Data Science for Public Policy

Alex Engler, Aaron R. Williams, & Alena Stern - Georgetown University

PPOL 670 | Assignment 05
APIs and Geospatial Analysis

Due Date: Friday, March 18th at 6:00 PM

Deliverable: There are three deliverables to submit for this assignment.

1. the resulting project .html file
2. the .Rmd file with your R code
3. the URL of the Git repository

Note: You must use a private Git repository for this assignment. There is an explanation of how to do this under 'Setup' below. Since you are all working on the same assignment, and GitHub repositories are public by default, this is an academic integrity issue. Thus, you absolutely must **only push your work to a private repository**. Please reach out with any questions or concerns.

Points: 10 points (plus 5 points for stretch exercise)

Plagiarism on homework or projects will be dealt with to the full extent allowed by Georgetown policy (see http://honorcouncil.georgetown.edu).

Setup

Create a new folder with a new R project (`.Rproj`) and R Markdown file (`.Rmd`). Then create a new **Private GitHub repository**. To do this, you need to select the "Private" option, directly below the "Description" when you initialize a new GitHub repository. Then you need to grant access to the instructors and teaching assistant. If you are working with a partner, you will also need to provide access to your partner. To do this:

- Click the "Settings" option on the right side of the top menu of your GitHub repo
- Click "Manage access", the second option on the left-side menu on this page
- Scroll down and click the green button "Invite a collaborator"
- Add awunderground and ncstabile17 (section 01) or alenastern and joshrrosen (section 02)
- Add your partner (if working with a partner)

Assignment Description

This assignment will focus on using real-world data to create a presentable, reproducible plot that can be updated easily. As we covered in class, R Markdown is ideal for programmatically generating reports that rely on constantly changing data. We will use this to create a report on the geographic distribution of crime in Chicago, focusing on homicides. The end goal is to create a map and a short paragraph discussing the map's implications.

Note that the data used in this assignment contains 7 million rows, so if you run into trouble with your computer's computational abilities, please contact the instructors.

Grading Rubric

- [1 Point] Create a private, well-managed GitHub repository, including an appropriate `.gitignore` (ignoring the data and Census API credentials) and an informative `README.md` file. At a minimum, your README should provide a brief overview of the objective of the code, a brief description of the files/directories in the repository, and instructions for how to run your code to replicate your analysis.
- [1 Point] Write a clean and well-composed `.Rmd file`, including separate named code chunks for each task required below. The resulting `.html` file should show code and results, but hide unnecessary warnings and messages.
- [5 Points] R code for geospatial analysis
- [3 Points] R code for querying an API

Download Chicago Crimes Data

This assignment will use crime and spatial data from the City of Chicago Data Portal. We will use two datasets from here, though we have provided a smaller version of the crimes data set on Canvas for ease of use. Please download that dataset `crimes_reduced.csv`, from Canvas.

- The crimes dataset, 2001 to present (CSV): Link
- The 2010 census tract boundaries (Shapefile): Link

Download the 2010 census tract boundaries **as a shapefile** using the Export button on the top right of its page. Note you need to unzip the shapefile before you can work with it. Also note that you need to keep all four files in the same folder, even if you only write code to interact with the `.shp` file (this is just how Shapefiles work). You should place these files and the other data files in a sub-folder called `data`. Add `data/` to your `.gitignore`.

1. Data Loading & Cleaning (1 Point)

First, you must read the crimes data into R. To prevent `read_csv` from misreading the data types, you must specify *as you read in the data* that the `Longitude` and `Latitude` columns are of type character. For more information, you can read this chapter in R4DS.

Note that there are several variable names in this data that have white space and varying levels of capitalization - write code to replace the white space with underscores and lower-case all of the letters in the column names using `names()` or `rename()`. The stringr package, which is part of the larger tidyverse, is useful for manipulating character variables. Unlike `library(dplyr)` or `library(ggplot2)`, which are loaded by `library(tidyverse)`, `library(stringr)` need to be explicitly loaded. The stringr cheatsheet is a useful resource.

2. Filtering the data to homicides within ten years of today (1 Point)

Now, filter the data to rows about homicides that have valid longitudes and latitudes (e.g. neither are NA) and filter the data to only dates within the past ten years of today. Imagine this is meant to be a living document, always providing context for the last ten years of data. So, write a filter that uses functions from the lubridate package to filter crimes down to within ten years of today without hardcoding any dates. The lubridate cheatsheet is a useful resource.

If you do this correctly, you should have around five thousand rows of data remaining. Your results will not exactly match those below, but they should be close.

```
table(year(crimes_lim$date))
```

```
##
## 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022
##  444  431  429  502  790  676  601  507  792  806   81
```

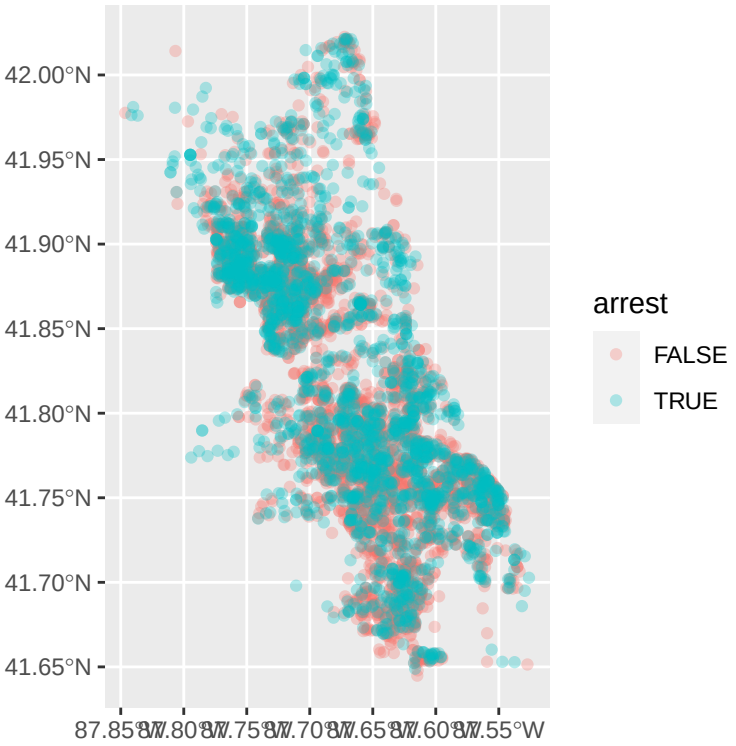
```
min(crimes_lim$date)
```

```
## [1] "2012-03-01"
```

3. Convert Lon/Lat to Points Geometry (1 Point)

Now that we've filtered our dataset to contain only recent murders, let's take a look at their geographic distribution. We need to convert the Longitude and Latitude columns in the crimes data into spatial geometries (points) before plotting. Look to the function we used to do this in class, specify CRS as 4326, and look to this function's arguments - find the one that allows you to keep the original Latitude and Longitude columns in the data.

Once you've successfully converted the lon/lat to point geometries, plot the map using ggplot2 and color the dots by whether there was an arrest - your results should look like those below.



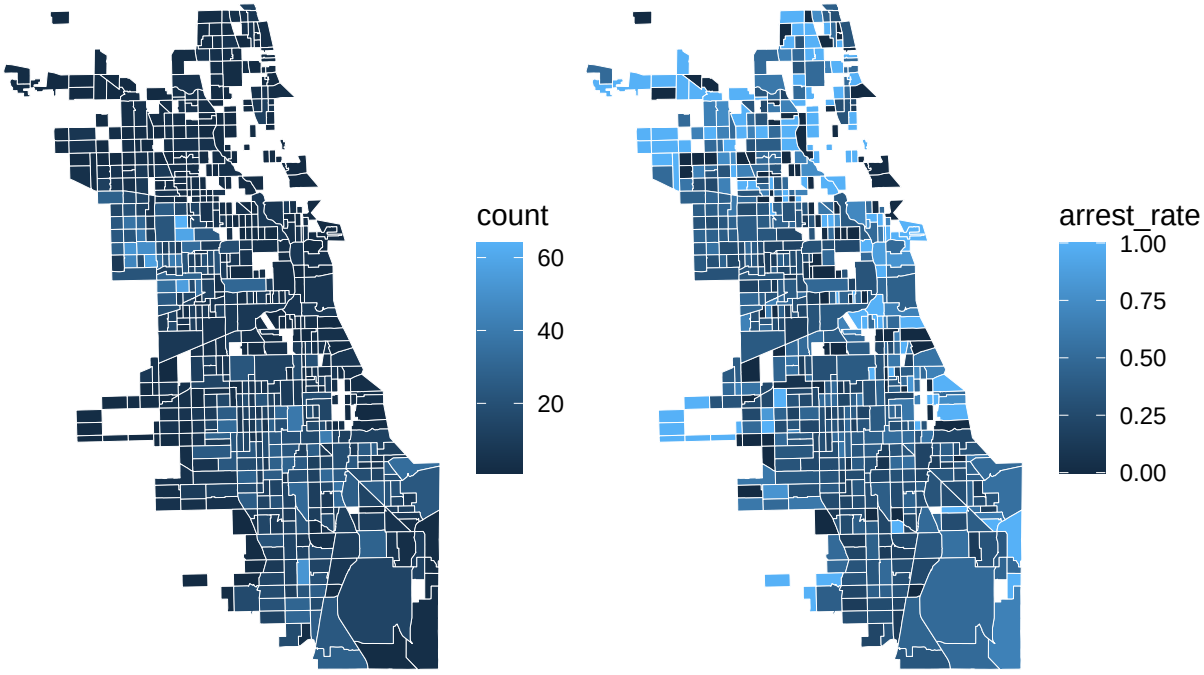
4. Load Census Tracts, Perform a Spatial Join, and Create Choropleth (2 Points)

Load the Chicago shapefile that you downloaded from Chicago's Open Data Portal. In R, overwrite this sf dataframe with a smaller version of itself, including only the `geoid10` column and the geometry.

Perform a spatial join, associating each point in the crimes data with a census tract polygon. Next calculate the count of homicides and the percent of arrests per homicide by census tract in a new dataframe `chicago_merged_agg`. Join the census tract geometry back to this new dataframe, and recreate the maps below. You can create the maps separately or use patchwork to combine them like below.

This e-book chapter has more information on spatial operations and spatial joins. Link

```
## # A tibble: 6,059 x 11
##          id date       primary_type arrest x_coordinate y_coordinate
##   *    <dbl> <date>     <chr>        <lgl>         <dbl>        <dbl>
##  1 10156667 2015-07-18 HOMICIDE     TRUE        1175712      1902172
##  2 10164471 2015-07-23 HOMICIDE     TRUE        1138403      1922795
##  3    22032 2015-08-15 HOMICIDE     TRUE        1152321      1902190
##  4 10222837 2015-09-03 HOMICIDE     TRUE        1168593      1938326
##  5 10372192 2015-12-12 HOMICIDE     TRUE        1170546      1857571
##  6 11823422 2019-08-10 HOMICIDE     TRUE        1180077      1864658
##  7    24568 2019-05-29 HOMICIDE     FALSE       1179224      1829773
##  8 10569766 2016-06-21 HOMICIDE     TRUE        1177193      1877179
##  9 10586888 2016-07-05 HOMICIDE     TRUE        1158439      1875197
## 10    25665 2020-12-11 HOMICIDE     TRUE        1178967      1868946
## # ... with 6,049 more rows, and 5 more variables: community_area <dbl>,
## #   latitude <chr>, longitude <chr>, location <chr>, geoid10 <chr>
```



5. Using the Census API (3 Points)

We will explore the Census API in two different ways - first with the R package tidycensus, then directly with the Census API.

To begin, get an API Key for the US Census Bureau. Install and load the `tidycensus` R package and save your API key using the `census_api_key()` function (you can use `install = TRUE` argument to permanently install the Census key into RStudio).

Then, in a separate R chunk, use the tidycensus package to retrieve median household income, population with a bachelor's degree, and population below the poverty line for all of the census tracts in Cook County, Illinois. Use the 2013-2017 5-year ACS as your data source. You can find the variable names, which follow the pattern "B#####_###" with tidycensus using the `tidycensus::load_variables()` function or with the Census Bureau documentation. **Hint:** You can assign `load_variables()` to an object, view the

object, and use the filter option to search the `label` and `concept` columns to identify the right variable code. Sometimes, a variable of interest will appear in multiple different tables.

Finally, examine the Census ACS API documentation and create a URL string that retrieves the same data. Use the `httr` and `jsonlite` packages to write a `GET` request and pull this data down. Parse the response into a dataframe and confirm by writing a unit test using `library(testthat)` that the resulting data is the same from this API query as the one from the tidycensus query. **Hint:** check out the `dplyr::all_equal()` function. You may need to perform some transformations before running the test including excluding non-matching columns, converting column types, and replacing missing value codes with NAs before running the test.

Stretch (5 points)

This stretch exercise asks you to deepen your spatial analysis skills and your ability to use `purrr` and custom functions for iteration.

Part 1: Identifying Crimes Near Transit Stations (3 points)

In this part, we will identify the percentage of crimes that occur near Chicago Transit Authority (CTA) 'L' (rail) train stations.

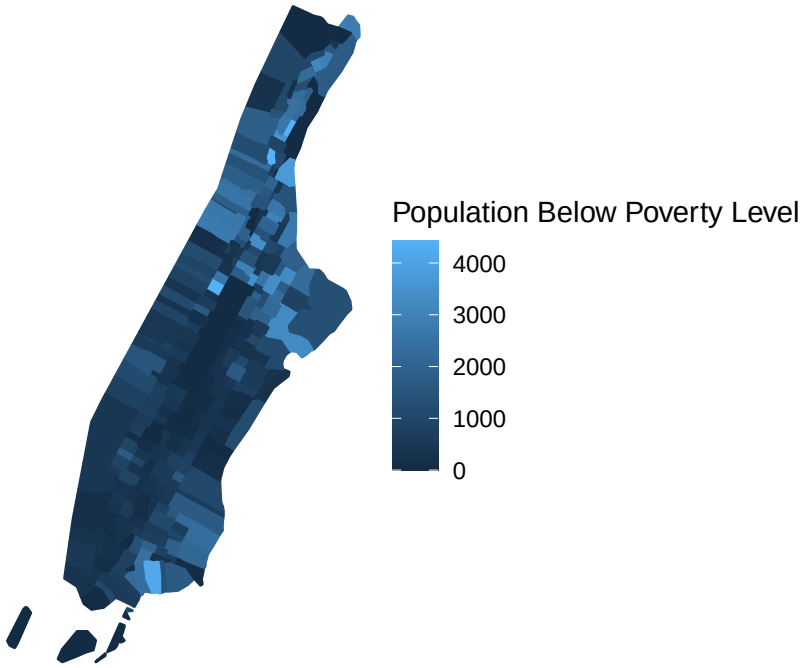
1. Download the shapefile of CTA 'L' Train Station Locations from the Chicago Open Data Portal and unzip the file in your data folder. Read the shapefile into R.
2. Using the `chicago_gdf` object earlier, map the 'L' stations overlaid on the Chicago census tracts. Notice that some of the 'L' stations are outside of the city of Chicago.
3. Use `st_union()` to create a polygon representing the boundary of Chicago (your computer may need time to finish this step) and perform a spatial join between this boundary and the 'L' stations to exclude all stations that are outside of the city of Chicago. Assign the result of this join to an object called `stations_filtered`. **Hint:** you will need to convert the output of `st_union()` to an sf object using `st_sf()` before performing the join.
4. Plot `stations_filtered` and the Chicago tracts again to confirm that you've filtered out all stations outside the city.
5. Identify all crimes within 400 meters (approximately a quarter mile) of each 'L' station by performing a spatial join between `crimes_lim` and `stations_filtered`. **Hint:** you will need to use `st_buffer()`.
6. Which 'L' Station has the highest number of crimes within 400 meters of the station?

Part 2: Iterating API pulls and map creation using `purrr` (2 points)

1. Write a simple function that takes four arguments: numeric year, Census variable code as a text string, human-readable variable name as a text string (e.g. "household income" or "educational attainment"), state FIPS code as a text string, and county FIPS code as a text string. Within the function, run a query using tidycensus (pull census tract level data), then create a simple choropleth of the resulting data using ggplot and `geom_sf()`. You should set the label for `fill` to be the human-readable variable name. **Important Tip:** rename the Census variable to a generic name before passing it to `ggplot()` (otherwise you will not be able to reference it). More information about custom functions is available in the functions chapter of R4DS. Document your function with a Roxygen skeleton.
2. Add a human readable title to your map. For example, if you mapped total population in Chicago, Illinois the title would be "Total Population by Tract in Chicago, Illinois", **Hint:** Parse the `NAME` column returned by `get_acs()` to obtain the county and state to create the title (stringr functions will be helpful here). After completing steps 1 and 2, the output of your function should look like the maps below.
3. Modify the function to save the image **instead of returning the ggplot object** in an `images` directory in your assignment repository.
4. Use `purrr` to iterate over at least 5 different combinations of year, variable, state/county with your modified function. Push the maps that you save in your `images` directory to GitHub.

Population With Bachelor's Degree in District of Columbia, District of Colum

Population Below Poverty Level in New York County, New York



Submission

Upon completion of the assignment, knit the .Rmd file to .html, and submit both, along with the URL of the GitHub Repository to Canvas. You must also have shared the GitHub repository as described in the 'Setup' section.