SF Data Tips and Tricks

OEWD Data and Performance Team

2024-04-03

Table of contents

Introduction

This WIP Quarto 'book' is a collection of (mostly R) tips and tricks relevant to data professionals with the City and County of San Francisco.

Comments, questions, or suggestions? Create an issue here.

Edits, additions, or corrections? Open a pull request.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 DataSF

Getting data from DataSF is a matter of copying the relevant URL into one of R's many read functions, e.g. `readr::read_csv`, `jsonlite::fromJSON`, `st::st_read`, etc.

```r
library(readr)
library(sf)
```

```
Linking to GEOS 3.9.1, GDAL 3.4.3, PROJ 7.2.1; sf_use_s2() is TRUE
```

```r
reg_businesses <- read_csv("https://data.sfgov.org/resource/g8m3-pdis.csv")
```

```
Rows: 1000 Columns: 37
```

```
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (22): uniqueid, ttxid, certificate_number, ownership_name, dba_name, fu...
dbl   (7): business_zip, supervisor_district, :@computed_region_6qbp_sg9q, :...
lgl   (2): parking_tax, transient_occupancy_tax
dttm  (6): dba_start_date, dba_end_date, location_start_date, location_end_d...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

⚠ Warning

Behind the scenes there is a limit parameter that defaults to 1000, *even if the 'All data' radio button is selected.* To retrieve all the data, either read the same URL with the RSocrata package:

```r
reg_businesses <- RSocrata::read.socrata("https://data.sfgov.org/resource/g8m3-pdis.csv")
```

Or append `?$limit=9999999` to the end of the URL:

```
reg_businesses <- read_csv("https://data.sfgov.org/resource/g8m3-pdis.csv?$limit=9999999")
```

Read in a ‘spatial’ object with `st_read` and the URL with the geojson file extension:

```
sup_dists <- st_read("https://data.sfgov.org/api/geospatial/f2zs-jevy?accessType=DOWNLOAD&me
```

```
Reading layer `OGRGeoJSON' from data source
  `https://data.sfgov.org/api/geospatial/f2zs-jevy?accessType=DOWNLOAD&method=export&format=
  using driver `GeoJSON'
Simple feature collection with 11 features and 7 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -123.1738 ymin: 37.63983 xmax: -122.3279 ymax: 37.8632
Geodetic CRS:  WGS 84
```

2 Excel Stuff

{readxl} is the best package to *read* excel (xlsx) files.

```r
library(readxl)
df <- read_xlsx("path_to_xlsx")
```

{writexl} is the best package to *write* excel (xlsx) files.

```r
library(writexl)
write_xlsx(mtcars, "car_data.xlsx")
```

2.1 Miscellaneous Tips and Tricks

You can also pass a named list of data frames to `write_xlsx`, and it will write each data frame to a separate sheet in the workbook.

```r
l <- list(
  "Car data" = mtcars,
  "Flower data" = iris
)

write_xlsx(l, "my_data.xlsx")
```

With the `{purrr}` package you can look through a directory of identically structured spreadsheets and bind them together:

```r
library(purrr)
dir("path_to_directory", full.names = TRUE) %>%
  map_dfr(\(file) read_csv(file)) # or map(\(file) read_csv(file)) %>% list_rbind()
```

With the `{kapow}` package package you can loop through each sheet in an xlsx workbook and assign the table to its sheet name in your global environment:

```
library(readxl)
library(purrr)
library(kapow) # remotes::install_github("daranzolin/kapow)

xlsx_path <- "path_to_xlsx"
sheet_names <- excel_sheets(xlsx_path)
sheet_names %>%
  map(\(sheet) read_excel(xlsx_path, sheet = sheet)) %>%
  set_names(sheet_names) %>%
  kapow()
```

To apply special formatting to an Excel workbook in R you can use the {openxlsx} package. Here's an example of how OEWD writes to xlsx:

```
library(openxlsx)
write_oewd_xlsx <- function(data, sheet_name, file, dateFormat = "yyyy/mm/dd", overwrite = FA
  wb <- createWorkbook()
  addWorksheet(wb, sheet_name)
  style <- createStyle(halign = "LEFT", valign = "CENTER")
  setColWidths(wb, sheet = 1, cols = 1:ncol(data), widths = "auto")
  addStyle(wb, 1,
           cols = 1:(ncol(data) + 1),
           rows = 1:(nrow(data) + 1),
           style = style,
           gridExpand = TRUE)
  headerStyle <- createStyle(
    halign = "LEFT",
    textDecoration = "Bold"
  )
  writeData(wb, 1, data, headerStyle = headerStyle)
  options("openxlsx.dateFormat" = dateFormat)
  saveWorkbook(wb, file, overwrite = overwrite)
}

write_oewd_xlsx(mtcars, "Car data", "car_data.xlsx")
```

3 Sharepoint

Sharepoint is part of many data pipelines, and the easiest way to interact with Sharepoint from R is through the {Microsoft365R} package. This package has been vetted by the Department of Technology (DT) and authentication should be straight-forward after calling one of the initial functions.

```r
library(Microsoft365R)

wp <- get_sharepoint_site("Workforce Programs")
wp_docs <- wp$get_drive("Documents")

wp_docs$list_files()

wp_docs$download_file("path_to_file.xlsx")
```

It is helpful to wrap common read/write operations into more usable functions:

```r
connect_to_sharepoint_site_docs <- function(sp_site) {
  sp_site <- Microsoft365R::get_sharepoint_site(sp_site)
  sp_site_doc <- sp_site$get_drive("Documents")
  return(sp_site_doc)
}

connect_to_wp_docs <- function() connect_to_sharepoint_site_docs("Workforce Programs")

download_from_wp <- function(sp_file, destination) {
  docs <- connect_to_wp_docs()
  docs$download_file(sp_file, dest = destination)
  cli::cli_alert_success(glue::glue("{sp_file} downloaded."))
}

upload_to_wp <- function(file, sp_destination) {
  docs <- connect_to_wp_docs()
  docs$upload_file(
    file,
    sp_location
  )
  cli::cli_alert_success(glue::glue("{file} uploaded."))
}

read_wp <- function(path) {
  tmp <- tempfile(fileext = "xlsx")
  download_from_wp(
    sp_file = glue("{path}.xlsx"),
    destination = tmp
  )
  out <- readxl::read_excel(tmp, .name_repair = janitor::make_clean_names)
  return(out)
}
```

4 Spatial Stuff

The most essential packages for doing GIS work (with ESRI products) are:

- sf
- mapview
- arcgis
- arcgisutils
- arcgislayers
- arcgisbinding
- tidycensus

Honorable mention:

- tmap
- terra
- leaflet

```r
library(sf)
```

```
Linking to GEOS 3.9.1, GDAL 3.4.3, PROJ 7.2.1; sf_use_s2() is TRUE
```

```r
library(mapview)
library(arcgis)
```

```
Attaching core arcgis packages:
```

```
> arcgisutils v0.1.1.9000
> arcgislayers v0.1.0
```

```r
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.5.1     v purrr   1.0.2
v tibble  3.2.1     v dplyr   1.1.0
v tidyr   1.2.1     v stringr 1.5.0
v readr   2.1.3     v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x purrr::%||%()    masks arcgisutils::%||%()
x purrr::compact() masks arcgisutils::compact()
x dplyr::filter()  masks stats::filter()
x dplyr::lag()     masks stats::lag()
```

4.1 ArcGIS REST API

The {arcgislayers} package allows users to read and write data from and to the ArcGIS REST API.

4.1.1 Reading Layers

```r
sf_libraries_url <- "https://services.arcgis.com/Zs2aNLFN00jrS4gG/arcgis/rest/services/SF_Li
# arc_open can read a FeatureServer or a FeatureLayer directly
(sf_libraries_fs <- arc_open(sf_libraries_url))
```

```
<FeatureServer <2 layers, 0 tables>>
CRS: 3857
Capabilities: Query
  0: Libraries (esriGeometryPoint)
  1: Libraries with Air Conditioning (esriGeometryPoint)
```

```r
(sf_libraries_lyr <- get_layer(sf_libraries_fs, name = "Libraries"))
```

```
<FeatureLayer>
Name: Libraries
Geometry Type: esriGeometryPoint
CRS: 3857
Capabilities: Query
```

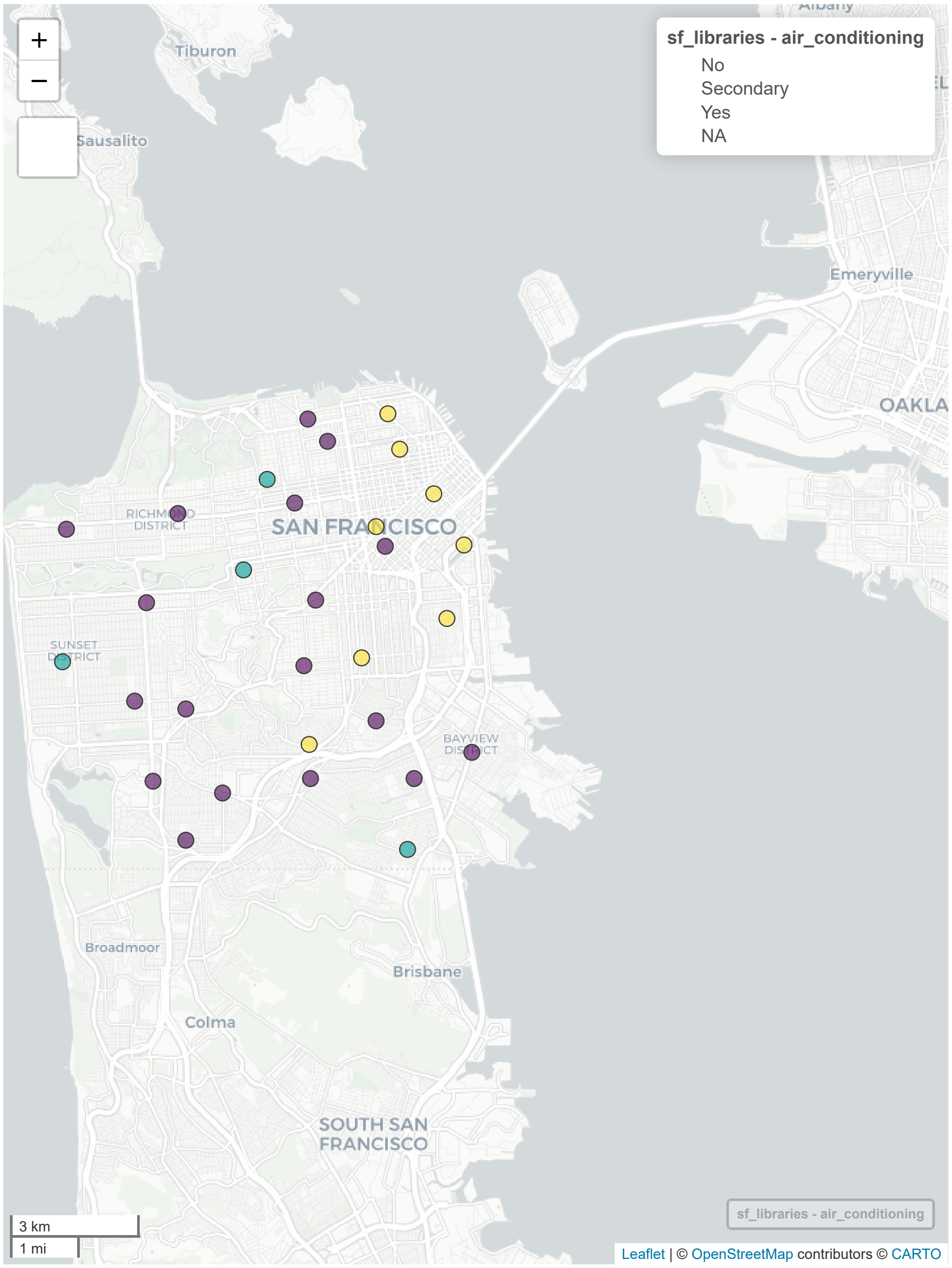
```r
sf_libraries <- arc_select(sf_libraries_lyr)
```

```
Registered S3 method overwritten by 'jsonify':
  method     from    
  print.json jsonlite
```

```r
glimpse(sf_libraries)
```

```
Rows: 33
Columns: 23
$ objectid         <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16~
$ gross_sq_f       <chr> "8500", "6465", "6096", "8536", "7633", "6100", "8000~
$ block_lot        <chr> "8708110", "3564095", "6539034", "2919031", "0469001"~
$ zip_code         <chr> "94158", "94114", "94114", "94127", "94123", "94112",~
$ facility_i       <chr> "1113", "648", "1184", "1853", "1053", "896", "1858",~
$ city             <chr> "San Francisco", "San Francisco", "San Francisco", "S~
$ latitude         <chr> "37.775369728", "37.76406037", "37.750228042", "37.74~
$ department       <chr> "Public Library", "Public Library", "Public Library",~
$ longitude        <chr> "-122.393097384", "-122.431881717", "-122.435090242",~
$ dept_id          <chr> "48", "48", "48", "48", "48", "48", "48", "48", "48",~
$ common_nam       <chr> "Mission Bay Library", "Eureka Valley Branch Library/~
$ address          <chr> "960 04th St", "1 Jose Sarria Ct", "451 Jersey St", "~
$ supervisor       <chr> "6", "8", "8", "7", "2", "7", "5", "6", "10", "8", "9~
$ city_tenan       <chr> " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "~
$ owned_leas       <chr> "Own", "Own", "Own", "Own", "Own", "Own", "Own", "Own~
$ globalid         <chr> "755632fc-18d9-4c0e-b65d-ec53d18a132b", "195a2a2b-302~
$ created_user     <chr> "nancy.milholland_sfdem", "nancy.milholland_sfdem", "~
$ created_date     <dttm> 2019-06-07 21:44:34, 2019-06-07 21:44:34, 2019-06-07~
$ last_edited_user <chr> "nancy.milholland_sfdem", "nancy.milholland_sfdem", "~
$ last_edited_date <dttm> 2019-06-07 21:44:34, 2019-06-07 21:44:34, 2019-06-07~
$ eas_id           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ air_conditioning <chr> "Yes", "No", "No", "No", "No", "No", "No", "No", "Sec~
$ geometry         <POINT [m]> POINT (-13624737 4547742), POINT (-13629055 454~
```

```r
# You can also specify which columns to select, e.g.:
# arc_select(
#   sf_libraries_lyr,
#   fields = c("common_nam", "gross_sq_f", "address"),
#   where = "gross_sq_f < 8000"
# )
# With pipes:
# sf_libraries_url %>%
#   arc_open() %>%
#   get_layer(name = "Libraries") %>%
#   arc_select()

# With pipes and tidyverse:
# (if the url points to a FeatureLayer instead of a FeatureServer)
# sf_libraries_url %>%
#   arc_open() %>%
#   select(common_nam, gross_sq_f, address) %>%
#   filter(gross_sq_f < 8000) %>%
#   collect()
mapview(sf_libraries, zcol = "air_conditioning")
```


+
−
sf_libraries - air_conditioning
No
Secondary
Yes
NA
Tiburon
Sausalito
Emeryville
OAKLA
RICHMOND DISTRICT
SAN FRANCISCO
SUNSET DISTRICT
BAYVIEW DISTRICT
Broadmoor
Brisbane
Colma
SOUTH SAN FRANCISCO
sf_libraries - air_conditioning
3 km
1 mi
Leaflet | © OpenStreetMap contributors © CARTO

It is also convenient to wrap this up into the body of a single function:

```r
get_arcgis_layer <- function(lyr_name) {
  url <- glue::glue("https://services.arcgis.com/Zs2aNLFN00jrS4gG/arcgis/rest/services/{lyr_
  out <- arcgislayers::arc_select(arcgislayers::arc_open(url))
  return(out)
}

libraries <- get_arcgis_layer("SF_Libraries")
```

4.1.2 Writing Layers

If you have an sfgov.maps.arcgis.com account, you can write layers directly to your content. Read the authorization page for more information on credentials and tokens.

```r
nc <- st_read(system.file("shape/nc.shp", package = "sf"))
tkn <- auth_code()
set_arc_token(tkn)

publish_res <- publish_layer(nc, "North Carolina SIDS sample")
```

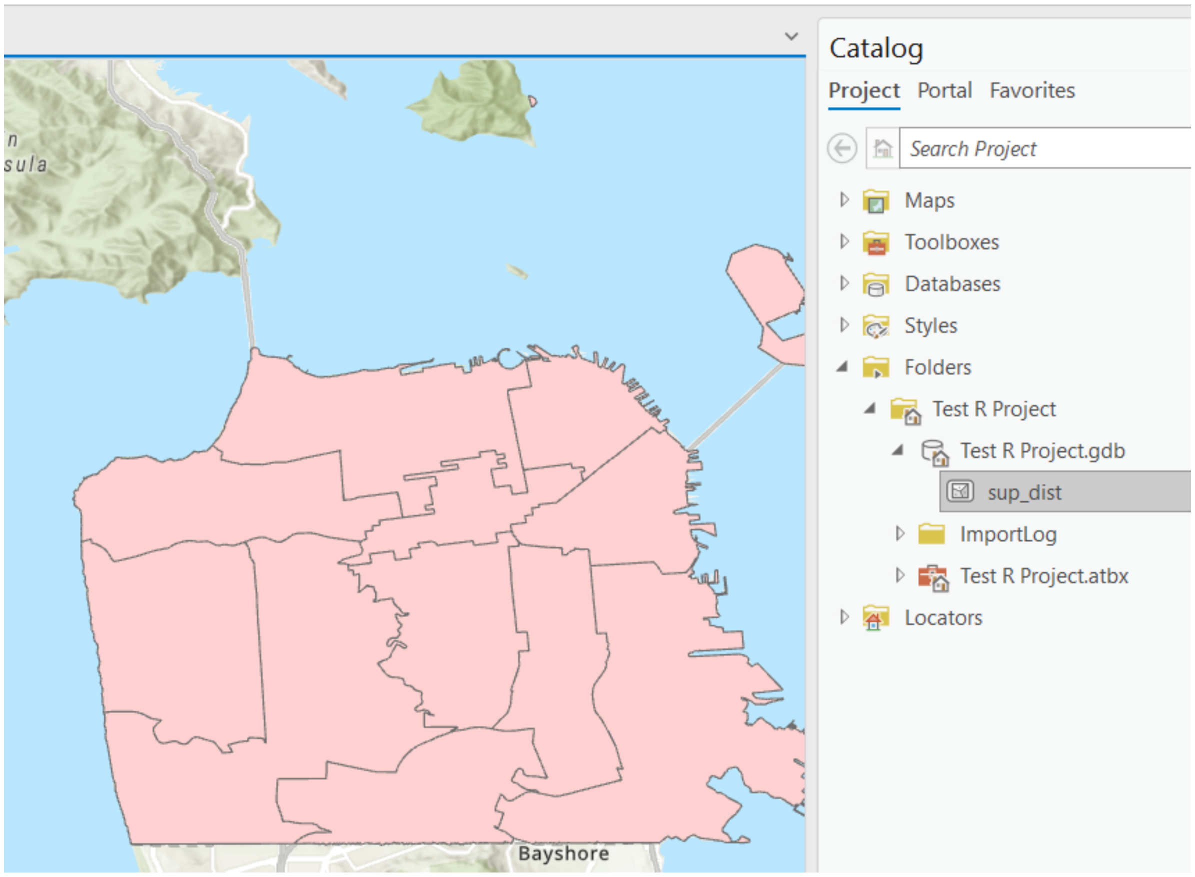
4.2 ArcGIS Pro

If you have an ArcGIS Pro license, you can write directly to geodatabases within Projects using the {arcgisbinding} package.

```
Reading layer `OGRGeoJSON' from data source
  `https://data.sfgov.org/api/geospatial/f2zs-jevy?accessType=DOWNLOAD&method=export&format=
  using driver `GeoJSON'
Simple feature collection with 11 features and 7 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -123.1738 ymin: 37.63983 xmax: -122.3279 ymax: 37.8632
Geodetic CRS:  WGS 84
```

```r
library(arcgisbinding)
# arc.check_product()

# Get Supervisor Districts from DataSF:
```

```
sup_dists <- st_read("https://data.sfgov.org/api/geospatial/f2zs-jevy?accessType=DOWNLOAD&me

# Write to ArcGIS Pro project geodatabase
proj_path <- "...<full_path>.../ArcGIS/Projects/Test R Project/Test R Project.gdb/sup_dist"
arc.write(path = proj_path, data = sup_dists)
```



4.3 Spatial Joins

Use spatial joins to determine which points are 'within' which polygon:

```
nhoods <- st_read("https://data.sfgov.org/resource/j2bu-swwd.geojson")
```

```
Reading layer `j2bu-swwd' from data source
  `https://data.sfgov.org/resource/j2bu-swwd.geojson' using driver `GeoJSON'
```

```
Simple feature collection with 41 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -122.5149 ymin: 37.70813 xmax: -122.357 ymax: 37.8333
Geodetic CRS:  WGS 84
```

```
# The coordinate reference systems must match
st_crs(sf_libraries) == st_crs(sup_dists)
```

```
[1] FALSE
```

```
sf_libraries %>%
  select(common_nam) %>%
  st_transform(st_crs(sup_dists)) %>%
  st_join(sup_dists %>% select(sup_dist), join = st_within) %>%
  st_join(nhoods, join = st_within) %>%
  st_drop_geometry() %>%
  mutate(common_nam = gsub(" Branch| Library", "", common_nam)) %>%
  head()
```

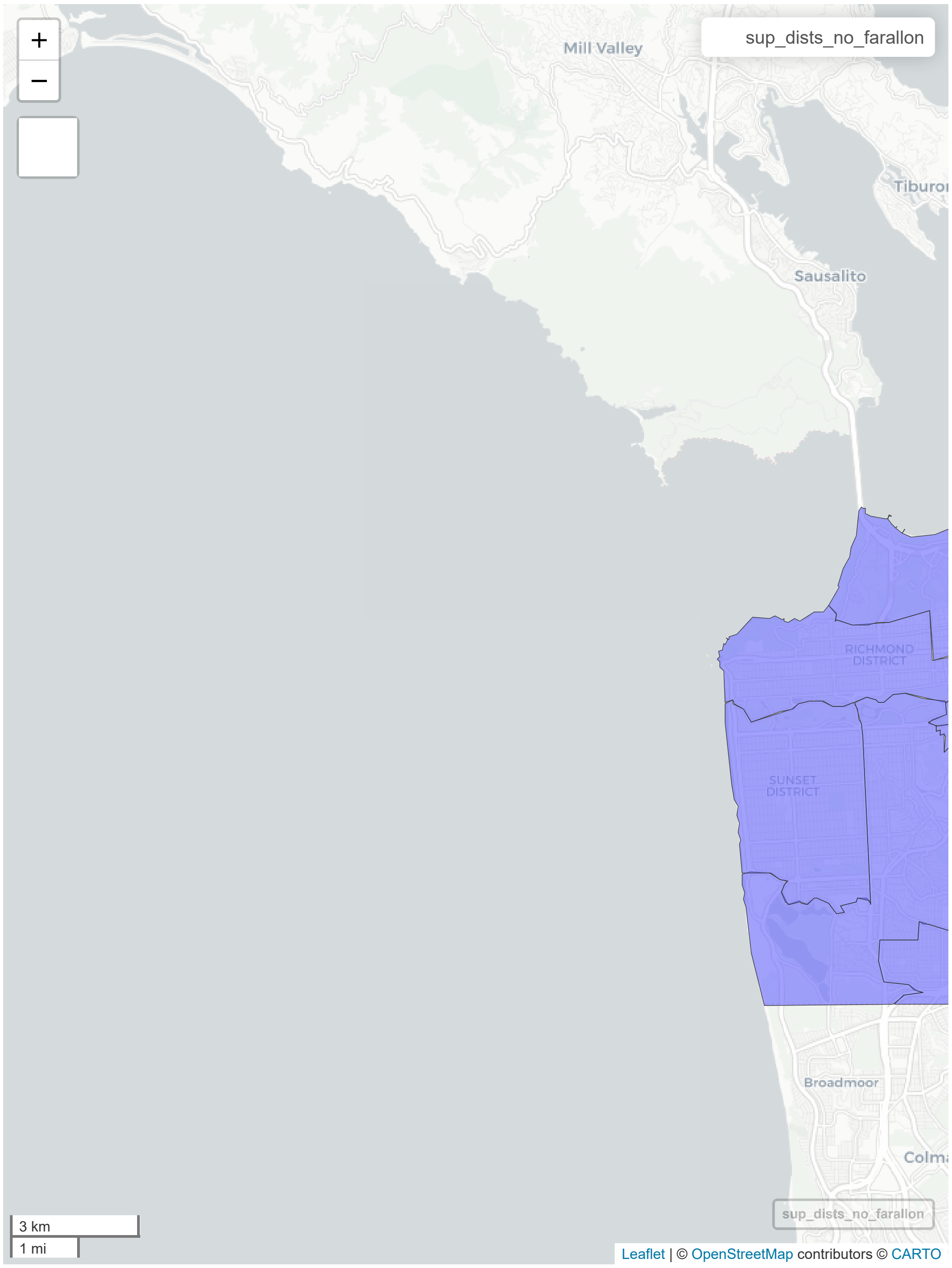
```
                           common_nam sup_dist               nhood
1                         Mission Bay        6         Mission Bay
2 Eureka Valley/ Harvey Milk Memorial        8 Castro/Upper Market
3                          Noe Valley        8          Noe Valley
4                         West Portal        7  West of Twin Peaks
5                              Marina        2              Marina
6                           Ingleside        7  West of Twin Peaks
```

4.4 Removing Farallon Islands from Supervisor Districts

```
d4 <- sup_dists %>%
  filter(sup_dist == 4) %>%
  st_cast("POLYGON") %>%
  slice(1) %>%
  st_cast("MULTIPOLYGON")
```

```
Warning in st_cast.sf(., "POLYGON"): repeating attributes for all
sub-geometries for which they may not be constant
```

```r
sup_dists_no_farallon <- sup_dists %>%
  filter(sup_dist != 4) %>%
  bind_rows(d4)

mapview(sup_dists_no_farallon)
```


+
−
sup_dists_no_farallon
Mill Valley
Tiburon
Sausalito
RICHMOND DISTRICT
SUNSET DISTRICT
Broadmoor
Colma
sup_dists_no_farallon
3 km
1 mi
Leaflet | © OpenStreetMap contributors © CARTO

4.5 Census Data

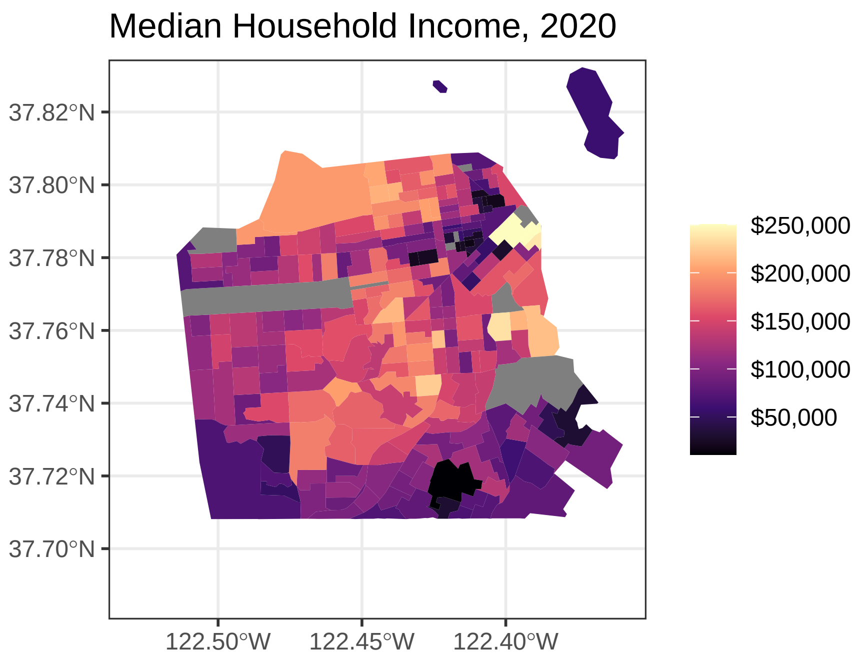
The {tidycensus package} is fantastic, and the documentation is full of helpful examples.

```
library(tidycensus)

sf <- get_acs(
  state = "CA",
  county = "San Francisco",
  geography = "tract",
  variables = "B19013_001",
  geometry = TRUE,
  year = 2020
) %>%
  st_transform(3857)
```

```
Getting data from the 2016-2020 5-year ACS
```

```
Downloading feature geometry from the Census website.  To cache shapefiles for use in future
```

```
sf_bbox <- libraries %>%
  drop_na(city) %>%
  st_buffer(3500) %>%
  st_bbox()

sf %>%
  ggplot(aes(fill = estimate)) +
  geom_sf(color = NA) +
  labs(title = "Median Household Income, 2020", fill = NULL) +
  coord_sf(xlim = sf_bbox[c("xmin", "xmax")], ylim = sf_bbox[c("ymin", "ymax")], expand = TRU
  scale_fill_viridis_c(option = "magma", labels = scales::dollar) +
  theme_bw()
```



Median Household Income, 2020
37.82°N
37.80°N
37.78°N
37.76°N
37.74°N
37.72°N
37.70°N
122.50°W
122.45°W
122.40°W
$250,000
$200,000
$150,000
$100,000
$50,000

5 Geocoding

5.1 City Locator

There are several ways to geocode addresses from R, but the easiest (and cheapest) way is with the {tidygeocoder} package and one of the city's internal locators.

Note

The locator will only geocode San Francisco addresses.

```r
library(tidygeocoder)
library(sf)
```

```
Linking to GEOS 3.9.1, GDAL 3.4.3, PROJ 7.2.1; sf_use_s2() is TRUE
```

```r
library(mapview)
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2
--
```

```
v ggplot2 3.5.1     v purrr   1.0.2
v tibble  3.2.1     v dplyr   1.1.0
v tidyr   1.2.1     v stringr 1.5.0
v readr   2.1.3     v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

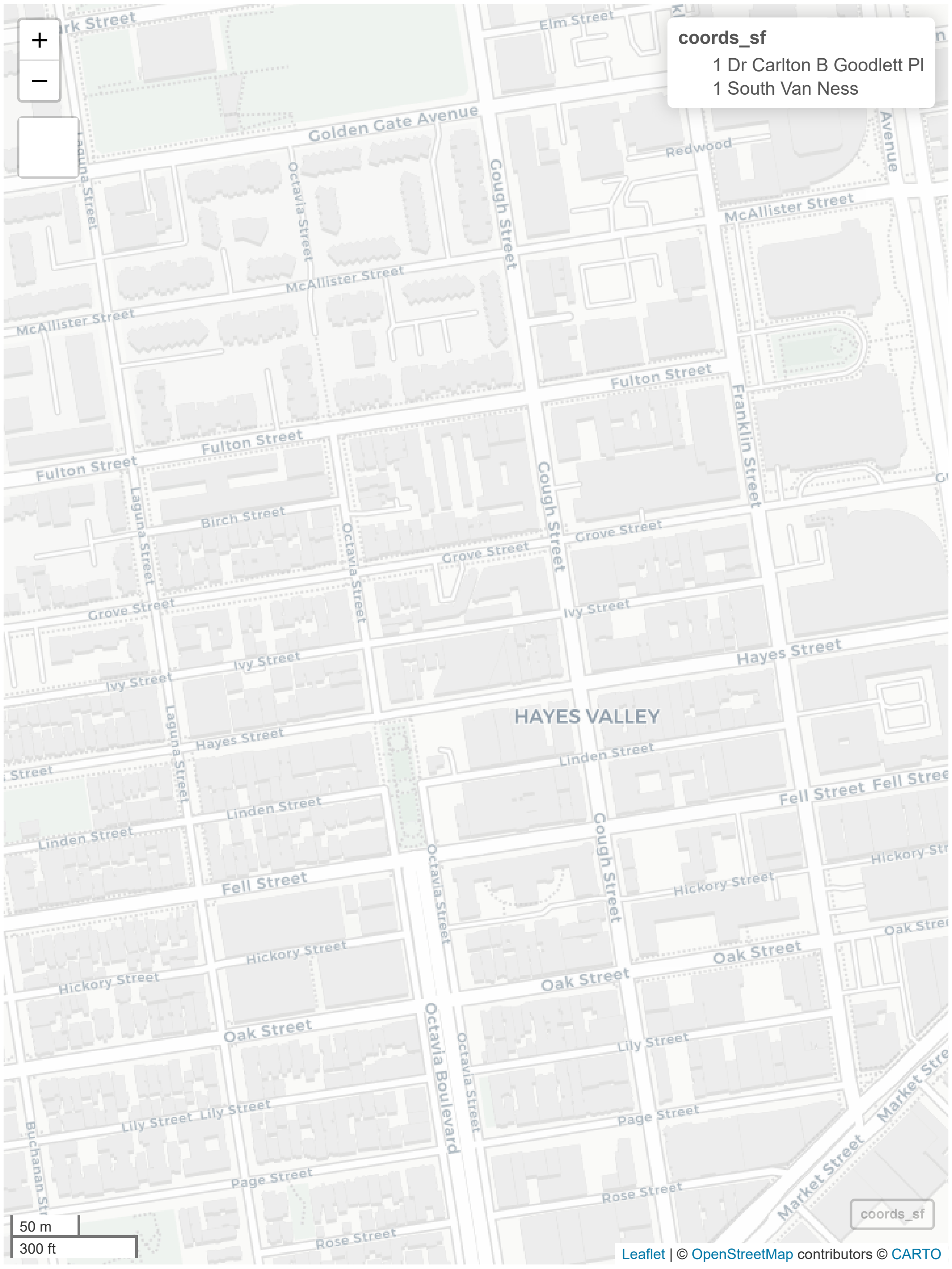
```r
df <- tibble(address = c("1 South Van Ness", "1 Dr Carlton B Goodlett Pl"))
locator <- "https://gis.sf.gov/dahl/rest/services/app_services/NRHP_Composite/GeocodeServer/

coords <- df %>%
  geocode(
    api_url = locator,
    address = address,
    custom_query = list(outSR = "4326"), # outSR (Spatial Reference) is a required parameter
    method = "arcgis"
  )
```

```
Passing 2 addresses to the ArcGIS single address geocoder
Query completed in: 0.2 seconds
```

```r
coords
```

```
# A tibble: 2 x 3
  address                       lat  long
  <chr>                       <dbl> <dbl>
1 1 South Van Ness             37.8 -122.
2 1 Dr Carlton B Goodlett Pl   37.8 -122.
```

```r
coords_sf <- coords %>% st_as_sf(coords = c("long", "lat"), crs = 4326)
mapview(coords_sf)
```


coords_sf
1 Dr Carlton B Goodlett Pl
1 South Van Ness
HAYES VALLEY
50 m
300 ft
coords_sf
Leaflet | © OpenStreetMap contributors © CARTO

5.2 geocodio

If you need to geocode addresses outside the city, the geocodio service is a nice option, but you'll first need to obtain your API key. Sign up for an account and register for an API key. Once you have it, you need to put it in your .Renviron file, a special text file that runs every time you open/restart R.

Edit your .Renviron file with the usethis package:

```r
library(usethis)
edit_r_environ() # this opens the file in RStudio
```

Paste your API key like so:

```
GEOCODIO_API_KEY='<your_api_key>'
```

Save the file and restart R. You should then be able to call `geocode` with the `method = 'geocodio'` argument. Note that there is a rate limit of 1000 addresses per hour.

6 Icons

The Digital Services team has provided a nifty set of icons on the San Francisco Design System website. You can use these icons in Quarto (HTML) documents by installing the sficons extension from GitHub here.

```
quarto install extension SFOEWD/sficons
```

To embed an icon, use the shortcode. Some examples:

```
{{< sficon wip >}}
{{< sficon alert >}}
```

```
{{< sficon arrow-right color=firebrick >}}

{{< sficon globe color=green size=5em >}}

{{< sficon pencil color=gold size=10em >}}
```

Control the color and size of the icons:

7 Snowflake

WIP

7.1 Setup

You will first need to install Snowflake's ODBC Driver. Then configure the DSN for either Windows or macOS.

7.2 Making the Connection

I wrapped the connection into a function:

```
connect_to_snowflake <- function() {
  conn <- DBI::dbConnect(
    odbc::odbc(),
    "<data_source_name>",
    uid = "<user_id>",
    pwd = rstudioapi::askForPassword())
  return(conn)
}

con <- connect_to_snowflake()
```

This works fine for interactive analysis, but you will need to stash your password as an environment variable for various workflows, or perhaps within {targets} projects. (see below)

7.3 Using SQL

Like any other database connection, you can pass SQL queries to the connection as a string.

⚠ Warning

If you set a default schema when configuring your Snowflake data source, you must explicitly reference other schemas when querying tables outside of it.

```
DBI::dbGetQuery(
  con,
  "select table_name, last_altered
  from information_schema.tables
  where table_name like 'STG%' limit 5
  ")
```

7.4 Using R

```
is <- tbl(con, in_schema("INFORMATION_SCHEMA", "TABLES"))
is %>%
  select(TABLE_NAME, LAST_ALTERED) %>%
  filter(str_detect(TABLE_NAME, '^STG')) %>%
  head(5) %>%
  collect()
```

7.5 Snowflake tables as targets

We can use `tarchetypes::tar_change()` and the `LAST_ALTERED` field referenced above as a trigger in {targets} pipelines. If the date changes, the target will rerun alongside with downstream targets. Here's a small example of how that might work:

`_targets.R`

```
library(targets)
library(tarchetypes)

tar_option_set(
  packages = c(
    "tibble",
    "DBI",
    "odbc",
    "dplyr",
    "dbplyr"
    )
  )

connect_to_snowflake <- function() {
  conn <- DBI::dbConnect(
    odbc::odbc(),
    "<dsn_name>",
    uid = "<user_id",
    pwd = "<password>")
  return(conn)
}

tar_plan(
  snowflake_con = connect_to_snowflake(),
  tar_change(
    my_table,
    collect(tbl(snowflake_con, "MY_TABLE")),
    change = dbGetQuery(
      snowflake_con,
      "select last_altered from db.information_schema.tables where table_name = 'MY_TABLE' ar
      )
  ),
  my_table_transformed = head(my_table)
)
```

8 {targets} Pipelines

WIP