General Bikeshare Feed Specification

~or~

an intro to APIs in R

citi bike® citi bike® citi bike® citi bike®

General Bikeshare Feed Specification

~or~

an intro to APIs in R

What are APIs?

- Application Programing Interfaces are tools for getting real time data into projects
- ”real time data” have not been a traditional part of medical statistical analysis

  BUT
- Working with these data streams is super fun with the right tools

What are APIs?

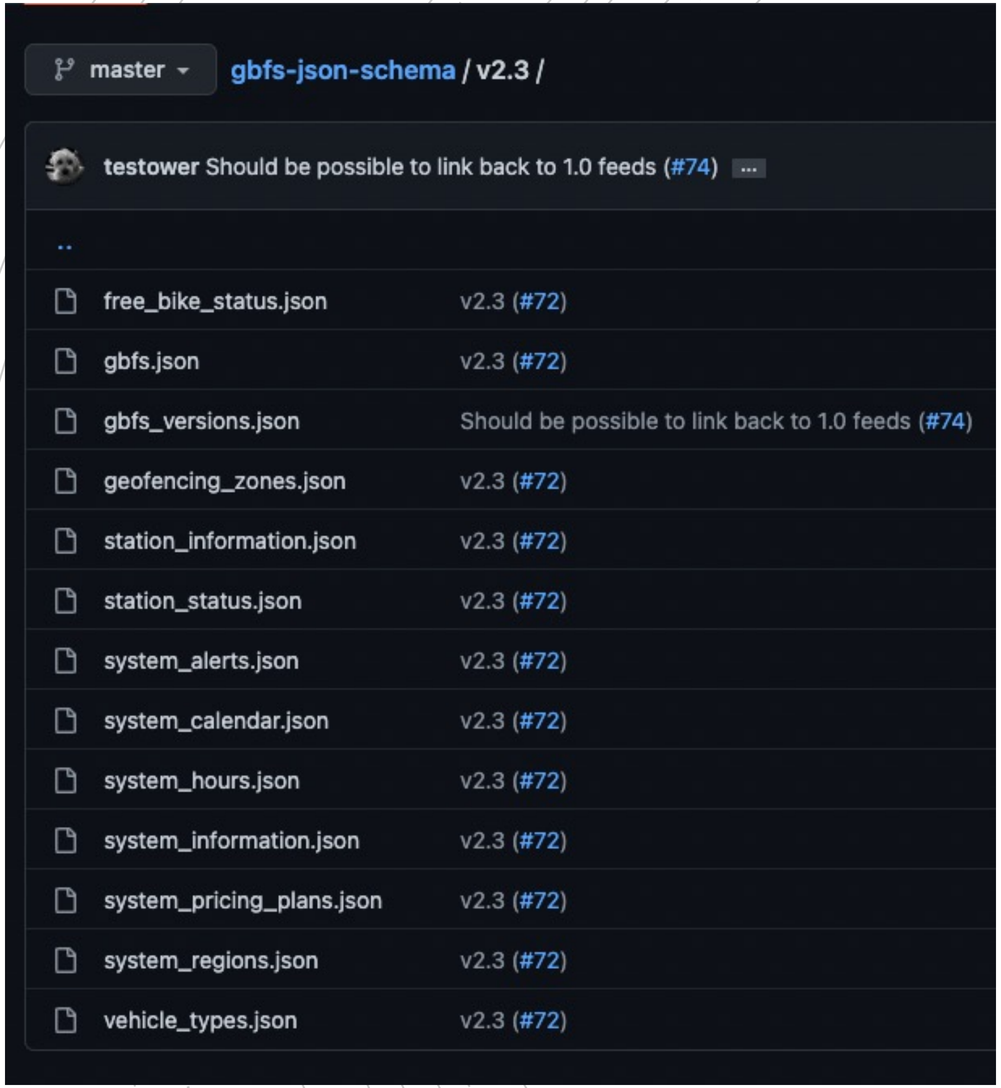
- APIs are tools. Just like physical tools, they can have unique looks, feel, and general quality. Here are the links we need to learn about today's API:
  - https://ride.citibikenyc.com/system-data

    (Data "Owner"/"Provider"/"Source" documentation)
  - https://cran.r-project.org/web/packages/gbfs/gbfs.pdf

    ("Wrapper" of data access methods)

API Principles

- APIs “wrap” data and data access up by standardizing common actions and data structures.
- Your knowledge of traditional (SQL) database principles can help you understand APIs.



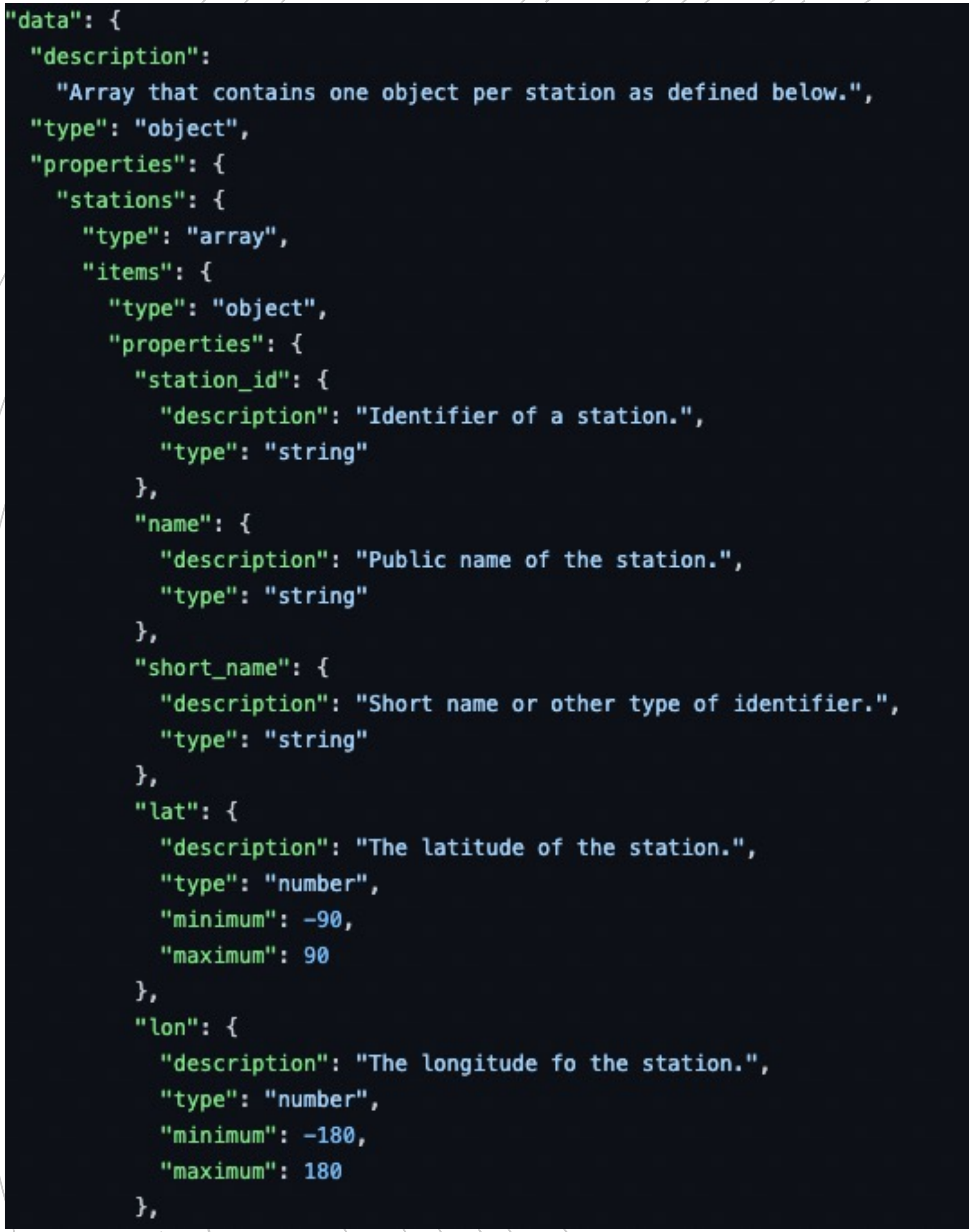
master / gbfs-json-schema / v2.3 /

testower Should be possible to link back to 1.0 feeds (#74) ...

..

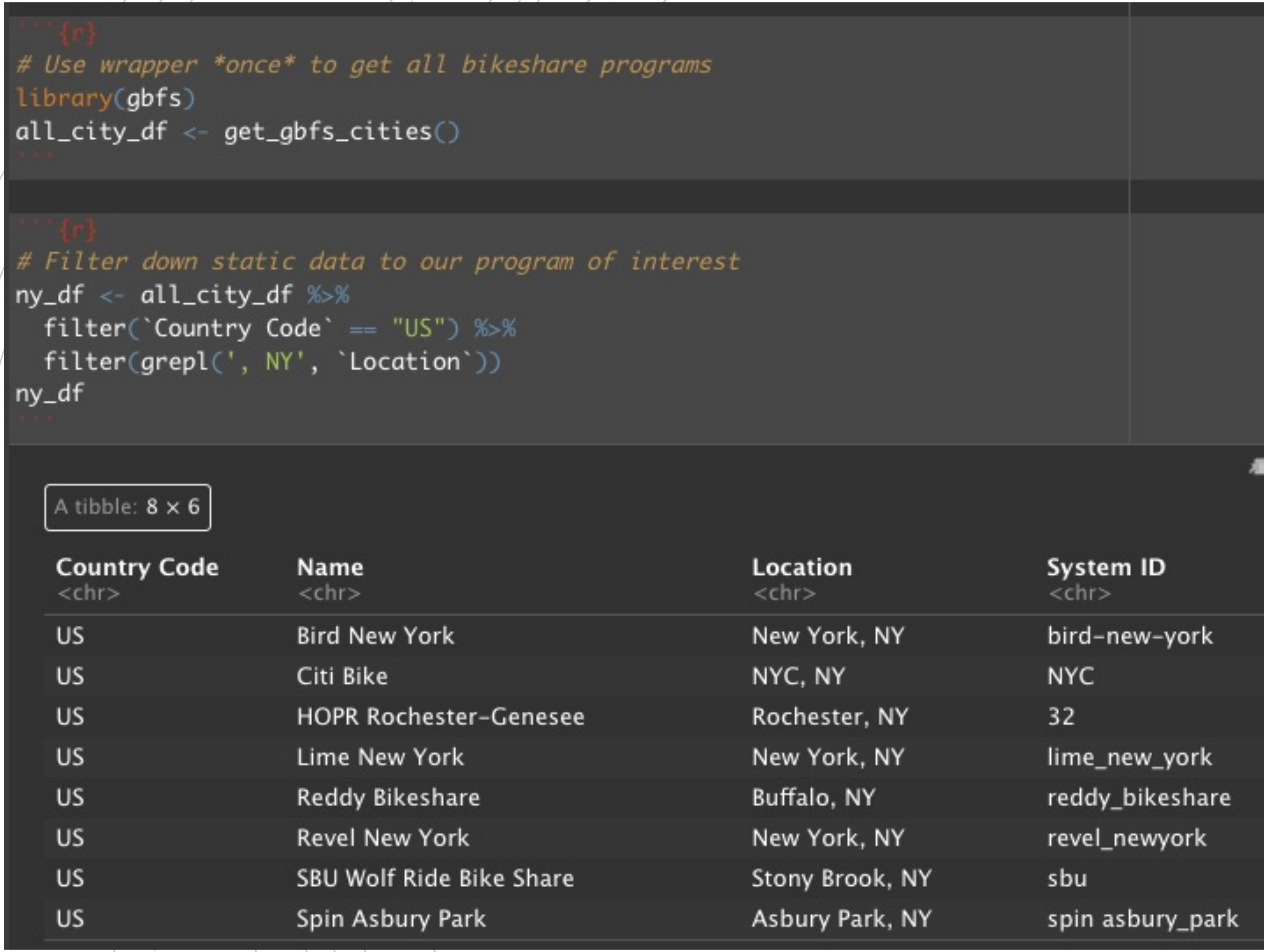
free_bike_status.json	v2.3 (#72)
gbfs.json	v2.3 (#72)
gbfs_versions.json	Should be possible to link back to 1.0 feeds (#74)
geofencing_zones.json	v2.3 (#72)
station_information.json	v2.3 (#72)
station_status.json	v2.3 (#72)
system_alerts.json	v2.3 (#72)
system_calendar.json	v2.3 (#72)
system_hours.json	v2.3 (#72)
system_information.json	v2.3 (#72)
system_pricing_plans.json	v2.3 (#72)
system_regions.json	v2.3 (#72)
vehicle_types.json	v2.3 (#72)

- Common <u>data structures:</u>
- A .json file type spells out the collection of all key:value pairs that you can access for a given data topic (just like SQL schemas!)
  - 13 .jsons describe the entire data space of the "General Bikeshare Feed Specification"
  - Let's look at station_information.json next



```
"data": {
  "description":
    "Array that contains one object per station as defined below.",
  "type": "object",
  "properties": {
    "stations": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "station_id": {
            "description": "Identifier of a station.",
            "type": "string"
          },
          "name": {
            "description": "Public name of the station.",
            "type": "string"
          },
          "short_name": {
            "description": "Short name or other type of identifier.",
            "type": "string"
          },
          "lat": {
            "description": "The latitude of the station.",
            "type": "number",
            "minimum": -90,
            "maximum": 90
          },
          "lon": {
            "description": "The longitude fo the station.",
            "type": "number",
            "minimum": -180,
            "maximum": 180
          },
```

- Common <u>API data structures:</u>
- The .json file type spells out the collection of all key:value pairs that you can access for a given data topic
  - The attributes of station_information.json have a nested structure. For example:
    - ”lat” and “lon” both have descriptions, data types, and max and min values.
    - ”lat” and “lon” themselves are both items in a collection of data objects that each physical station has associated with it



~~~
```{r}
# Use wrapper *once* to get all bikeshare programs
library(gbfs)
all_city_df <- get_gbfs_cities()
```
~~~

~~~
```{r}
# Filter down static data to our program of interest
ny_df <- all_city_df %>%
  filter(`Country Code` == "US") %>%
  filter(grepl(', NY', `Location`))
ny_df
```
~~~

A tibble: 8 × 6

Country Code <chr>	Name <chr>	Location <chr>	System ID <chr>
US	Bird New York	New York, NY	bird-new-york
US	Citi Bike	NYC, NY	NYC
US	HOPR Rochester-Genesee	Rochester, NY	32
US	Lime New York	New York, NY	lime_new_york
US	Reddy Bikeshare	Buffalo, NY	reddy_bikeshare
US	Revel New York	New York, NY	revel_newyork
US	SBU Wolf Ride Bike Share	Stony Brook, NY	sbu
US	Spin Asbury Park	Asbury Park, NY	spin asbury_park

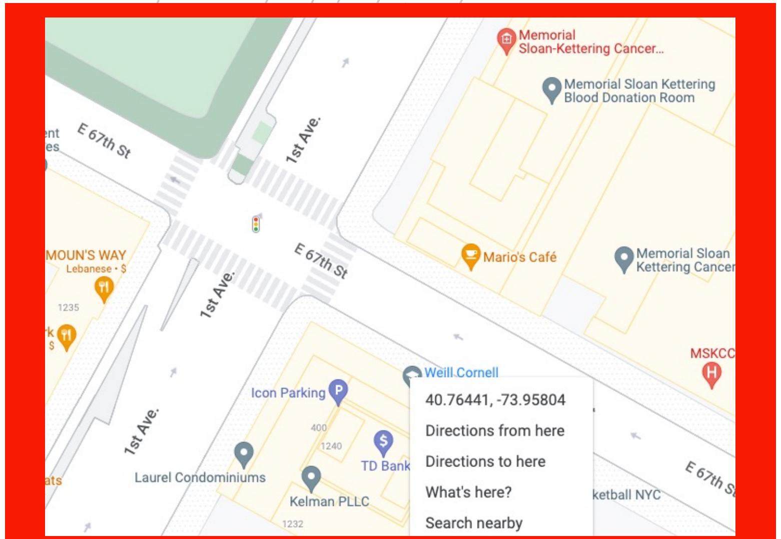
- Common actions:
  - The CRAN package `gbfs` wraps accessing data so that it is simple
  - While we've learned API work by defining and passing around .json files, you can get tidy dataframes directly by using the package
- Looks like `NYC` is our system id of interest

Let’s Review

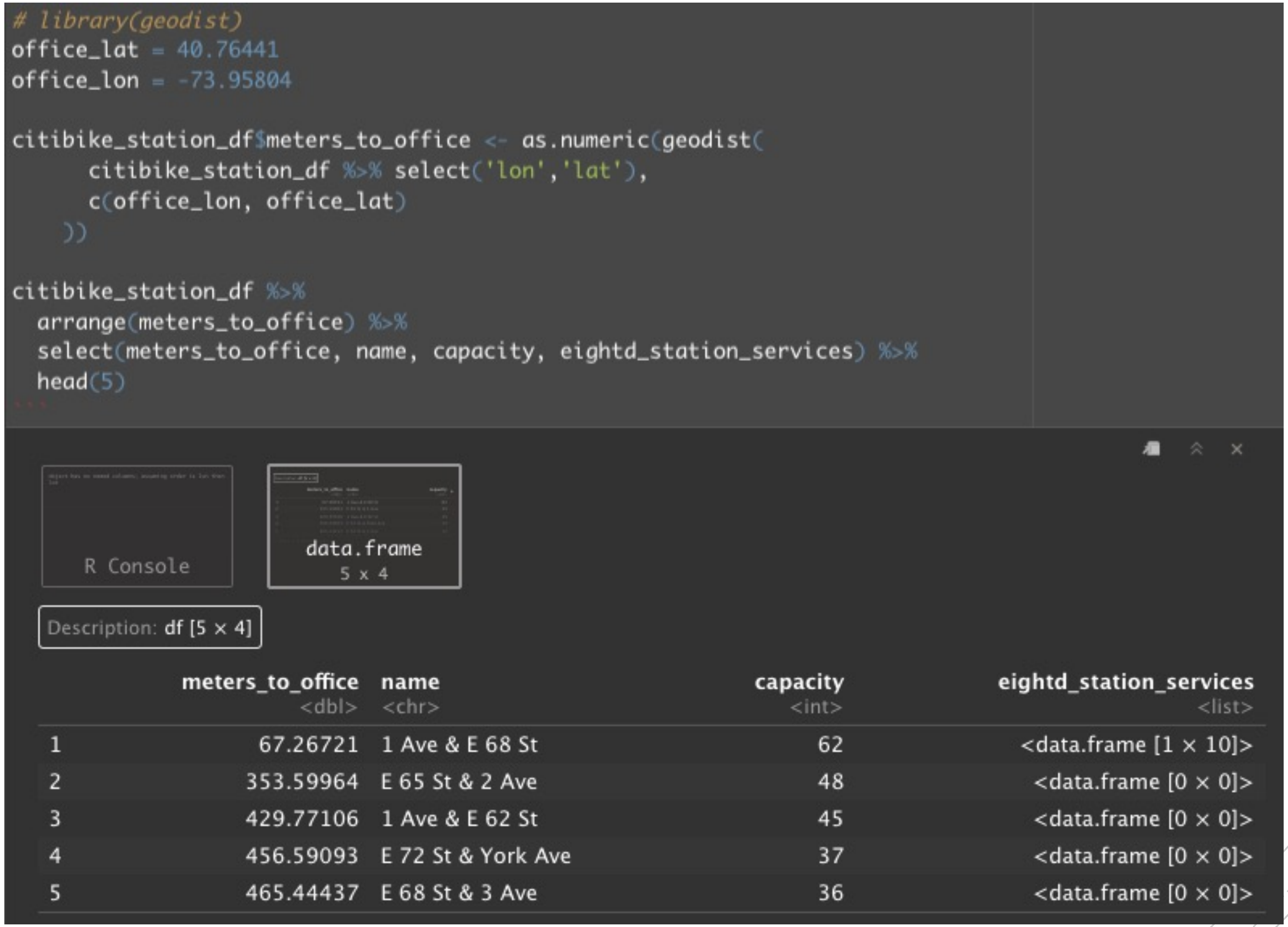
- We got our first taste of the API *CRAN::GBFS*
- We used it to pull “fresh” data directly into a tidy-data-frame

Using R with API data

Finding Home



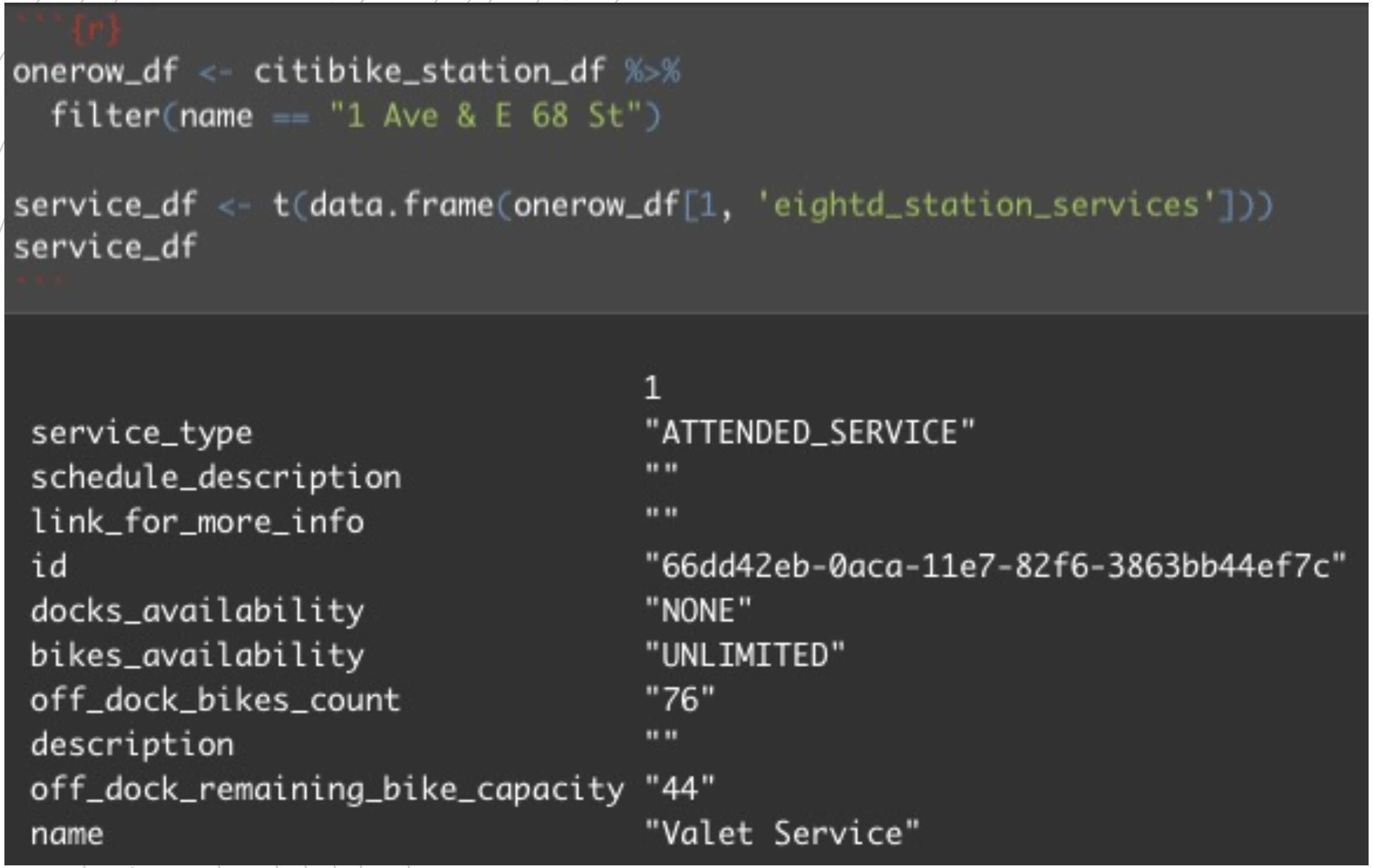
Knowing the office coordinates, finding the closest station is easy



```r
# library(geodist)
office_lat = 40.76441
office_lon = -73.95804

citibike_station_df$meters_to_office <- as.numeric(geodist(
      citibike_station_df %>% select('lon','lat'),
      c(office_lon, office_lat)
    ))

citibike_station_df %>%
  arrange(meters_to_office) %>%
  select(meters_to_office, name, capacity, eightd_station_services) %>%
  head(5)
```

```
Description: df [5 × 4]

  meters_to_office name                capacity eightd_station_services
             <dbl> <chr>                  <int>                  <list>
1         67.26721 1 Ave & E 68 St           62   <data.frame [1 × 10]>
2        353.59964 E 65 St & 2 Ave           48    <data.frame [0 × 0]>
3        429.77106 1 Ave & E 62 St           45    <data.frame [0 × 0]>
4        456.59093 E 72 St & York Ave        37    <data.frame [0 × 0]>
5        465.44437 E 68 St & 3 Ave           36    <data.frame [0 × 0]>
```

Using R with API data cont.



~~~
```{r}
onerow_df <- citibike_station_df %>%
  filter(name == "1 Ave & E 68 St")

service_df <- t(data.frame(onerow_df[1, 'eightd_station_services']))
service_df
```
~~~

```
                                 1
service_type                     "ATTENDED_SERVICE"
schedule_description             ""
link_for_more_info               ""
id                               "66dd42eb-0aca-11e7-82f6-3863bb44ef7c"
docks_availability               "NONE"
bikes_availability               "UNLIMITED"
off_dock_bikes_count             "76"
description                      ""
off_dock_remaining_bike_capacity "44"
name                             "Valet Service"
```

- Looks like this station is special!
- Even though the official capacity of the station is only 62, there seems to be… 44, 76, or UNLIMITED extra bikes available via valet

Next Steps

- *CRAN::GBFS* is a “best case” kind of API, that abstracts over a dozen useful data operations you might need with citibike-like data.
- The next slides will example the “average case” API

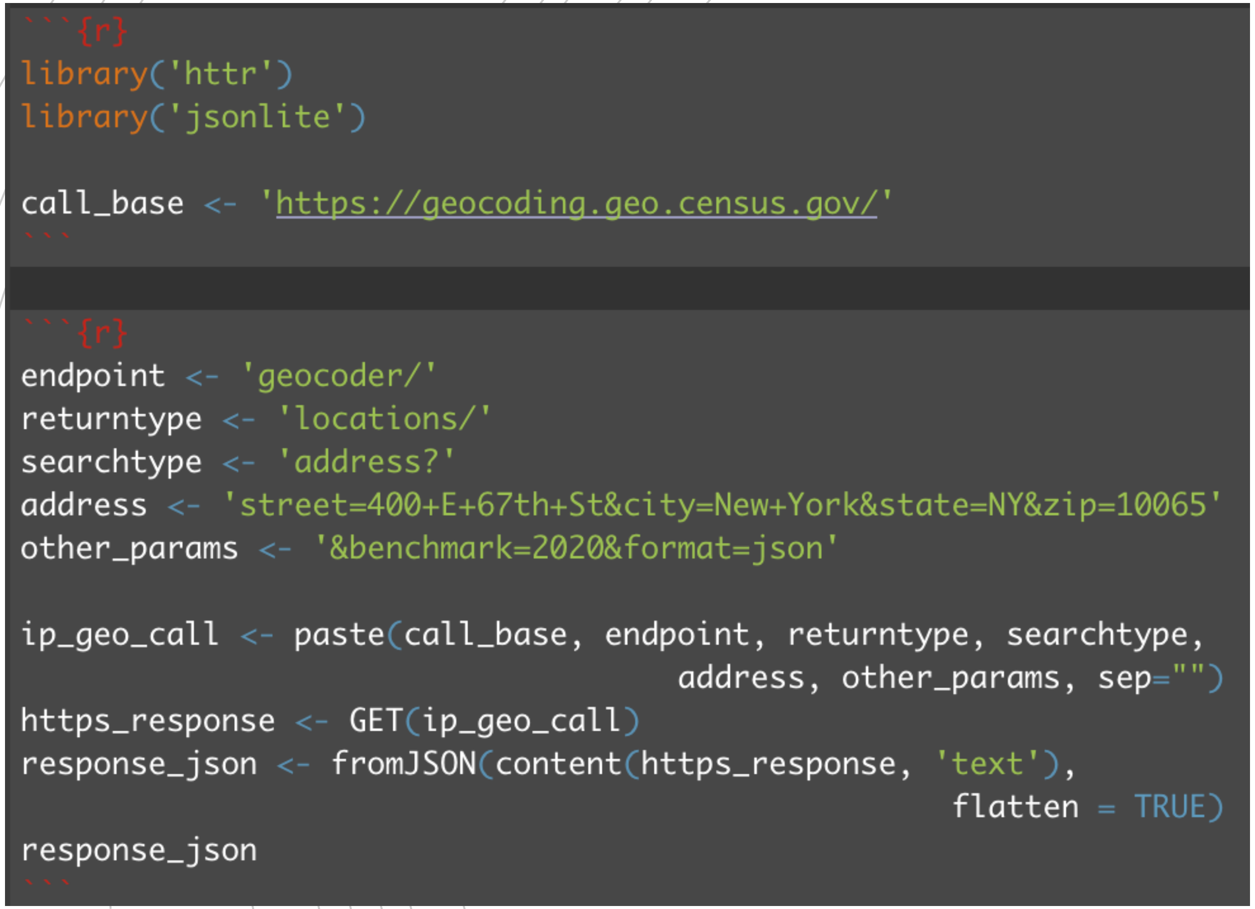
Using R with API data cont.

Finding Home…

…without google maps

- Let’s go through a common API task: *geocoding*, or (Address) -> (Lat, Lon)
- The US Census provides us the API
  - *https://www.census.gov/programs-surveys/geography/technical-documentation/complete-technical-documentation/census-geocoder.html*

Using R with API data cont.



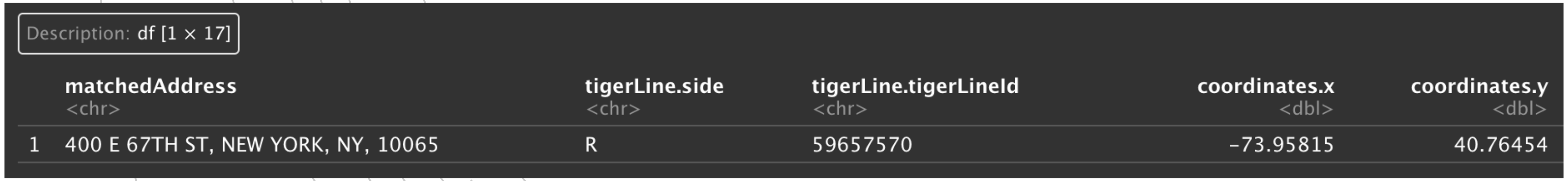
~~~
```{r}
library('httr')
library('jsonlite')

call_base <- 'https://geocoding.geo.census.gov/'
```
~~~

~~~
```{r}
endpoint <- 'geocoder/'
returntype <- 'locations/'
searchtype <- 'address?'
address <- 'street=400+E+67th+St&city=New+York&state=NY&zip=10065'
other_params <- '&benchmark=2020&format=json'

ip_geo_call <- paste(call_base, endpoint, returntype, searchtype,
                       address, other_params, sep="")
https_response <- GET(ip_geo_call)
response_json <- fromJSON(content(https_response, 'text'),
                            flatten = TRUE)
response_json
```
~~~

- Using R-httr & R-jsonlite we perform the more basic *request*

  GET(‘url’)

- After some un-wrapping we get the *response* as a df and json



Description: df [1 × 17]

	matchedAddress <chr>	tigerLine.side <chr>	tigerLine.tigerLineId <chr>	coordinates.x <dbl>	coordinates.y <dbl>
1	400 E 67TH ST, NEW YORK, NY, 10065	R	59657570	-73.95815	40.76454

Using R with API data cont.

Finding Home…

…without google maps

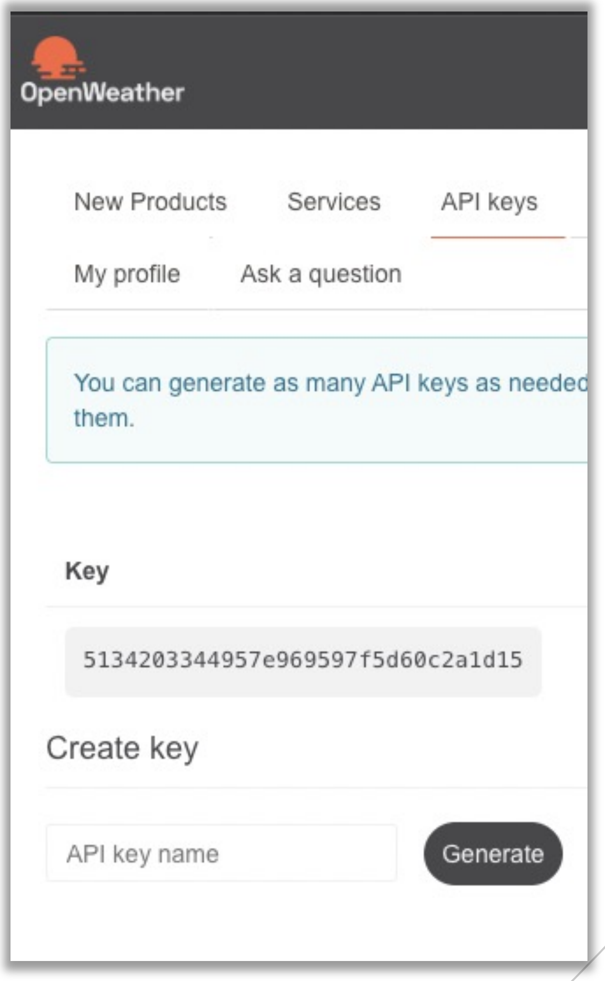
- Using API(s) we've been able to locate the closest station to the building, and the services offered at that station using R *alone*

Final Words:

Level 3 APIs

Many API require signing up for an *API Key:*

- These prevent spam and often involve giving your email to the site hosting documentation and waiting up to a few hours for the key to start working!



You can generate as many API keys as needed
them.

Key

5134203344957e969597f5d60c2a1d15

API key name

Thank You!

Colby Lewis V (he/him)
Research Biostatistician
Weill Cornell Medicine | NewYork-Presbyterian
MS Biostatistics | Columbia Mailman SPH
Department of Population Health Sciences
402 East 67th Street, New York, NY 10065
jcl7005@med.cornell.edu