata** for [OpenStreetMap spatial data](#)

Explicit API Endpoint, No Key

Explicit API Endpoint, No Key

Let's work through an example of the first type of API: **explicit API without a key**.

Let's use the impressive **NYC Open Data** to look at... 🌲.

Seriously, trees.

Application 1: NYC Tree Census

First, open the [Tree Census page](#) in your browser (if you haven't done so already)

NYC OpenData

HomeDataAbout ▾Learn ▾AlertsContact Us

Q

Sign In

Actions ▾Export

2015 Street Tree Census - Tree Data

Environment

Street tree data from the TreesCount! 2015 Street Tree Census, conducted by volunteers and staff organized by NYC Parks & Recreation and partner organizations. Tree data collected includes tree species, diameter and perception of health. Accompanying blockface data is available indicating status of data collection and data release...

[Read more ▾](#)

Last Updated

May 9, 2022

Data Provided By

Department of Parks and Recreation (DPR)

About this Dataset

Updated

May 9, 2022

Data Last Updated

October 4, 2017

Metadata Last Updated

May 9, 2022

Date Created

June 3, 2016

Views

62.2K

Downloads

17.7K

Dataset Information

Agency	Department of Parks and Recreation (DPR)
--------	--

Update

Update Frequency	Historical data
Automation	No
Date Made Public	6/3/2016

Attachments

StreetTreeCensus2015TreesDataDictionary20161102.pdf

Application 1: NYC Tree Census

Click on **Actions > API** in upper-right to display the API Endpoint

NYC OpenData Home Data About ▾ Learn ▾ Alerts Contact Us 🔍 Sign In

2015 Street Tree Census - Tree Data Environment

Street tree data from the TreesCount! 2015 Street Tree Census, conducted by volunteers and staff organized by NYC Parks & Recreation and partner organizations. Tree data collected includes tree species, diameter and perception of health. Accompanying blockface data is available indicating status of data collection and data release...

[Read more ▾](#)

Last Updated
May 9, 2022

Data Provided By
Department of Parks and Recreation (DPR)

Actions ▾ Export

- Query data
Group, aggregate and more
- Visualize ▸
- API** API
- Access via oData
- Share

About this Dataset

Updated
May 9, 2022

Data Last Updated: October 4, 2017 | Metadata Last Updated: May 9, 2022

Date Created: June 3, 2016

Views: **62.2K** | Downloads: **17.7K**

Dataset Information

Agency	Department of Parks and Recreation (DPR)
--------	--

Update

Update Frequency	Historical data
Automation	No
Date Made Public	6/3/2016

Attachments

- StreetTreeCensus2015TreesDataDictionary20161102.pdf

Application 1: NYC Tree Census

Click on **Actions > API** in upper-right to display the API Endpoint

The screenshot shows the NYC OpenData website interface. The top navigation bar includes links for Home, Data, About, Learn, Alerts, and Contact Us, along with a search icon and a Sign In button. On the right side of the page, there are buttons for 'Actions' and 'Export'. The main content area displays the '2015 Street Tree Census - Tree Data' dataset page. An 'Export dataset' modal is open, showing options to 'Download file' or 'API Endpoint'. The 'API Endpoint' option is selected, and the modal displays the API endpoint URL: `https://data.cityofnewyork.us/resource/uvpi-gqnh.js`. A warning message indicates that the default API limit of 1,000 rows has been exceeded. The modal also includes links for 'API documentation' and 'Developer portal', and buttons for 'Cancel' and 'Copy to clipboard'.

NYC OpenData

Home Data About Learn Alerts Contact Us

Sign In

Actions Export

2015 Street Tree Census - Tree Data

Street tree data from the TreesCount project, collected and staff organized by NYC Parks and Recreation. The data includes tree species, diameter, and blockface data is available indicating the location of the tree.

Read more

About this Dataset

Updated
May 9, 2022

Data Last Updated: October 4, 2017
Metadata Last Updated: May 9, 2022

Date Created: June 3, 2016

Views: **62.2K**
Downloads: **17.7K**

Export dataset

2015 Street Tree Census - Tree Data

Download file API Endpoint

Data Format
JSON

All data (683788 rows)

Warning: Default API limit exceeded
Our API has a default limit of providing 1,000 rows. [Learn more](#) about how you can modify the default limit.

API Endpoint
`https://data.cityofnewyork.us/resource/uvpi-gqnh.js`

[API documentation](#) [Developer portal](#)

Cancel Copy to clipboard

StreetTreeCensus2015TreesDataDictionary20161102.pdf

Application 1: NYC Tree Census

Copy/Paste or clickc "go to" to open the API endpoint link in another tab.
You'll see a bunch of JSON text.



The screenshot shows a web browser window with the address bar displaying `data.cityofnewyork.us/resource/uvpi-gqnh.json`. The main content area shows a JSON array of tree census records. Each record is an object with various attributes including tree ID, block ID, creation date, tree details (dbh, stump diam, curb loc, status, health, spc latin, spc common), steward, guards, sidewalk, user type, problems, root stone, root grate, root other, trunk wire, trunk light, trunk other, brch light, brch shoe, brch other, address, zipcode, zip city, cb_num, borocode, boroname, cnclldist, st_assem, st_senate, nta, nta_name, boro_ct, state, latitude, longitude, x_sp, y_sp, council_district, census_tract, bin, and bbl.

```
[{"tree_id": "180683", "block_id": "348711", "created_at": "2015-08-27T00:00:00.000", "tree_dbh": "3", "stump_diam": "0", "curb_loc": "OnCurb", "status": "Alive", "health": "Fair", "spc_lat": "Acer rubrum", "spc_common": "red maple", "steward": "None", "guards": "None", "sidewalk": "NoDamage", "user_type": "TreesCount Staff", "problems": "None", "root_stone": "No", "root_grate": "No", "root_other": "No", "trunk_wire": "No", "trnk_light": "No", "trnk_other": "No", "brch_light": "No", "brch_shoe": "No", "brch_other": "No", "address": "108-005 70 AVENUE", "zipcode": "11375", "zip_city": "Forest Hills", "cb_num": "406", "borocode": "4", "boroname": "Queens", "cnclldist": "29", "st_assem": "28", "st_senate": "16", "nta": "QN17", "nta_name": "Forest Hills", "boro_ct": "4073900", "state": "New York", "latitude": "40.72309177", "longitude": "-73.84421522", "x_sp": "1027431.148", "y_sp": "202756.7687", "council_district": "29", "census_tract": "739", "bin": "4052307", "bbl": "4022 210001"}, {"tree_id": "200540", "block_id": "315986", "created_at": "2015-09-03T00:00:00.000", "tree_dbh": "21", "stump_diam": "0", "curb_loc": "OnCurb", "status": "Alive", "health": "Fair", "spc_lat": "Quercus palustris", "spc_common": "pin oak", "steward": "None", "guards": "None", "sidewalk": "Damage", "user_type": "TreesCount Staff", "problems": "Stones", "root_stone": "Yes", "root_grate": "No", "root_other": "No", "trunk_wire": "No", "trnk_light": "No", "trnk_other": "No", "brch_light": "No", "brch_shoe": "No", "brch_other": "No", "address": "147-074 7 AVENUE", "zipcode": "11357", "zip_city": "Whitestone", "cb_num": "407", "borocode": "4", "boroname": "Queens", "cnclldist": "19", "st_assem": "27", "st_senate": "11", "nta": "QN49", "nta_name": "Whitestone", "boro_ct": "4097300", "state": "New York", "latitude": "40.79411067", "longitude": "-73.81867946", "x_sp": "1034455.701", "y_sp": "228644.8374", "council_district": "19", "census_tract": "973", "bin": "4101931", "bbl": "4044 750045"}, {"tree_id": "204026", "block_id": "218365", "created_at": "2015-09-05T00:00:00.000", "tree_dbh": "3", "stump_diam": "0", "curb_loc": "OnCurb", "status": "Alive", "health": "Good", "spc_lat": "Gleditsia triacanthos var. inermis", "spc_common": "honeylocust", "steward": "1or2", "guards": "None", "sidewalk": "Damage", "user_type": "Volunteer", "problems": "None", "root_stone": "No", "root_grate": "No", "root_other": "No", "trunk_wire": "No", "trnk_light": "No", "trnk_other": "No", "brch_light": "No", "brch_shoe": "No", "brch_other": "No", "address": "390 MORGAN AVENUE", "zipcode": "11211", "zip_city": "Brooklyn", "cb_num": "301", "borocode": "3", "boroname": "Brooklyn", "cnclldist": "34", "st_assem": "50", "st_senate": "18", "nta": "BK90", "nta_name": "East Williamsburg", "boro_ct": "3044900", "state": "New York", "latitude": "40.71758074", "longitude": "-73.9366077", "x_sp": "1001822.831", "y_sp": "200716.8913", "council_district": "34", "census_tract": "449", "bin": "3338310", "bbl": "30288 70001"}, {"tree_id": "204337", "block_id": "217969", "created_at": "2015-09-05T00:00:00.000", "tree_dbh": "10", "stump_diam": "0", "curb_loc": "OnCurb", "status": "Alive", "health": "Good", "spc_lat": "Gleditsia triacanthos var. inermis", "spc_common": "honeylocust", "steward": "None", "guards": "None", "sidewalk": "Damage", "user_type": "Volunteer", "problems": "Stones", "root_stone": "Yes", "root_grate": "No", "root_other": "No", "trunk_wire": "No", "trnk_light": "No", "trnk_other": "No", "brch_light": "No", "brch_shoe": "No", "brch_other": "No", "address": "1027 GRAND STREET", "zipcode": "11211", "zip_city": "Brooklyn", "cb_num": "301", "borocode": "3", "boroname": "Brooklyn", "cnclldist": "34", "st_assem": "53", "st_senate": "18", "nta": "BK90", "nta_name": "East Williamsburg", "boro_ct": "3044900", "state": "New York", "latitude": "40.71353749", "longitude": "-73.93445616", "x_sp": "1002420.358", "y_sp": "199244.2531", "council_district": "34", "census_tract": "449", "bin": "3338342", "bbl": "3029 250001"}]
```

Application 1: NYC Tree Census

If you installed JSONView, it will look a lot nicer

- A lot like a transposed dataframe row...

data.cityofnewyork.us/resource/uvpi-gqnh.json

```
[ {  
  "tree_id": "180683",  
  "block_id": "348711",  
  "created_at": "2015-08-27T00:00:00.000",  
  "tree_dbh": "3",  
  "stump_diam": "0",  
  "curb_loc": "OnCurb",  
  "status": "Alive",  
  "health": "Fair",  
  "spc_latin": "Acer rubrum",  
  "spc_common": "red maple",  
  "steward": "None",  
  "guards": "None",  
  "sidewalk": "NoDamage",  
  "user_type": "TreesCount Staff",  
  "problems": "None",  
  "root_stone": "No",  
  "root_grate": "No",  
  "root_other": "No",  
  "trunk_wire": "No",  
  "trnk_light": "No",  
  "trnk_other": "No",  
  "brch_light": "No",  
  "brch_shoe": "No",  
  "brch_other": "No",  
  "address": "108-005 70 AVENUE",  
  "zipcode": "11375",  
  "zip_city": "Forest Hills",  
  "cb_num": "406",  
  "borocode": "4",  
  "boroname": "Queens",  
  "cnclldist": "29",  
  "st_assem": "28",  
  "st_senate": "16",  
  "nta": "QN17",  
  "nta_name": "Forest Hills",  
  "boro_ct": "4073900",  
  "state": "New York",  
  "latitude": "40.72309177",  
  "longitude": "-73.84421522",  
  "x_sp": "1027431.148",  
  "y_sp": "202756.7687"  
}
```

Extensions

Full access
These extensions can see and change information on this site.

- JSONView
- SelectorGadget

Manage extensions

Load JSON into R

Next, use the `fromJSON()` function in the [excellent jsonlite package](#) to read JSON into R.

```
nyc_trees <- fromJSON("https://data.cityofnewyork.us/resource/uvpi-gqnh.json")
as.data.frame(nyc_trees)
```

	tree_id	block_id	created_at	tree_dbh	stump_diam
## 1	180683	348711	2015-08-27T00:00:00.000	3	0
## 2	200540	315986	2015-09-03T00:00:00.000	21	0
## 3	204026	218365	2015-09-05T00:00:00.000	3	0
## 4	204337	217969	2015-09-05T00:00:00.000	10	0
## 5	189565	223043	2015-08-30T00:00:00.000	21	0
## 6	190422	106099	2015-08-30T00:00:00.000	11	0
## 7	190426	106099	2015-08-30T00:00:00.000	11	0
## 8	208649	103940	2015-09-07T00:00:00.000	9	0
## 9	209610	407443	2015-09-08T00:00:00.000	6	0
## 10	192755	207508	2015-08-31T00:00:00.000	21	0
## 11	203719	302371	2015-09-05T00:00:00.000	11	0
## 12	203726	302371	2015-09-05T00:00:00.000	8	0

Aside: API Limits

Note that while the full census of NYC trees contains over 700,000 trees, we only downloaded a small portion.

By default, the API limits us to 1,000 rows.

For our use case (seeing an example of the workflow) this isn't an issue, but reading the documentation reveals a way to get around that (if so desired).

Punchline: always read the API documentation!

Work with the Data

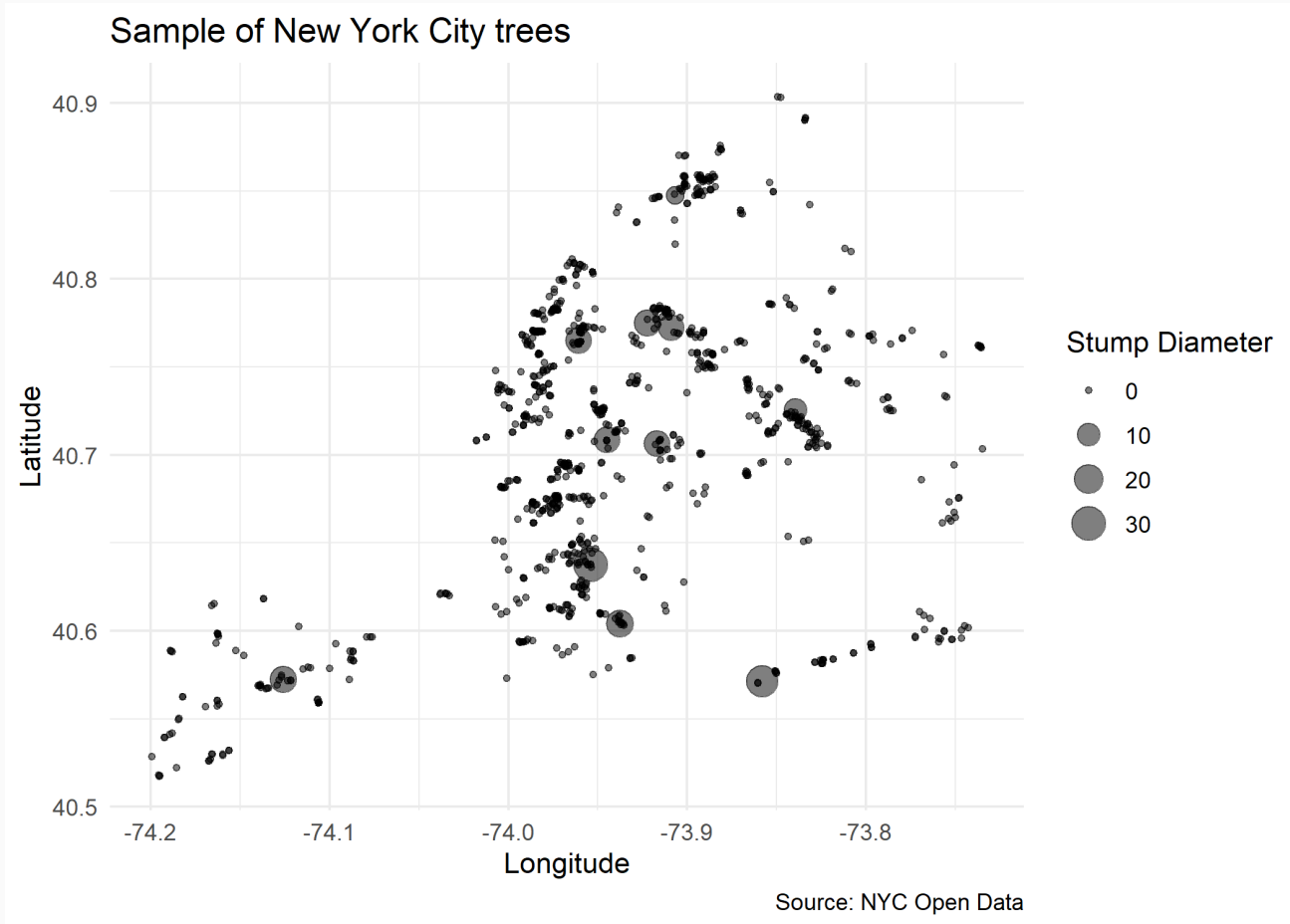
Now we have the data, accessed through an API Endpoint!

Seriously, that's it. Now we can do whatever we want with the data - how about plotting it?

```
nyc_trees %>%
  select(longitude, latitude, stump_diam, spc_common, spc_latin, tree_id) %>%
  mutate_at(vars(longitude:stump_diam), as.numeric) %>%
  ggplot(aes(x=longitude, y=latitude, size=stump_diam)) +
  geom_point(alpha=0.5) +
  scale_size_continuous(name = "Stump Diameter") +
  labs(
    x = "Longitude", y = "Latitude",
    title = "Sample of New York City trees",
    caption = "Source: NYC Open Data"
  ) +
  theme_minimal()
```

Plotting Some Trees

```
.small[
```



Explicit API Endpoint with Key

Explicit API Endpoint with Key

For the second application, we're going to focus on the more common circumstance: an API endpoint that **requires an API key**.

An **API key** is a way to add a unique ID signature to every API call you make

- Required for all APIs linked to gov't agencies or institutions (e.g. Census, BEA).

Let's do an application working with the **Federal Reserve Bank of St. Louis (FRED) API**

Application 2: FRED API

FRED makes available a **ton of U.S. economic data** that can easily be accessed via API.



The only real difference from the last example: we first have to **register for an API key**


Application 2: FRED API

First, navigate to the [FRED API page](https://fred.stlouisfed.org/docs/api/fred/#)

→ fred.stlouisfed.org/docs/api/fred/#

ECONOMIC RESEARCH

Economic Research Resources ▼ Switch Products ▼ My Account ▼

FRED  Your trusted data source since 1991.

ECONOMIC DATA | ST. LOUIS FED

Search FRED ...

Release Calendar FRED Tools ▼ FRED News FRED Blog About FRED ▼

API Keys | Terms of Use

FRED® API

[General Documentation](#) | [API](#) | [Maps API](#) | [Toolkits](#)

The FRED® API is a web service that allows developers to write programs and build applications that retrieve economic data from the FRED® and ALFRED® websites hosted by the [Economic Research Division](#) of the [Federal Reserve Bank of St. Louis](#). Requests can be customized according to data source, release, category, series, and other preferences.

General Documentation

- Overview
- What is FRED®?
- What is ALFRED®?
- FRED® versus ALFRED®
- Real-Time Periods
- Errors


API

Categories


- [fred/category](#) - Get a category.
- [fred/category/children](#) - Get the child categories for a specified parent category.
- [fred/category/related](#) - Get the related categories for a category.
- [fred/category/series](#) - Get the series in a category.
- [fred/category/tags](#) - Get the tags for a category.

Application 2: FRED API

Click on "API Keys" in the upper-right then "Request or view your API keys"




FRED
ECONOMIC DATA | ST. LOUIS FED



Your trusted data source
since 1991.

Economic Research Resources ▼ Switch Products ▼ My Account ▼

Search FRED ... 

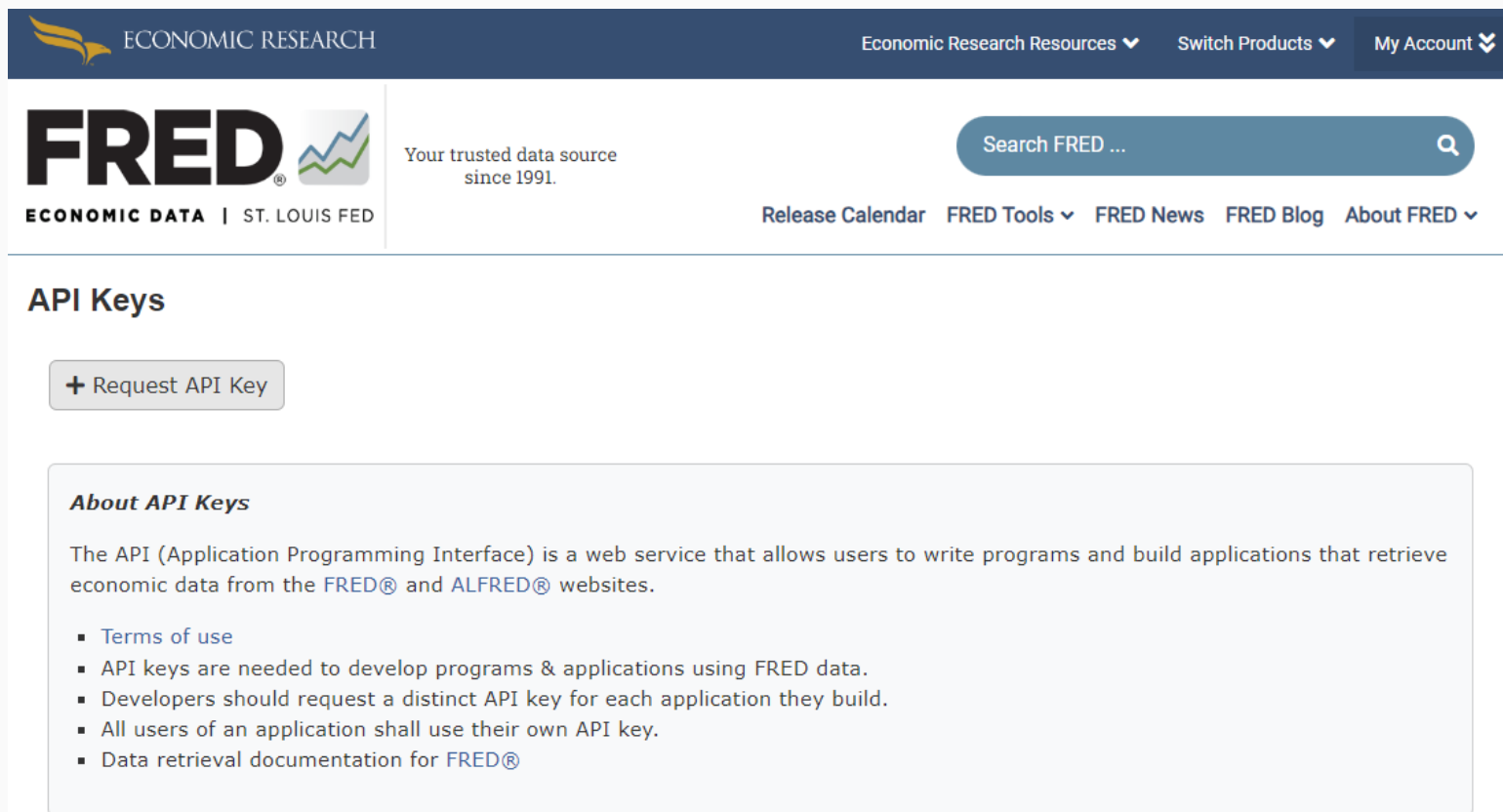
Release Calendar FRED Tools ▼ FRED News FRED Blog About FRED ▼

[API Keys](#) | [Terms of Use](#)

Application 2: FRED API

Make an account, then request an API Key


- My Account > API Keys > Request API Key




The screenshot shows the FRED (Federal Reserve Economic Data) website's 'Request API Key' page. The header is dark blue with the 'ECONOMIC RESEARCH' logo and navigation links: 'Economic Research Resources', 'Switch Products', and 'My Account'. Below the header, the FRED logo is prominently displayed next to the tagline 'Your trusted data source since 1991.' and 'ECONOMIC DATA | ST. LOUIS FED'. A search bar is located on the right. The main content area is titled 'API Keys' and features a '+ Request API Key' button. Below this, a section titled 'About API Keys' explains that the API is a web service for retrieving economic data and lists four key points: terms of use, the need for distinct API keys per application, the requirement for users to have their own keys, and a link to data retrieval documentation.

ECONOMIC RESEARCH

Economic Research Resources ▼ Switch Products ▼ My Account ▼

FRED  Your trusted data source since 1991.

ECONOMIC DATA | ST. LOUIS FED

Search FRED ... 

Release Calendar FRED Tools ▼ FRED News FRED Blog About FRED ▼

API Keys

[+ Request API Key](#)

About API Keys

The API (Application Programming Interface) is a web service that allows users to write programs and build applications that retrieve economic data from the [FRED®](#) and [ALFRED®](#) websites.

- [Terms of use](#)
- API keys are needed to develop programs & applications using FRED data.
- Developers should request a distinct API key for each application they build.
- All users of an application shall use their own API key.
- Data retrieval documentation for [FRED®](#)

Setting API Keys in R

Treat your API keys **like passwords**

- You're responsible for following API terms & conditions, and you can't control that if someone else is using your key

This means we shouldn't ever **harcode API keys into R scripts**.

Instead, we can define them as **environment variables**, either

1. For the current R session only
2. Globally across all R sessions

Environment Variables: Current Session

To define an environment variable for **your current session only**, run the following **in your console**:¹

```
Sys.setenv(FRED_API_KEY = "abcdefghijklmnopqrstuvwxyz0123456789")
```

Replacing the fake API with your actual API.

You can then use an environment variable in your script with

```
# assign to R object and print out  
my_key ← Sys.getenv("FRED_API_KEY")  
my_key
```

¹ setting `Sys.setenv()` in scripts defeats the entire purpose!

Environment Variables: Globally

To set an environment variable **globally across all sessions**, we'll need to add a special `~/.Renviron` file.

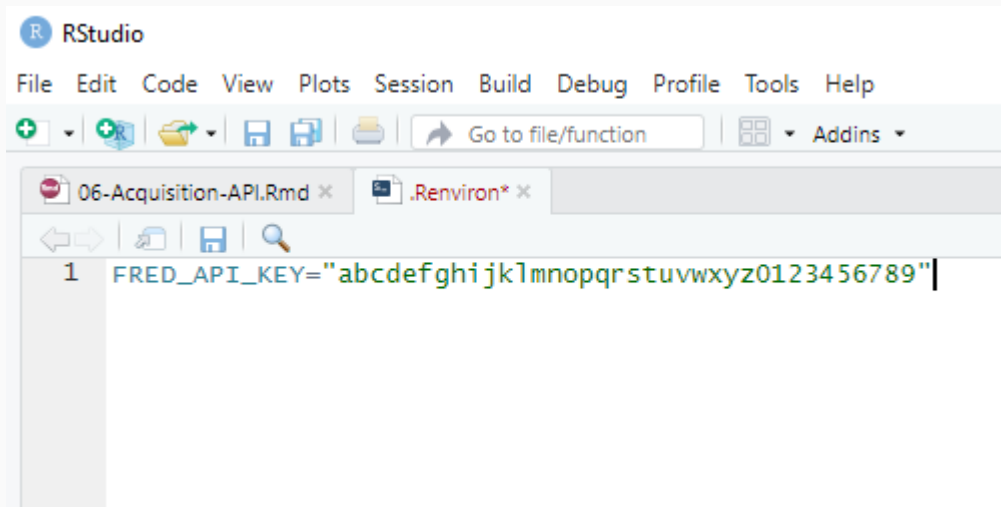
- Text file that lives in your home directory (hence the `~/` path)

To edit, we can use the ``usethis::edit_r_environ()` function:

```
usethis::edit_r_environ()
```

Environment Variables: Globally

This opens your `~/.Renviron` file in a new tab, where you can easily add a line defining your API key.



Hit save, then restart R or refresh your environment session to use it:

```
## Only necessary when reading in a new Renviron variable  
readRenviron("~/Renviron")
```

Using the FRED API

With your API key set, let's figure out how to use the FRED API.

Looking at the [FRED API documentation](#) you'll see

- There are many types of requests you can make of FRED using the API
- If we want the underlying data for a data series, we want the [fred/series/observations](#) path.

Using the FRED API

Clicking the link takes us to [specific documentation for that path]
(https://fred.stlouisfed.org/docs/api/fred/series_observations.html)

Here we can learn

1. In which **file formats** the API will return data
 - XML, JSON, zipped tab delimited, zipped Excel
2. The exact **syntax** for making a request
 - Potentially specific to each format
3. The **parameters** that you can provide
 - e.g. function arguments

Using the FRED API

One required parameter is `series_id`.

- Let's look at the median home listing price in East Lansing.
- By searching the FRED website, I found that the **ID** for this series is `MELIPRMSA29620`.

For the `file_type`, you can use either XML or JSON.

- Both can easily be read into R.
- Since we already saw JSON, let's try using XML.

Try out the API right in your browser. Paste this into your location bar, replacing the fake API key with your own:

```
https://api.stlouisfed.org/fred/series/observations?  
series_id=MELIPRMSA29620&api_key=abcdefghijklmnopqrstuvwxyz123456&file_type=
```

Using the FRED API

To use the API in R, we'll use the tidyverse package **httr**. First we need to define some variables with the information we need:

```
url = "https://api.stlouisfed.org/" # base URL
endpoint = "fred/series/observations" # endpoint path
params = list(api_key = Sys.getenv("FRED_API_KEY"), # API key
              file_type = "json", # format
              series_id = "MELIPRMSA29620") # series we want
```

Now we can use the **GET** method to request data from the server:

```
response = httr::GET(url = url, path = endpoint, query = params)
response
```

```
Response [https://api.stlouisfed.org/fred/series/observations?api_key=[YOUR_API_KEY]]
Date: 2024-02-19 20:23
Status: 200
Content-Type: application/json; charset=UTF-8
Size: 9.11 kB
```

Using the FRED API

The response is a complex list object. Extracting the XML content:

```
json ← response %>%  
  httr::content(type = "text") %>% # from httr package  
  jsonlite::fromJSON() # Parse the JSON file
```

Viewing this object reveals that what we really want is the `json$observations` data frame.

```
el_house_prices ← json$observations  
el_house_prices
```

##	realtime_start	realtime_end	date	value
## 1	2024-02-19	2024-02-19	2016-07-01	.
## 2	2024-02-19	2024-02-19	2016-08-01	.
## 3	2024-02-19	2024-02-19	2016-09-01	.
## 4	2024-02-19	2024-02-19	2016-10-01	.
## 5	2024-02-19	2024-02-19	2016-11-01	.
## 6	2024-02-19	2024-02-19	2016-12-01	.

Using the FRED API

Cleaning up a little bit:

```
el_house_prices <- mutate(el_house_prices,  
  date = ymd(date),  
  median_price = as.numeric(value)  
) %>%  
  select(date, median_price)  
el_house_prices
```

```
##           date median_price  
## 1  2016-07-01           NA  
## 2  2016-08-01           NA  
## 3  2016-09-01           NA  
## 4  2016-10-01           NA  
## 5  2016-11-01           NA  
## 6  2016-12-01           NA  
## 7  2017-01-01           NA  
## 8  2017-02-01           NA  
## 9  2017-03-01           NA  
## 10 2017-04-01           NA
```

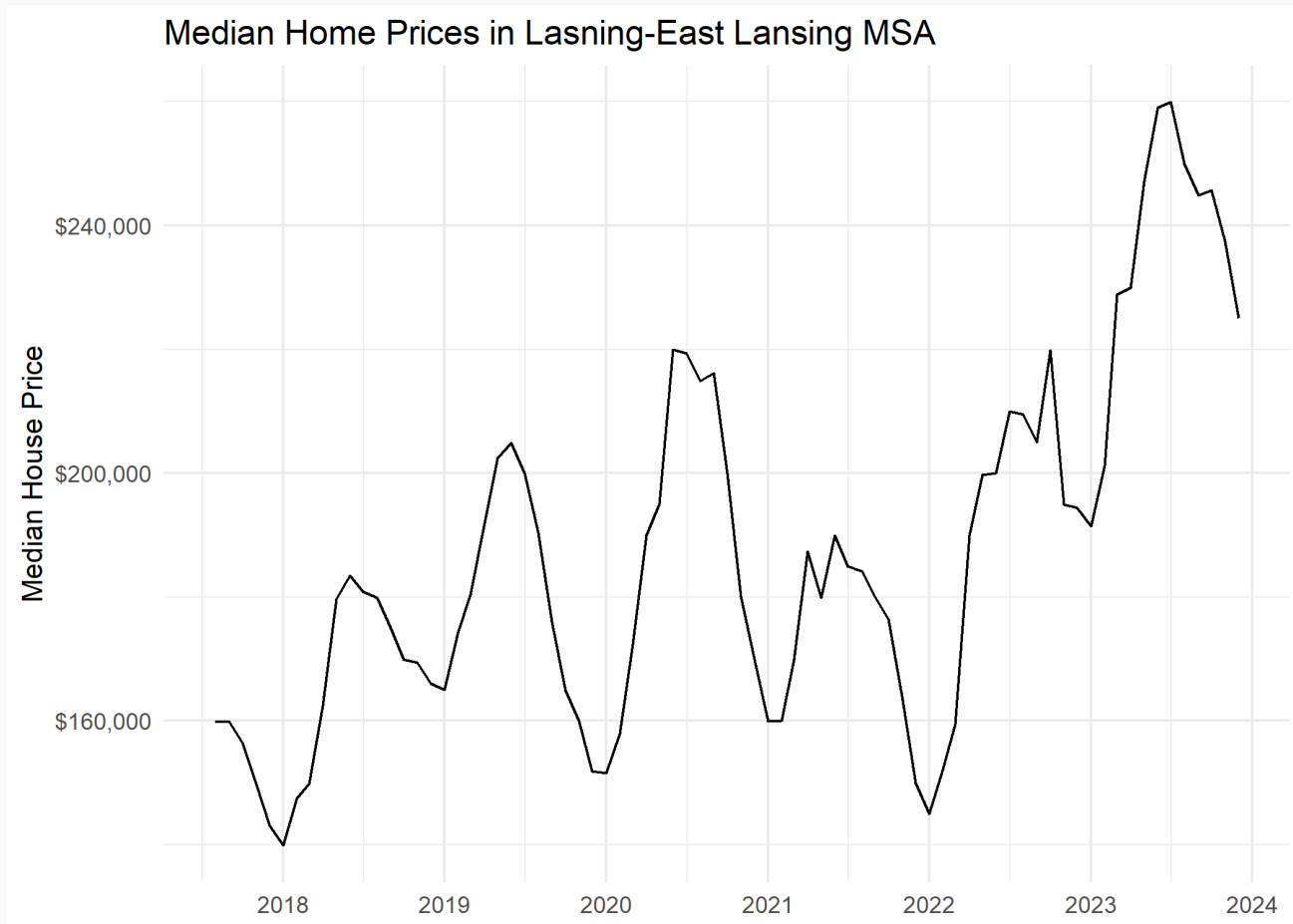

Using the FRED API

Plotting:

```
drop_na(el_house_prices) %>%  
ggplot(aes(x = date, y = median_price)) +  
  geom_line() +  
  theme_minimal() +  
  scale_y_continuous(labels = scales::label_dollar(),  
                      name = "Median House Price") +  
  scale_x_date(name = NULL,  
               date_breaks = "1 year",  
               date_labels = "%Y") +  
  labs(title = "Median Home Prices in Lansing-East Lansing MSA")
```

Using the FRED API

Plotting:



Challenge

How has East Lansing's pandemic housing market compared with other cities?

- Get similar data from 2-3 other housing markets you're interested in.
- Plot time-series graphs for each housing market.
- More advanced: combine all the results in 1 dataframe (keep it tidy!).

Hidden APIs

Hidden APIs

The third kind of APIs are known as **Hidden APIs**.

These are endpoints used for a website's **Internal page generation** and is not intended for public use.

- For example, Airbnb uses or has used a hidden API to display search results on its web interface.
- Grant McDermott has an [example](#) of how to find and use hidden APIs.
- We're not going to go in-depth into hidden API access because...

Hidden APIs

One caveat: the legality of using hidden APIs is **less clear**

- In April 2022 (*HiQ Labs v. LinkedIn*), the Ninth Circuit Court of Appeals held that webscraping is legal in the U.S. so long as the data is **publicly available**
- Do hidden APIs constitute publicly available information?

Even if generally legal, web scraping may be a violation of a website's Terms & Conditions

- Airbnb: "Do not use bots, crawlers, scrapers, or other automated means to access or collect data or other content from or otherwise interact with the Airbnb Platform."
- Your IP could be blocked or your account could be blocked/blacklisted

Considerate Web Scraping

Considerate Web Scraping

In addition to our internal evaluation of

Just because we can, doesn't mean we should

another useful tool in this process is **checking for and reading the robots.txt file**.

Many sites have a `robots.txt` file that outlines what parts of the site web crawlers *can* and *can't* access.

- Specific pages or API endpoints
- Specific types of users

Considerate Web Scraping

The `robotstxt` package makes it easy to check for and read these files. For example, Wikipedia:

```
get_robotstxt(domain = "https://www.wikipedia.org")
```

```
## [robots.txt]
## -----
##
## # robots.txt for http://www.wikipedia.org/ and friends
## #
## # Please note: There are a lot of pages on this site, and there are
## # some misbehaved spiders out there that go _way_ too fast. If you're
## # irresponsible, your access to the site may be blocked.
## #
##
## # Observed spamming large amounts of https://en.wikipedia.org/?curid=NNNNNN
## # and ignoring 429 ratelimit responses, claims to respect robots:
## # http://mj12bot.com/
## User-agent: MJ12bot
## Disallow: /
##
## # advertising-related bots:
## User-agent: Mediapartners-Google*
## Disallow: /
##
## # Wikipedia work bots:
```

Considerate Web Scraping

Best Practices for considerate web scraping:

- Check the `robots.txt` file and scrape what's allowed
- If no `robots.txt` file exists, stick to
 1. Public information
 2. Scraped at a reasonable speed (i.e. human frequency)
- Scrape the data **once**, not **every time you run the script**
- Ask yourself: "would I feel comfortable having an undergrad RA scrape this?"

Table of Contents

1. **Prologue**
2. **APIs**
3. **Explicit API Endpoint, No Key**
4. **Explicit API Endpoint with Key**
5. **Hidden APIs**
6. **Considerate Web Scraping**