Extracting data from text and geocoding to study officer-involved shootings

Greg Ridgeway

2025-09-16

Table of contents

1 Introduction

In this section, we are going to explore officer-involved shootings (OIS) in Philadelphia. The Philadelphia Police Department posts a lot of information about officer-involved shootings online going back to 2016. Have a look at their OIS webpage. While a lot of information has been posted to the webpage, more information is buried in text linked to each of the incidents. In order for us to explore these data, we are going to scrape the basic information from the

webpage, have R dig into the text for dates, clean up addresses using regular expressions, geocode the addresses to latitude/longitude with the ArcGIS geocoder (using JSON), and then make maps describing the shootings.

Start by loading the packages we will need.

```
library(lubridate)
library(jsonlite)
library(sf)
library(leaflet)
library(dplyr)
library(tidyr)
library(foreach)

library(chromote) # steer Chrome from R
library(rvest)    # helpful webscraping tools
library(purrr)    # for pluck()
```

2 Scraping the OIS data

We can run `scan()`, as we did previously, on the PPD OIS webpage and regex our way to a data frame with the data elements that we want to store.

```
# pull the whole OIS page
a <- scan("https://www.phillypolice.com/accountability/ois/",
          what="", sep="\n")
# search for the start of a table
i <- grep("<tbody>", a)
a[i] |> substring(1, 500)
```

```
[1] "   </script><script id=\"7df2eafa34b89fa67704cfd13478287f-1\"
 ↪  type=\"nitropack/inlinescript\"
 ↪  class=\"nitropack-inline-script\">jQuery(document).ready(function(){var
 ↪  e=jQuery(\"#ast-seach-full-screen-form\");jQuery(\".site-header\").after
 ↪  (e);jQuery(\"#close\").removeAttr(\"tabindex\");var
 ↪  t=jQuery(\".ast-header-html-2\");jQuery(\".ast-search-wrapper
 ↪  .ast-container\").html(t);t.css(\"display\",\"flex\");jQuery(\"#closeSea
 ↪  rch\").click(function(e){e.preventDefault();var
 ↪  t=jQuery(\"#ast-seach-full-screen-form\");t.animate({\""
```

```
# search for start of table <tbody> in a[i]
iStart <- gregexpr("<tbody>",  a[i]) |> unlist()
a[i] |> substring(iStart, iStart+500)
```

```
[1] "<tbody><tr><td><a
 ↪ href=\"https://www.phillypolice.com/ois/25-14/\">25-14</a></td><td>2600
 ↪ block of South 21st Street</td><td>2025</td><td>N/A</td><td>N/A</td><td>
 ↪ No</td><td>Pending</td><td></td> <td><p style=\"text-align:justify;\">On
 ↪ June 20th, 2025, at approximately 10:41 a.m., Officer A*, equipped with
 ↪ an Animal Control Pole, responded to a radio call in the 2600 block of
 ↪ South 21st Street, in reference to a dog bite victim.</p> <p
 ↪ style=\"text-align:justify;\">Note: Officers initially responded to "
```

Buried in the HTML code are the entries in the table. Even though all the information does not appear on the main webpage, it is in the HTML. This is not always the case. Some pages dynamically generate information as the user interacts with a page and the page elements. We are going to use this as an opportunity to learn more advanced webscraping methods.

Instead of using `scan()`, we are going to use the `chromote` package to open a hidden Chrome browser that we can control remotely from R. For this to work you do need to have a Chrome browser installed on your computer. From R, we can simulate user actions, like typing a URL in the address bar, selecting elements on the page, and clicking buttons.

The *page source* is the static HTML the browser receives from the server (e.g. what `scan()` would capture). The *Document Object Model* (DOM) is the live, in-memory object the browser constructs from that source and then updates as JavaScript runs. When we interact with a webpage, like clicking, we are interacting with the DOM.

Start by initiating a new hidden Chrome browser.

```
browser <- ChromoteSession$new()
# Good idea to setup the browser to close when we exit R
#    When creating these notes it runs too soon, so commented out here
# on.exit(browser$close(), add = TRUE)
```

Nothing will be visible on your screen at this point. If you want to watch what the browser is doing in response to R actions, you can use `view()`, but this is not necessary.

```
browser$view()
```

Let's tell our hidden browser to navigate over to the PPD OIS webpage. Sometimes pages take a moment to load. `go_to()` waits for the navigation to finish before allowing R to move on to the next line of code.

```
browser$go_to("https://www.phillypolice.com/accountability/ois/")
```

I will grab a screenshot to show that everything is working so far.

```
# get page size
pageSize <- browser$Page$getLayoutMetrics()$contentSize
browser$screenshot(cliprect = c(left=0, top=0,
                                width=pageSize$width, height=800))
```

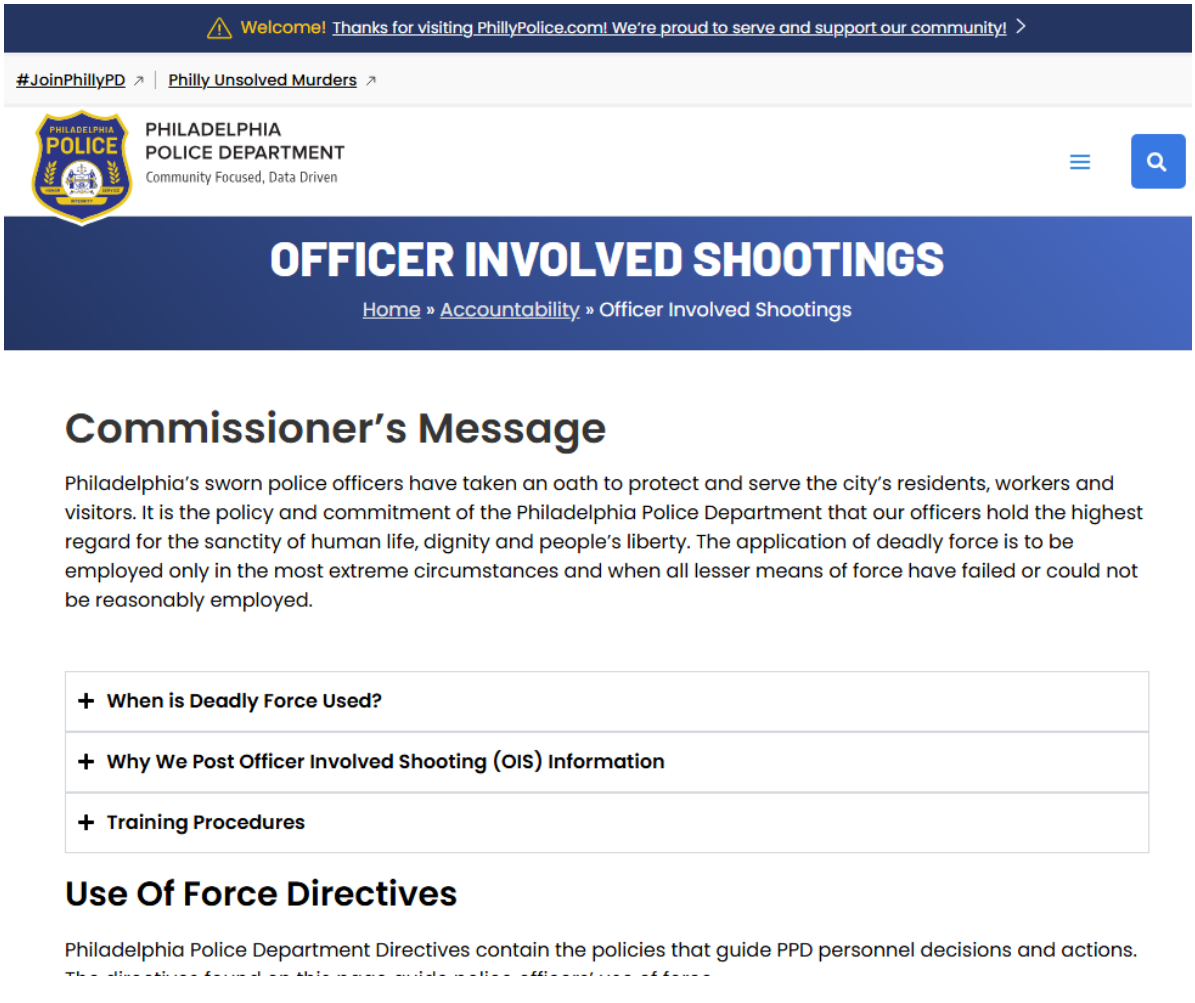

Figure 1: Screenshot from the hidden Chrome browser view of the PPD OIS

Let's take a little cybersecurity detour for a moment. If you right-click on a webpage and select "Inspect," then you can see the DOM for the page. Right-click and Inspect the OIS table and you will find that the table's ID is `#data-table-ois`. From R I can tell the DOM to change elements on the page. To demonstrate, I will change the address in the 25th row to 3718 Locust Walk. `tr:nth-child(25)` selects the 25th row and `td:nth-child(2)` selects the column. Setting `.textContent = '3718 Locust Walk'` changes the cell text in the DOM immediately.

```
browser$Runtime$evaluate("document.querySelector('#data-table-ois tr:nth-child(25)
    td:nth-child(2)').textContent = '3718 Locust Walk';")
```

```
$result
$result$type
[1] "string"

$result$value
[1] "3718 Locust Walk"
```

Now let's look at the page

```
pageSize <- browser$Page$getLayoutMetrics()$contentSize
browser$screenshot("screenshot2.png",
             cliprect = c(left=0, top=pageSize$height-800,
                          width=pageSize$width, height=800))
```

This is how the "refund/overpayment scam" works. A fraudster gets the victim to view their account webpage and to give the fraudster some level of remote control. The fraudster then manipulates the DOM to give the impression that they have mistakenly sent the victim $10,000 instead of $1,000. The fraudster then convinces the victim to send the $9,000 difference even though no money was ever given to them.

Rather than try to scam someone, let's get back to the business of pulling all the OIS data into R. I will first get the ID of the main HTML from the DOM. Then I will read in the HTML associated with that ID.

```
# get the ID of the main HTML
html <- browser$DOM$getDocument()$root$nodeId
html
```

```
[1] 16
```

```
# pull in raw html source code, like view page source or scan()
page_html <- browser$DOM$getOuterHTML(nodeId = html)$outerHTML
# page_html is one long string of HTML code
#   look at a few lines
page_html |>
   substring(1,1500)
```

24-23	3000 block of Ruth Street	2024	N/A	N/A	No
24-22	6100 block of West Columbia Avenue	2024	N/A	N/A	No
24-21	3500 block of F Street	2024	No	Yes	Yes
24-20	3718 Locust Walk	2024	Yes	Yes	No

Showing 1 to 25 of 147 entries Previous 1 2 3 4 5 6 Next

Figure 2: Screenshot of the manipulated PPD OIS webpage

```
[1] "<!DOCTYPE html><html lang=\"en-US\"><head><meta
 ↪  http-equiv=\"origin-trial\" content=\"A7vZI3v+Gz7JfuRolKNM4Aff6zaGuT7XOm
 ↪  f3wtoZTnKv6497cVMnhy03KDqX7kBz/q/iidW7srW31oQbBt4VhgoAAACUeyJvcmlnaW4iOi
 ↪  JodHRwczovL3d3dy5nb29nbGUuY29tOjQ0MyIsImZlYXR1cmUiOiJEaXNhYmxlVGhpcmRQYX
 ↪  J0eVN0b3JhZ2VQYXJ0aXRpb25pbmczIiwiZXhwaXJ5IjoxNzU3OTgwODAwLCJpc1N1YmRvbW
 ↪  FpbiI6dHJ1ZSwiaXNUaGlyZFBhcnR5IjpOcnVlfQ==\"><script async=\"\"
 ↪  src=\"//cse.google.com/adsense/search/async-ads.js\"></script><script
 ↪  type=\"text/javascript\" async=\"\" charset=\"utf-8\"
 ↪  src=\"https://www.gstatic.com/recaptcha/releases/44LqIOwVrGhp2lJ3fODa493
 ↪  O/recaptcha__en.js\" crossorigin=\"anonymous\"
 ↪  integrity=\"sha384-81KiK8GhWyH8OMsZl1vdzmMYc0XQX9aDGqEfT0ckWF5GLPX10rwo5
 ↪  y8vjpybBo6f\"></script><script>if(navigator.userAgent.match(/MSIE|Intern
 ↪  et Explorer/i)||navigator.userAgent.match(/Trident\\/7\\..*?rv:11/i)){let
 ↪  e=document.location.href;if(!e.match(/[?&]nonitro/)){if(e.indexOf(\"?\")
 ↪  ==-1){if(e.indexOf(\"#\")==-1){document.location.href=e+\"?nonitro=1\"}e
 ↪  lse{document.location.href=e.replace(\"#\",\"?nonitro=1#\")}}else{if(e.i
 ↪  ndexOf(\"#\")==-1){document.location.href=e+\"&nonitro=1\"}else{document
 ↪  .location.href=e.replace(\"#\",\"&nonitro=1#\")}}}}</script><link
 ↪  rel=\"preconnect\" href=\"https://www.phillypolice.com\"><link
 ↪  rel=\"preconnect\" href=\"https://www.google.com\"><link
 ↪  rel=\"preconnect\" href=\"https://www.googletagmanager.com\"><link
 ↪  rel=\"preconnect\" href=\"https://cdn-ilcomil.nitrocdn.com\"><meta
 ↪  charset=\"UTF-8\"><meta name=\"viewport\" content=\"width=device-width,
 ↪  initial-scale=1\"><meta"
```

Now convert all that HTML code into an HTML document object with which R knows how to work.

```
page <- read_html(page_html)
page
```

```
{html_document}
<html lang="en-US">
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8
 ↪  ...
[2] <body itemtype="https://schema.org/WebPage" itemscope="itemscope" class="
 ↪  ...
```

In this page we are looking for any `table` elements.

```
page |> html_elements("table")
```

```
{xml_nodeset (4)}
[1] <table cellspacing="0" cellpadding="0" role="presentation" class="gsc-sea
 ↪  ...
```

```
[2] <table cellspacing="0" cellpadding="0" role="presentation" id="gs_id50" c
 ↪  ...
[3] <table id="data-table-ois" class="display dataTable no-footer" aria-descr
 ↪  ...
[4] <table cellspacing="0" cellpadding="0" role="presentation" class="gstl_50
 ↪  ...
```

There appear to be four tables on the page, but I noticed that the third one of these has `id="data-table-ois"`. That must be the one we want. Let's extract it by name and convert it to an R data frame object.

```
# extract the data-table-ois by name
page |>
   html_elements("table#data-table-ois") |>
   html_table() |>
   data.frame()
```

```
   Title                                Location Year Subject.Injury
1  25-14          2600 block of South 21st Street 2025            N/A
2  25-13        100 block of West Somerset Street 2025         Killed
3  25-12           300 block of North 65th Street 2025            N/A
4  25-11             4600 block of Roosevelt Blvd 2025         Killed
5  25-10               4100 block of Ogden Street 2025            N/A
6  25-09        4600 block of Roosevelt Boulevard 2025         Killed
7  25-08               1600 block of Moore Street 2025        Wounded
8  25-06              2800 block of Jasper Street 2025            N/A
9  25-05 1 Philadelphia International Airport Way 2025         Killed
10 25-04               4100 block of Leidy Avenue 2025            N/A
11 25-02          800 block of West Master Street 2025             No
12 25-01             600 block of Chamounix Drive 2025            N/A
13 24-37               3400 block of Vista Street 2024            N/A
14 24-36                   3200 block of A Street 2024            N/A
15 24-35          5400 block of Chancellor Street 2024         Killed
16 24-32                  2900 block of E. Street 2024            N/A
17 24-31               3300 block of Willits Road 2024         Killed
18 24-30             6100 block of Lebanon Avenue 2024            N/A
19 24-29            2600 block of Glenwood Avenue 2024            N/A
20 24-28           3900 block of Whittaker Avenue 2024        Wounded
21 24-27             2200 block of S. 65th Street 2024            N/A
22 24-23                3000 block of Ruth Street 2024            N/A
23 24-22       6100 block of West Columbia Avenue 2024            N/A
24 24-21                   3500 block of F Street 2024             No
25 24-20                        3718 Locust Walk 2024            Yes
   Subject.Arrested Officer.Injury
1               N/A             No
```

```
2               N/A             No
3               N/A            Yes
4               N/A             No
5               N/A             No
6               N/A            Yes
7               Yes             No
8               N/A            Yes
9               N/A             No
10              N/A            Yes
11              Yes            Yes
12              N/A             No
13              N/A             No
14              N/A             No
15              N/A             No
16              N/A             No
17              N/A             No
18              N/A             No
19              N/A             No
20               No             No
21              N/A             No
22              N/A             No
23              N/A             No
24              Yes            Yes
25              Yes             No
```

Looks like we got all 25 of the OIS incidents from the first page. Note that it still has our manipulated address for the 25th OIS incident. Shortly we will refresh the webpage to scrape all of the incidents and that will reset the addresses to their original values. On the PPD's page, each OIS's ID has a hyperlink that gets us more detailed information about each incident. Extract those URLs by looking for elements in the table body with an `<a>` HTML tag. Eventually we will store these URLs so that we can scrape them for the OIS details.

```
page |>
   html_elements("table#data-table-ois") |>
   html_elements("tbody a") |>
   html_attr("href")
```

```
[1] "https://www.phillypolice.com/ois/25-14/"
[2] "https://www.phillypolice.com/ois/25-13/"
[3] "https://www.phillypolice.com/ois/25-12/"
[4] "https://www.phillypolice.com/ois/ps25-11/"
[5] "https://www.phillypolice.com/ois/25-10/"
[6] "https://www.phillypolice.com/ois/25-09/"
[7] "https://www.phillypolice.com/ois/25-08/"
```

```
 [8] "https://www.phillypolice.com/ois/25-06/"
 [9] "https://www.phillypolice.com/ois/25-05/"
[10] "https://www.phillypolice.com/ois/25-04/"
[11] "https://www.phillypolice.com/ois/25-02/"
[12] "https://www.phillypolice.com/ois/ps25-01/"
[13] "https://www.phillypolice.com/ois/24-37/"
[14] "https://www.phillypolice.com/ois/24-36/"
[15] "https://www.phillypolice.com/ois/24-35/"
[16] "https://www.phillypolice.com/ois/24-32/"
[17] "https://www.phillypolice.com/ois/24-31/"
[18] "https://www.phillypolice.com/ois/ps24-30/"
[19] "https://www.phillypolice.com/ois/24-29/"
[20] "https://www.phillypolice.com/ois/24-28/"
[21] "https://www.phillypolice.com/ois/24-27/"
[22] "https://www.phillypolice.com/ois/24-23/"
[23] "https://www.phillypolice.com/ois/24-22/"
[24] "https://www.phillypolice.com/ois/24-21/"
[25] "https://www.phillypolice.com/ois/24-20/"
```

With that we have been able to get all of the information for the first 25 OIS incidents. Now we have to "click" the Next button to get to the next set of 25 OIS incidents. First, I will check to see if it is disabled. Since we are still looking at the first page it will not be disabled, but when we are looking at the final page of OIS incidents then it will be disabled. To figure out the name of that Next button, in my browser I right-clicked on the Next button, selected Inspect, and then reviewed the resulting HTML code.

```
<a class="paginate_button next" aria-controls="data-table-ois" role="link" data-dt-
    idx="next" tabindex="0" id="data-table-ois_next">Next</a>
```

I see that the HTML tag is `<a>` and its classes are `paginate_button` and `next`. So `a.paginate_button.next` will search the HTML code for an `<a>` element with both a `paginate_button` and `next` class.

```
# check whether the Next button is disabled
browser$Runtime$evaluate(
  expression = 'document.
                  querySelector("a.paginate_button.next").
                  classList.
                  contains("disabled");') |>
   pluck("result","value")
```

```
[1] FALSE
```

On a single page there might be several such buttons. A more robust method of finding the right button is to query it by name, in this case `data-table-ois_next`.

```
# check whether the Next button is disabled
browser$Runtime$evaluate(
  expression = 'document.
                  querySelector("#data-table-ois_next").
                  classList.
                  contains("disabled");') |>
   pluck("result","value")
```

```
[1] FALSE
```

Either method gives the same response: the button is not disabled. That means we can click it to move on to the next page.

```
# Click the "Next" button
browser$Runtime$evaluate(
  expression = 'document.
                  querySelector("#data-table-ois_next").
                  click();')
```

Armed with all the skills to navigate and extract key information from this site, we can wrap these ideas in a `while()` loop that will keep scraping OIS data as long as the Next button is active.

```
# reset page to beginning
browser$go_to("https://www.phillypolice.com/accountability/ois/")
# give the browser additional time to fully load page
Sys.sleep(5)
# you can force scroll to bottom if you have the view open
#   browser$Runtime$evaluate("window.scrollTo(0, document.body.scrollHeight);")

# store results from each page in ois (a list of data frames)
ois <- list()
isFinished <- FALSE
iPage <- 1
while(!isFinished)
{
  message(paste0("Read HTML from page ",iPage))
  # get ID of main HTML
  html <- browser$DOM$getDocument()$root$nodeId
  # raw html source code
  page_html <- browser$DOM$getOuterHTML(nodeId = html)$outerHTML
  # parse the HTML into a structured document
  page <- read_html(page_html)

  # Pull the table
  oisTable <- page |>
    html_elements("table#data-table-ois")
```

```
  # extract the table text to a data frame
  ois[[iPage]] <- oisTable |>
     html_table() |>
     data.frame()
  message("Read in ", nrow(ois[[iPage]]), " rows")

  # extract the URLs from each row
  oisURLs <- oisTable |>
    html_elements("tbody a") |>
    html_attr("href")
  ois[[iPage]]$url <- oisURLs

  # is "Next" button disabled?
  isFinished <- browser$Runtime$evaluate(
    expression = 'document.
                    querySelector("#data-table-ois_next").
                    classList.
                    contains("disabled");') |>
     pluck("result","value")

  if(!isFinished)
  {
    # Click the "Next" button
    browser$Runtime$evaluate(
      expression = 'document.
                      querySelector("#data-table-ois_next").
                      click();')
    Sys.sleep(2)
    iPage <- iPage + 1
  }
}
```

```
Read HTML from page 1
```

```
Read in 25 rows
```

```
Read HTML from page 2
```

```
Read in 25 rows
```

```
Read HTML from page 3
```

```
Read in 25 rows
```

```
Read HTML from page 4
```

```
Read in 25 rows
```

```
Read HTML from page 5
```

```
Read in 25 rows
```

```
Read HTML from page 6
```

```
Read in 22 rows
```

```
# close the browser
browser$close()
```

```
[1] TRUE
```

```
# combine pages into single data frame
#   drop year since we are going to extract the actual date later
ois <- bind_rows(ois) |>
  select(-Year) |>
  rename(id = Title,
         location  = Location,
         subInjury = Subject.Injury,
         subArrest = Subject.Arrested,
         offInjury = Officer.Injury)
```

Let's check that it read everything in.

```
head(ois)
```

```
     id                           location subInjury subArrest offInjury
1 25-14   2600 block of South 21st Street       N/A       N/A        No
2 25-13 100 block of West Somerset Street    Killed       N/A        No
3 25-12    300 block of North 65th Street       N/A       N/A       Yes
4 25-11      4600 block of Roosevelt Blvd    Killed       N/A        No
5 25-10        4100 block of Ogden Street       N/A       N/A        No
6 25-09 4600 block of Roosevelt Boulevard    Killed       N/A       Yes
                                      url
1   https://www.phillypolice.com/ois/25-14/
2   https://www.phillypolice.com/ois/25-13/
3   https://www.phillypolice.com/ois/25-12/
4 https://www.phillypolice.com/ois/ps25-11/
5   https://www.phillypolice.com/ois/25-10/
6   https://www.phillypolice.com/ois/25-09/
```

```
tail(ois)
```

```
       id                                 location subInjury
142 16-11             Unit Block of Salford street        No
143 16-10 5700 N. Park street/5700 N. Broad street    Killed
144 16-07      3100 Block of north Carlisle Street   Wounded
```

```
145 16-03                Near Loudon and D streets        No
146 16-02           100 block of north 55th Street        No
147 16-01           300 block of south 60th street   Wounded
               subArrest        offInjury
142 Yes (both offenders) P/O #1 (wounded)
143                  Yes               No
144                  Yes               No
145      2 of 4 arrested               No
146                  Yes               No
147                  Yes          Wounded
                                      url
142 https://www.phillypolice.com/ois/16-11/
143 https://www.phillypolice.com/ois/16-10/
144 https://www.phillypolice.com/ois/16-07/
145 https://www.phillypolice.com/ois/16-03/
146 https://www.phillypolice.com/ois/16-02/
147 https://www.phillypolice.com/ois/16-01/
```

3 Extracting OIS incident details

Now let's dig into the details of the incident, starting with the first OIS. The hyperlink in the very first OIS incident points to the page https://www.phillypolice.com/ois/25-14/. Let's read in the incident details from that page. In your browser, if you right-click and Inspect the text description of the incident, then you will find that the text has the id `ois-content-area`. We can grab that by name.

```
read_html(ois$url[1]) |>
   html_element("div.ois-content-area") |>
   html_text() |>
   trimws() # trim whitespace
```

```
[1] "On June 20th, 2025, at approximately 10:41 a.m., Officer A*, equipped
↪  with an Animal Control Pole, responded to a radio call in the 2600 block
↪  of South 21st Street, in reference to a dog bite victim.\nNote: Officers
↪  initially responded to a dog bite incident, which occurred on the 2100
↪  block of Shunk Street, where a 59-year-old victim, had been attacked by
↪  two brown pit bull dogs while walking on the 2100 block of Shunk Street.
↪  The victim sustained multiple injuries, including bite marks on her left
↪  cheek and chin, and was bleeding from the face. The victim was
↪  transported to Presbyterian Hospital in stable condition by Philadelphia
↪  Fire Department paramedics.\nUpon Officer A's arrival in the area of
↪  South 21st Street, the same two brown pit bull dogs were observed walking
↪  south on 21st Street toward Oregon Avenue. At that time, Officers #1 and
↪  #2 were attempting to control one of the dogs with an Animal Control
↪  Pole. The dogs then began chasing Officer #1 into the street, with one
↪  appearing to bite his hand.\nOfficers #1 and #2 both drew their service
↪  weapons and discharged multiple rounds at the dogs, striking both. The
↪  injured dogs fled and collapsed in a nearby parking lot. One of the dogs
↪  died at the scene; the other was secured and transported to ACCT Philly
↪  for treatment.\nNote: Officer A was present at the time of the
↪  officer-involved shooting but did not discharge their firearm\nOfficer #1
↪  is 34-years-old and a 2-year veteran of the Philadelphia Police
↪  Department\nOfficer #2 is also 34-years-old and is a 5-year veteran of
↪  the Philadelphia Police Department.\nBoth officers are assigned to the
↪  17th Police District.\nAs is standard procedure in all officer-involved
↪  shootings, both officers have been placed on administrative duty pending
↪  the outcome of investigations by the Internal Affairs Bureau, the Officer
↪  Involved Shooting Investigation Unit, and the District Attorney's
↪  Office."
```

Now we are ready to read in all the incidents' details.

```r
ois$text <- NA
for(i in 1:nrow(ois))
{
   # message(paste0("Incident: ",i))
   a <- try( read_html(ois$url[i]) )

   if(inherits(a, "try-error"))
   { # in case a page does not exist
      message(paste0("Could not access webpage for ", ois$id[i]))
   } else
   {
      # grab text between <div class="ois-content-area"> and </div>
      ois$text[i] <- a |>
         html_element("div.ois-content-area") |>
```

```
        html_text() |>
        trimws()
  }
}
```

And let's just check that we have some descriptions now.

```
ois$text |> substring(1, 30)
```

```
 [1] "On June 20th, 2025, at approxi"   "On Wednesday, May 21, 2025, at"
 [3] "On Wednesday, April 30, 2025, "   "On Monday, April 28, 2025, at "
 [5] "4100 block of Ogden Street\nOn "  "4600 block of Roosevelt Boulev"
 [7] "1600 block of Moore Street\nOn "  "2800 block of Jasper Street On"
 [9] "1 Philadelphia International A"   "4100 block of Leidy Avenue\nOn "
[11] "800 West Master Street On Satu"   "3600 block of Chamounix Drive\n"
[13] "3400 block of Vista Street\nOn "  "3200 block of A Street\nOn Tues"
[15] "5400 block of Chancellor Stree"   "29oo block of E. Street\nOn Fri"
[17] "3300 Willits Road\nOn Thursday,"  "6100 block of Lebanon Avenue\nO"
[19] "2600 block of Glenwood Avenue\n"  "3900 block of Whittaker Avenue"
[21] "2200 block of S. 65th Street\nT"  "3000 block of Ruth Street\nThe "
[23] "6100 block of West Columbia Av"   "3500 block of F Street\nA Phila"
[25] "2700 block of North 6th Street"   "1500 block of North 57th Stree"
[27] "2100 block of East Westmorelan"   "1600 South Dover Street\nOn Thu"
[29] "3000 block of North 16th Stree"   "1500 block of South 58th Stree"
[31] "2200 block of West Oxford Aven"   "3900 block of Fairmont Avenue\n"
[33] "Unit block of East Cliveden St"   "2100 block of Eastburn Avenue\n"
[35] "1000 block of West Dakota Stre"   "6200 block of Haverford Avenue"
[37] "1000 block of North 48th Stree"   "300 block of Adams Avenue\nOn T"
[39] "2800 block of Mascher Street\nA"  "2300 block of Borbeck Avenue\nA"
[41] "3600 block of Sepviva Street\nA"  "1800 block of South 29th St.\nO"
[43] "8000 block of North Frankford "   "3700 block of Fairmount Street"
[45] "7500 block of Whitaker Ave.\nAt"  "1500 block of N. 62nd Street\nO"
[47] "Unit Block of E. Phil Ellena S"   "7600 block of Lexington Avenue"
[49] "3100 block of Emerald St.\nOn T"  "On Monday, August 14, at appro"
[51] "2300 block of Fawn Street\nOn T"  "400 block of West Bringhurst S"
[53] "15xx E. Johnson Street\nOn Frid"  "200 block of North 60th Street"
[55] "800 block of North 10th Street"   "3300 block of North 10th Stree"
[57] "1300 block of Chancellor Stree"   "500 block of East Brinton Stre"
[59] "400 block of South Street\nOn S"  "2200 block of West Hunting Par"
[61] "4700 block of Leiper Street\nOn"  "4000 block of Lancaster Avenue"
[63] "1700 block of Barbara Street\nO"  "2000 block of South Beechwood "
[65] "100 block of West Lehigh Avenu"   "4800 block of Keyser Street\nOn"
[67] "1700 Dickinson Street\nOn Wedne"  "2700 block of Brown Street\nOn "
[69] "1900 block of South Bancroft S"   "5700 block of Overbrook Avenue"
```

```
 [71] "4100 block of Parkside Avenue\n"  "Whitaker Avenue and Roosevelt "
 [73] "9th Street and Hunting Park Av"   "3000 block of North Water Stre"
 [75] "300 Glen Echo Road\nOn Monday, "  "Broad Street and Somerville Av"
 [77] "3300 Emerald Street\nOn Friday,"  "4700 block of Rorer Street\nOn "
 [79] "3500 block of Kyle Road\nOn Mon"  "3500 block of Wharton Street\nO"
 [81] "1900 block of East Hart Lane\nO"  "6100 block of  Locust Street\nO"
 [83] "5600 block of Greene Street\nOn"  "1400 block of Sharpnack Street"
 [85] "4200 block of Clarissa Street\n"  "6th Street and McKean\nOn Tuesd"
 [87] "2500 Block of South 7th Street"   "1500 block of Bailey Street\nOn"
 [89] "7600 Block of Roosevelt Blvd\nO"  "Jasper Street and Hart Lane\nOn"
 [91] "On November 21, 2019, at appro"   "On Saturday, November 2, 2019,"
 [93] "On 9-02-19, at 10:15 PM, two u"   "On May 20, 2019, at approximat"
 [95] "On Saturday, May 11th 2019 at "   "On Thursday, April 25, 2019, u"
 [97] "On Saturday April 20, 2019, at"   "On March 28, 2019, at approxim"
 [99] "On March 6, 2019, at approxima"   "OIS 18-28\nAt approximately 8:4"
[101] "OIS# 18-27\nOn November 13, 201"  "OIS# 18-26\nOn November 13, 201"
[103] "OIS # 18-25\nOn Wednesday, Nove"  "OISI # 18-22\nOn Saturday, Augu"
[105] "OISI # 18-19\nOn Monday, August"  "OIS# 18-17\nOn Thursday, August"
[107] "OIS# 18-16\nOn Monday, August 6"  "OIS # 18-12\nOn Friday, June 8,"
[109] "OIS# 18-08\nOn Wednesday, April"  "OIS# 18-02\nOn Monday, January "
[111] "OIS# 18-01\nOn Saturday, Januar"  "OIS# 17-37\nOn Wednesday, Decem"
[113] "OIS# 17-36\nOn Tuesday, Decembe"  "OIS# 17-30\nOn Saturday, Novemb"
[115] "OIS# 17-28\nOn Saturday, Septem"  "OIS# 17-25\nOn Saturday, August"
[117] "OIS# 17-23\nOn Friday, August 1"  "OIS# 17-22\nOn Monday, August 7"
[119] "OIS# 17-20\nOn Thursday, July 2"  "OIS# 17-19\nOn Wednesday, July "
[121] "OIS# 17-17 (June 8, 2017)\nOn T"  "OIS# 17-13\nOn Friday, May 12, "
[123] "OIS# 17-08 (March 29, 2017)\nOn"  "OIS# 17-03 (February 15, 2017)"
[125] "PS#16-43\n11/25/16\nOn Friday, N" "PS#16-40\n11/07/16\nOn Monday, N"
[127] "PS#16-38\n11/2/16\nOn Saturday, " "PS#16-37\n10/27/16\nOn Thursday,"
[129] "PS# 16-35\n10/19/16\nOn Saturday" "PS# 16-34\n10/19/16\nOn Wednesda"
[131] "PS# 16-33\n10/18/16\nOn Tuesday," "PS#16-32\n9/28/16\nOn Wednesday,"
[133] "PS#16-30\n9/16/16\nOn Friday, Se" "PS#16-29\n9/09/16\nOn Friday, Se"
[135] "PS#16-28\n9/08/16\nOn Thursday, " "PS# 16-26\n9/05/16\nOn Monday, S"
[137] "PS#16-19\n5/31/16\nOn Tuesday, M" "PS#16-18\n5/31/16\nOn Tuesday, M"
[139] "PS# 16-16 5/20/16 On Friday, M"   "PS#16-13\n5/04/16\nOn Wednesday "
[141] "PS#16-12\n5/03/16\nOn Tuesday, M" "PS# 16-11\n4/17/16\nOn Sunday, A"
[143] "PS#16-10\n4/09/16\nOn Saturday, " "PS#16-07\n3/17/16\nOn Thursday, "
[145] "PS#16-03\n2/04/16\nOn Thursday, " "PS#16-02\n2/02/16\nOn Tuesday, F"
[147] "PS# 16-01\n1/07/16\nOn Thursday,"
```

4 Extracting dates from the text

While the main OIS webpage did not give the date of the incident, the text details always show the date. We can extract those dates with regular expressions. The dates may come in a variety of formats, but we can use the `lubridate` package to parse them. Let's start with those where the date is spelled out.

```
# extract dates in January 11, 2024 format
#    (?x) - ignore spaces and newlines
#         lets me break regex into several lines
#    (?s) - allows .* to include \n
#    both (?x) and (?s) require perl=TRUE
#    \s - whitespace (spaces, tabs, line feeds, carriage return)
a <- gsub("(?xs)
  .*?(January|February|March|April|May|June|
         July|August|September|October|November|December)
  \\s* ([0-9]{1,2})
  ( , \\s* (20[0-9]{2}) )?.*",
          "\\1 \\2\\3", ois$text, perl=TRUE)
```

For those incidents matching that January 11, 2024 format, they should have less than 20 characters in them. Let's check those out.

```
a[nchar(a) < 20]
```

```
 [1] "June 20"            "May 21, 2025"       "April 30, 2025"
 [4] "April 28, 2025"     "March 22, 2025"     "March 20, 2025"
 [7] "March 19, 2025"     "February 4, 2025"   "February 3, 2025"
[10] "January 27, 2025"   "January 11, 2025"   "January 10, 2025"
[13] "December 10, 2024"  "November 12, 2024"  "November 10, 2024"
[16] "October 11, 2024"   "October 3, 2024"    "October 2, 2024"
[19] "September 26, 2024" "September 19, 2024" "September 6, 2024"
[22] "July 4"             "June 24, 2024"      "June 22, 2024"
[25] "June 15"            "June 5, 2024"       "May 30, 2024"
[28] "May 23, 2024"       "May 15, 2024"       "May 14, 2024"
[31] "May 12"             "May 1, 2024"        "April 20, 2024"
[34] "April 17, 2024"     "April 15, 2024"     "April 14, 2024"
[37] "April 10"           "February 15, 2024"  "January 26"
[40] "January 17, 2024"   "January 10, 2024"   "December 31, 2023"
[43] "December 10, 2023"  "November 4, 2023"   "October 4"
[46] "October 4, 2023"    "October 2, 2023"    "September 27, 2023"
[49] "September 14, 2023" "August 14"          "May 4, 2023"
[52] "April 29, 2023"     "March 24, 2023"     "February 8, 2023"
[55] "October 12, 2022"   "October 7, 2022"    "September 11, 2022"
[58] "September 10, 2022" "June 4, 2022"       "May 11, 2022"
```

```
 [61] "March 19, 2022"     "March 1, 2022"      "February 15, 2022"
 [64] "February 11, 2022"  "February 9, 2022"   "February 2, 2022"
 [67] "January 4, 2022"    "August 18"          "July 22, 2021"
 [70] "June 14, 2021"      "April 7, 2021"      "November 30, 2020"
 [73] "September 18, 2020" "August 18, 2020"    "June 23, 2020"
 [76] "May 9, 2020"        "April 10, 2020"     "February 28, 2020"
 [79] "February 20, 2020"  "November 21, 2019"  "November 2, 2019"
 [82] "May 20, 2019"       "May 11"             "April 25, 2019"
 [85] "April 20, 2019"     "March 28, 2019"     "March 6, 2019"
 [88] "December 5, 2018"   "November 13, 2018"  "November 13, 2018"
 [91] "November 7, 2018"   "August 25, 2018"    "August 20, 2018"
 [94] "August 9, 2018"     "August 6, 2018"     "June 8, 2018"
 [97] "April 18, 2018"     "January 29"         "January 13"
[100] "December 27, 2017"  "December 26, 2017"  "November 11, 2017"
[103] "September 23, 2017" "August 19, 2017"    "August 11, 2017"
[106] "August 7, 2017"     "July 27, 2017"      "July 19"
[109] "June 8, 2017"       "May 12, 2017"       "March 29, 2017"
[112] "February 15, 2017"  "November 25, 2016"  "November 7, 2016"
[115] "October 29, 2016"   "October 27, 2016"   "October 22, 2016"
[118] "October 19, 2016"   "October 18, 2016"   "September 9, 2016"
[121] "September 16, 2016" "September 9, 2016"  "September 8, 2016"
[124] "September 5, 2016"  "May 31, 2016"       "May 31, 2016"
[127] "May 20, 2016"       "May 4, 2016"        "May 3, 2016"
[130] "April 17, 2016"     "April 9, 2016"      "March 17, 2016"
[133] "February 4, 2016"   "February 2, 2016"   "January 7, 2016"
```

The code seems to work for many dates, but we also see that some of the dates did not include the year. The first two digits of the incident ID are the last two digits of the year.

```
a[nchar(a) < 20 & !grepl("20[0-9]{2}", a)]
```

```
 [1] "June 20"    "July 4"     "June 15"    "May 12"     "April 10"
 [6] "January 26" "October 4"  "August 14"  "August 18"  "May 11"
[11] "January 29" "January 13" "July 19"
```

```
# first two digits of id have the year
i <- nchar(a) < 20 & !grepl("20[0-9]{2}", a)
paste0(a[i], ", 20", substring(ois$id[i],1,2))
```

```
 [1] "June 20, 2025"    "July 4, 2024"     "June 15, 2024"    "May 12, 2024"
 [5] "April 10, 2024"   "January 26, 2024" "October 4, 2023"  "August 14,
  ↪  2023"
 [9] "August 18, 2021"  "May 11, 2019"     "January 29, 2018" "January 13,
  ↪  2018"
[13] "July 19, 2017"
```

The rest of the dates have formats that are in some variation of 01/11/2024 or 01-11-2024 or 01/11/24 or 01-11-24, sometimes with / separators and sometimes with - separators and sometimes with a four digit year and sometimes with a two digit year. We can craft our regular expression to capture all these variations.

```
# get the remaining dates in #/#/# or #-#-# format
#   .*  is "greedy" and will absorb as much as possible
#   .*? is "lazy" and will stop at the first pattern match
gsub(".*?([0-9]{1,2}[/-][0-9]{1,2}[/-](20)?[1-2][0-9]).*",
     "\\1",
     a[nchar(a) > 20])
```

```
 [1] "4/6/2022"   "1-31-2022"  "10-26-21"   "10-4-2021"  "7-29-21"
 [6] "12-25-2020" "12-9-2020"  "11-27-20"   "11-12-20"   "10-26-2020"
[11] "10-8-2020"  "9-02-19"
```

We have covered all the cases and can now create a column of incident dates extracted from the incident details.

```
ois <- ois |>
   mutate(date = gsub(
      "(?xs)
  .*?(January|February|March|April|May|June|
   July|August|September|October|November|December)
  \\s* ([0-9]{1,2})
  ( , \\s* (20[0-9]{2}) )?.*",
      "\\1 \\2\\3", text, perl=TRUE),
      date = if_else(nchar(date)<20, date, NA),
      date = if_else(!is.na(date) & !grepl("20[0-9]{2}", date),
                     paste0(date, ", 20", substring(id,1,2)),
                     date),
      date = if_else(
         is.na(date),
         gsub(".*?([0-9]{1,2}[/-][0-9]{1,2}[/-](20)?[1-2][0-9])[^0-9].*",
              "\\1", text),
         date),
      date = mdy(date))
```

For a little test, let's check if there are any incidents where the year we scraped from the webpage differs from the first two digits of the incident ID.

```
table(year(ois$date), substring(ois$id,1,2))
```

```
      16 17 18 19 20 21 22 23 24 25
  2016 23  0  0  0  0  0  0  0  0  0
  2017  0 13  0  0  0  0  0  0  0  0
```

2018	0	0	12	0	0	0	0	0	0	0
2019	0	0	0	9	0	0	0	0	0	0
2020	0	0	0	0	14	0	0	0	0	0
2021	0	0	0	0	0	7	0	0	0	0
2022	0	0	0	0	0	0	15	0	0	0
2023	0	0	0	0	0	0	0	13	0	0
2024	0	0	0	0	0	0	0	0	29	0
2025	0	0	0	0	0	0	0	0	0	12

That is a good sign!

5 Geocoding the OIS locations

Our OIS data frame has the address for every incident, but to be more useful we really need the geographic coordinates. If we had the coordinates, then we could put them on a map, tabulate how many incidents occur within an area, calculate distances, and answer questions about the geography of these data.

Geocoding is the process of converting a text description of a location (typically an address or intersection) to obtain geographic coordinates (often longitude/latitude, but other coordinate systems are also possible). Google Maps currently reigns supreme in this area. Google Maps understands very general descriptions of locations. You can ask for the coordinates of something like "chipotle near UPenn" and it will understand that "UPenn" means the University of Pennsylvania and that "chipotle" is the burrito chain. Unfortunately, as of June 2018 Google Maps requires a credit card in order to access its geocoding service. Previously, anyone could geocode up to 2,500 locations per day without needing to register.

We will use the ArcGIS geocoder to get the coordinates of every location. Many web data sources use a standardized language for providing data. JSON (JavaScript Object Notation) is quite common and ArcGIS uses JSON.

The URL for ArcGIS has the form

https://geocode.arcgis.com/arcgis/rest/services/World/GeocodeServer/findAddressCandidates?f=json&singleL

You can see the address for Penn's McNeil Building embedded in this URL. Spaces need to be replaced with `%20` (the space character has ASCII code 20). Let's see what data we get back from this URL.

```
scan("https://geocode.arcgis.com/arcgis/rest/services/World/GeocodeServer/
    findAddressCandidates?f=json&singleLine=38th%20and%20Walnut,%20Philadelphia,%20
    PA&outFields=Match_addr,Addr_type",
     what="", sep="\n")
```

```
[1] "{\"spatialReference\":{\"wkid\":4326,\"latestWkid\":4326},\"candidates\
 ↪  ":[{\"address\":\"S 38th St & Walnut St, Philadelphia, Pennsylvania,
 ↪  19104\",\"location\":{\"x\":-75.198781031742,\"y\":39.953632005469},\"sc
 ↪  ore\":99.36,\"attributes\":{\"Match_addr\":\"S 38th St & Walnut St,
 ↪  Philadelphia, Pennsylvania, 19104\",\"Addr_type\":\"StreetInt\"},\"exten
 ↪  t\":{\"xmin\":-75.199781031742,\"ymin\":39.952632005469,\"xmax\":-75.197
 ↪  781031742,\"ymax\":39.954632005469}},{\"address\":\"S 38th St & Walnut
 ↪  St, Philadelphia, Pennsylvania,
 ↪  19104\",\"location\":{\"x\":-75.198651028424,\"y\":39.953613020458},\"sc
 ↪  ore\":99.36,\"attributes\":{\"Match_addr\":\"S 38th St & Walnut St,
 ↪  Philadelphia, Pennsylvania, 19104\",\"Addr_type\":\"StreetInt\"},\"exten
 ↪  t\":{\"xmin\":-75.199651028424,\"ymin\":39.952613020458,\"xmax\":-75.197
 ↪  651028424,\"ymax\":39.954613020458}}]}"
```

It is messy, but readable. You can see embedded in this text the `lat` and `lon` for this address. You can also see that it should not be too hard for a machine to extract these coordinates, and the rest of the information here, from this block of text. This is the point of JSON, producing data in a format that a human could understand in a small batch, but a machine could process fast and easily.

The `jsonlite` R package facilitates the conversion of JSON text like this into convenient R objects.

```
library(jsonlite)
fromJSON("https://geocode.arcgis.com/arcgis/rest/services/World/GeocodeServer/
    findAddressCandidates?f=json&singleLine=38th%20and%20Walnut,%20Philadelphia,%20
    PA&outFields=Match_addr,Addr_type")
```

```
$spatialReference
$spatialReference$wkid
[1] 4326

$spatialReference$latestWkid
[1] 4326


$candidates
                                                address location.x
1 S 38th St & Walnut St, Philadelphia, Pennsylvania, 19104  -75.19878
2 S 38th St & Walnut St, Philadelphia, Pennsylvania, 19104  -75.19865
  location.y score                                    attributes.Match_addr
1   39.95363 99.36 S 38th St & Walnut St, Philadelphia, Pennsylvania, 19104
2   39.95361 99.36 S 38th St & Walnut St, Philadelphia, Pennsylvania, 19104
  attributes.Addr_type extent.xmin extent.ymin extent.xmax extent.ymax
1            StreetInt   -75.19978    39.95263   -75.19778    39.95463
```

```
2            StreetInt   -75.19965    39.95261   -75.19765    39.95461
```

`fromJSON()` converts the JSON results from the ArcGIS geocoder to an R `list` object. The JSON tags turn into list names and columns in a data frame.

To make geocoding a little more convenient, here is an R function that automates the process of taking an address, filling in special characters (like spaces) with their ASCII codes with `URLencode()`, and retrieving the JSON results from the ArcGIS geocoding service.

```
geocodeARCGIS <- function(address)
{
   paste0("https://geocode.arcgis.com/arcgis/rest/services/World/GeocodeServer/
    findAddressCandidates?f=json&singleLine=",
          URLencode(address),
          "&outFields=Match_addr,Addr_type") |>
      fromJSON()
}
```

Let's test out `geocodeARCGIS()` by pulling up a map of the geocoded coordinates. Once we have the latitude and longitude for the McNeil Building, where we typically hold our crime data science courses at Penn, we can use `leaflet()` to show us a map of the area.

```
gcPenn <- geocodeARCGIS("3718 Locust Walk, Philadelphia, PA") |>
   pluck("candidates") |>
   head(1) |>
   mutate(lon = location$x,
          lat = location$y)

leaflet(width = 1200, height = 800) |>
   # addTiles() |>
    addTiles(
       urlTemplate = "https://api.maptiler.com/maps/streets/{z}/{x}/{y}.png?key={
    key}",
       options = tileOptions(key = Sys.getenv("MAPTILER_API_KEY"),
                             tileSize = 512,
                             zoomOffset = -1),
       attribution = '© MapTiler © OpenStreetMap contributors') |>
   setView(lng=gcPenn$lon, lat=gcPenn$lat, zoom=18) |>
   addCircleMarkers(lng=gcPenn$lon,
                    lat=gcPenn$lat)
```

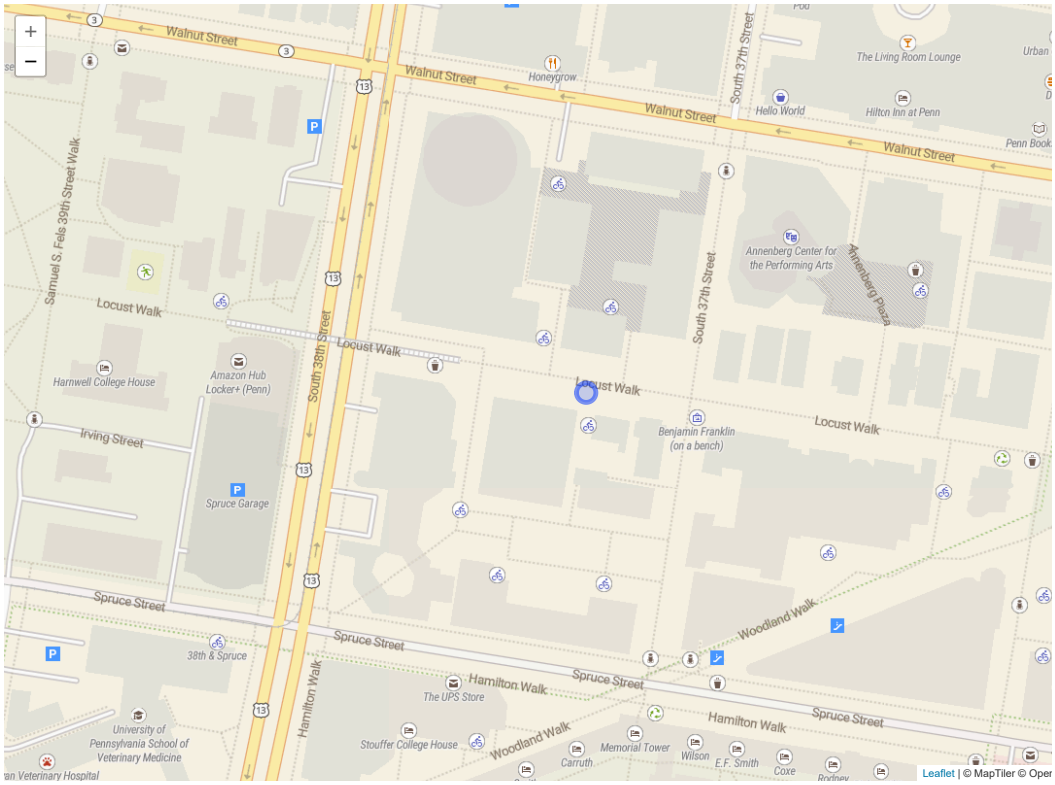

Figure 3: ArcGIS geocoding result for 3718 Locust Walk

`leaflet()` prepares the mapping process (the `width` and `height` do not have to be set, by I needed to for these notes). `addTiles()` pulls in the relevant map image (buildings and streets). Generally, you can use `addTiles()` with no other arguments. I regularly recompile these notes which hits the OpenStreetMap server a little too much, so I have to rely on a different map provider that can better handle the load of requests. `setView()` takes the longitude and latitude from our `gcPenn` object, sets that as the center of the map, and zooms in to level "18," which is a fairly close zoom of about one block. `addCircleMarkers()` creates a circle at the selected point.

We are almost ready to throw all of our addresses at the geocoder, but let's first make sure the addresses look okay. Several locations have `&` where the ArcGIS geocoder wants `and`.

```
# & → and for geocoding
grep('&', ois$location, value=TRUE)
```

```
[1] "Jasper Street & Hart Lane"           "Front Street & Allegheny Avenue"
[3] "Bridge Street & Roosevelt Boulevard" "49th & Walnut Streets"
[5] "4800, 4900 & 5100 blocks of Sansom"  "near 16th Street & Allegheny Ave"
```

Many addresses just give the block, like "3700 block of Locust Walk." We will need to change these to the midpoint of the block like "3750 Locust Walk" so that we get a geocoding hit that is nearby.

```
grep("[Bb]lock", ois$location, value=TRUE)
```

```
 [1] "2600 block of South 21st Street"
 [2] "100 block of West Somerset Street"
 [3] "300 block of North 65th Street"
 [4] "4600 block of Roosevelt Blvd"
 [5] "4100 block of Ogden Street"
 [6] "4600 block of Roosevelt Boulevard"
 [7] "1600 block of Moore Street"
 [8] "2800 block of Jasper Street"
 [9] "4100 block of Leidy Avenue"
[10] "800 block of West Master Street"
[11] "600 block of Chamounix Drive"
[12] "3400 block of Vista Street"
[13] "3200 block of A Street"
[14] "5400 block of Chancellor Street"
[15] "2900 block of E. Street"
[16] "3300 block of Willits Road"
[17] "6100 block of Lebanon Avenue"
[18] "2600 block of Glenwood Avenue"
[19] "3900 block of Whittaker Avenue"
[20] "2200 block of S. 65th Street"
```

```
[21] "3000 block of Ruth Street"
[22] "6100 block of West Columbia Avenue"
[23] "3500 block of F Street"
[24] "2700 block of North 6th Street"
[25] "1500 block of North 57th Street"
[26] "2100 block of East Westmoreland Street"
[27] "3000 block of North 16th Street"
[28] "1500 block of 58th Street"
[29] "2200 block of West Oxford Avenue"
[30] "3900 block of Fairmont Avenue"
[31] "Unit block of East Cliveden Street"
[32] "2100 block of Eastburn Avenue"
[33] "1000 block of West Dakota Street"
[34] "6200 block of Haverford Avenue"
[35] "1000 block of North 48th Street"
[36] "300 block of Adams Avenue"
[37] "2800 block of Mascher Street"
[38] "2300 block of Borbeck Avenue"
[39] "3600 block of Sepviva Street"
[40] "1800 block of South 29th Street"
[41] "8000 block of N. Frankford Ave."
[42] "3700 block of Fairmount Street"
[43] "7500 block of Whitaker Avenue"
[44] "1500 block of N. 62nd Street"
[45] "Unit block of E. Phil Ellena St."
[46] "7600 block of Lexington Avenue"
[47] "3100 block of Emerald Street"
[48] "100 block of E. Willard St."
[49] "2300 block of Fawn St."
[50] "400 block of W. Bringhurst St."
[51] "2700 block of Brown Street,"
[52] "3000 block of N. Water Street"
[53] "300 block of Glen Echo Road"
[54] "2500 Block of south 7th Street"
[55] "1500 Block of Bailey Street"
[56] "7600 Block of Roosevelt Blvd"
[57] "5900 block of Torresdale Avenue"
[58] "1000 block of Chelten Avenue"
[59] "3400 block of G Street"
[60] "1800 block of N. Broad Street"
[61] "2100 block of Taney Terrace"
[62] "1300 block of Kater Street"
[63] "4900 block of Hazel Avenue"
```

```
 [64] "7500 block of Brookhaven Road"
 [65] "8700 block of Crispin Street"
 [66] "3200 block of G Street"
 [67] "2700 block of Dickinson Street"
 [68] "7100 block of Hegerman Street"
 [69] "2000 block of Snyder Avenue"
 [70] "4800 block of Knox Street"
 [71] "1400 block of Lardner Street"
 [72] "3100 block of N. 33rd Street"
 [73] "1300 block of Bigler Street"
 [74] "2800 block of Kensington Avenue"
 [75] "1200 block of S. 29th Street"
 [76] "2300 block of north 13th Street"
 [77] "2500 Block of N. Alder Street"
 [78] "3100 Block of N. Darien Street"
 [79] "2200 Block of N. Fairhill Street"
 [80] "8300 Block of Horrocks Street"
 [81] "400 block of East Somerset Street"
 [82] "4200 block of Whitaker Avenue"
 [83] "1200 block of south 51st street"
 [84] "6400 block of Lambert Street"
 [85] "2900 Block of Amber Street"
 [86] "3100 block of N. 9th Street"
 [87] "2900 block of N. Front Street"
 [88] "3800 block of Elsinore Street"
 [89] "500 block of N. 56 Street"
 [90] "6300 block of Crafton Street"
 [91] "1400 block of Chew Avenue"
 [92] "6300 block of Hazel Avenue"
 [93] "4800, 4900 & 5100 blocks of Sansom"
 [94] "200 block of Millick Street"
 [95] "600 block of E. Clearfield Street"
 [96] "3400 block of Broad street"
 [97] "6300 Block of Overbrook Avenue"
 [98] "5300 Block of Grays Avenue"
 [99] "Unit Block of Salford street"
[100] "3100 Block of north Carlisle Street"
[101] "100 block of north 55th Street"
[102] "300 block of south 60th street"
```

Lastly, there are some addresses with the word "near" that need to be deleted.

```
grep("[Nn]ear", ois$location, value=TRUE)
```

```
[1] "near 16th Street & Allegheny Ave" "Near Loudon and D streets"
```

Let's make all these fixes.

```
ois <- ois |>
   mutate(location = gsub("&","and",location),
          location = gsub("00 [Bb]lock( of)?", "50", location),
          location = gsub("[Uu]nit [Bb]lock( of)?", "50", location),
          location = gsub("[Nn]ear ", "", location))
```

Some OIS incidents are missing locations.

```
ois |>
  filter(grepl("[Ww]ithheld", location)) |>
  pull(text)
```

```
[1] "PS# 16-26\n9/05/16\nOn Monday, September 5, 2016, at approximately 6:28
↪  P.M., an off-duty officer, in plainclothes, became involved in a verbal
↪  and physical altercation with his son at their residence. During the
↪  physical altercation the officer discharged his personal weapon, striking
↪  his son.\nThe officer's son was transported to Aria-Torresdale Hospital
↪  for treatment.\nThe officer's firearm, a .40 caliber semi-automatic
↪  pistol, loaded with three live rounds, was recovered at the scene.\nThere
↪  were no other injuries as a result of this incident.\n*** Information
↪  posted in the original summary reflects a preliminary understanding of
↪  what occurred at the time of the incident. This information is posted
↪  shortly after the incident and may be updated as the investigation leads
↪  to new information. The DA's Office is provided all the information from
↪  the PPD's investigation prior to their charging decision."
```

```
[2] "PS#16-18\n5/31/16\nOn Tuesday, May 31, 2016, at approximately 1:12 PM,
↪ an off-duty officer, in civilian attire, arrived home at his residence.
↪ Upon entering the front door, the officer observed the living room
↪ television missing, the rear kitchen window open, and the rear kitchen
↪ door ajar. The officer heard voices coming from the basement. The officer
↪ went to the top of the basement steps and announced, "Police." A male
↪ appeared at the bottom of the steps and charged up the steps toward the
↪ officer with a dark metal object in his hand. In response, the officer
↪ discharged his weapon one time missing the offender. The offender fell
↪ down the steps crashing into the basement wall. The offender along with a
↪ second offender that was also in the basement fled out the rear basement
↪ door with the officer in foot pursuit. Upon exiting the rear basement
↪ door, the officer observed a green/blue van pull away from his property.
↪ The officer apprehended one offender in the 3200 block of Wellington
↪ Street. An off-duty detective apprehended the other offender near
↪ Brighton and Hawthorne Streets.\nThere were no reported injuries as a
↪ result of this police firearm discharge.\nNo weapon was recovered.\n***
↪ Information posted in the original summary reflects a preliminary
↪ understanding of what occurred at the time of the incident. This
↪ information is posted shortly after the incident and may be updated as
↪ the investigation leads to new information. The DA's Office is provided
↪ all the information from the PPD's investigation prior to their charging
↪ decision."
```

The text of OIS 16-18 gives a nearby address (3250 Wellington Street). We will drop incident 16-26, since it is not really a police shooting.

Several other incidents have quirky addresses.

```
ois |>
  filter(id %in% c("16-30","16-10","17-08")) |>
  select(id, location, text)
```

```
     id                                                    location
1 17-08 New Castle County (see summary for details of PPD involvement)
2 16-30                                  4800, 4900 and 5150s of Sansom
3 16-10                       5700 N. Park street/5700 N. Broad street
```

```
1 OIS# 17-08 (March 29, 2017)\nOn Wednesday, March 29, 2017, at approximately
 ↪ 5:39 PM, two uniformed officers in a marked vehicle responded to a radio
 ↪ assignment of "Person with a gun" at 5600 Whitby Avenue. A description of
 ↪ the individual was broadcast. Upon arrival at the location both officers
 ↪ observed a male who met the description entering the driver's seat of a
 ↪ parked minivan. As the male approached the open driver's door area both
 ↪ officers instructed him to stop. The male instead sat in the driver seat.
 ↪ The male accelerated the vehicle rearward striking officer number one,
 ↪ knocking him to the ground. The officer got back on his feet observed the
 ↪ male driver reach under the driver's seat (through the open drivers side
 ↪ door), and officer number one discharged one round at the male. The male
 ↪ drove from the location, struck a vehicle at 5700 Woodland Avenue, and
 ↪ continued toward Island Avenue and Lindbergh Boulevard where the minivan
 ↪ became disabled. The male exited the minivan, entered an unoccupied
 ↪ vehicle that was nearby with the engine running and the keys in the
 ↪ ignition. He ushered five passengers from the minivan into the stolen
 ↪ vehicle. The male drove the stolen vehicle to a house in New Castle
 ↪ Delaware, and dropped off four of the passengers. The male and a
 ↪ remaining female passenger drove to a second house in New Castle Delaware
 ↪ in the stolen vehicle. New Castle County Police Officers responded to the
 ↪ second location and report to Philadelphia Police that New Castle County
 ↪ Police Officers attempted to arrest the male who was inside the stolen
 ↪ vehicle. A uniformed New Castle County officer discharged his firearm
 ↪ multiple times, striking the male. The male was pronounced deceased at
 ↪ Christiana Hospital.\nThere were no other injuries as a result of this
 ↪ incident. No firearm was recovered from the male or the vehicles. The
 ↪ female was not charged with any offense.\n*** Information posted in the
 ↪ original summary reflects a preliminary understanding of what occurred at
 ↪ the time of the incident. This information is posted shortly after the
 ↪ incident and may be updated as the investigation leads to new
 ↪ information. The DA's Office is provided all the information from the
 ↪ PPD's investigation prior to their charging decision.
```

```
2                          PS#16-30\n9/16/16\nOn Friday, September 16, 2016,
↪ at approximately 11:18 P.M., a uniformed sergeant in a marked police
↪ vehicle was seated in her parked vehicle in the 5100 block of Sansom
↪ Street, when a male approached and without warning, began to discharge a
↪ firearm, striking the sergeant, as she remained seated in her vehicle.
↪ The offender then began walking east on Sansom Street, stopping at a
↪ lounge/bar in the 5100 block of Sansom Street, where he discharged his
↪ firearm into the lounge/bar, striking a female employee and a male
↪ security guard. The offender continued walking east on Sansom Street to
↪ the 4900 block, where he discharged his firearm into an occupied parked
↪ vehicle, striking one female and one male occupant.\nResponding uniformed
↪ officers, in marked police vehicles, along with an officer from the
↪ University of Pennsylvania police force, located the offender in an
↪ alleyway in the rear of the 4800 blocks of Sansom and Walnut Streets.
↪ While in the 4800 block of Sansom Street the offender discharged his
↪ firearm, striking the University of Pennsylvania Officer as well as a
↪ marked police vehicle. Four Officers (one of whom was the University of
↪ Pennsylvania Officer) discharged their firearms, striking the offender.
↪ The offender fell to the ground and dropped his firearm. Fire Rescue
↪ responded and pronounced the offender deceased.\nThe offender's firearm,
↪ a 9MM, semi-automatic pistol, with an obliterated serial number, loaded
↪ with 14 live rounds, was recovered at the scene. There were three empty
↪ magazines from the offender's firearm recovered throughout the
↪ scene.\nThe sergeant, the University of Pennsylvania Officer, along with
↪ the four civilians who were all struck by gunfire, were transported to
↪ Penn-Presbyterian Hospital for treatment.\nThe female from the parked
↪ vehicle was later pronounced deceased at Penn-Presbyterian Hospital.\n***
↪ Information posted in the original summary reflects a preliminary
↪ understanding of what occurred at the time of the incident. This
↪ information is posted shortly after the incident and may be updated as
↪ the investigation leads to new information. The DA's Office is provided
↪ all the information from the PPD's investigation prior to their charging
↪ decision.
```

```
3
 ↪ PS#16-10\n4/09/16\nOn Saturday, April 9, 2016, at approximately 2:26
 ↪ P.M., an off-duty police officer, in plainclothes, observed two males
 ↪ involved in a physical struggle on the highway, in the 5700 block of N.
 ↪ Park Avenue. The officer observed that one of the males was armed with a
 ↪ handgun. The officer, who was armed, identified himself as a police
 ↪ officer and ordered the offender to drop his gun. The offender turned
 ↪ toward the officer and pointed the firearm at him. In response, the
 ↪ officer discharged his weapon at the offender. The offender fled to a
 ↪ rear parking lot in the 5700 block of N. Broad Street, where he fell to
 ↪ the ground, dropping his weapon. The offender retrieved his weapon and
 ↪ again pointed it at the officer. The officer responded by discharging his
 ↪ weapon, striking the offender. The offender again fled but collapsed near
 ↪ the corner of Broad and Chew Streets, where he was arrested.\nThe
 ↪ offenders' firearm, a .22 caliber revolver, loaded with three spent
 ↪ casings, was recovered at the scene.\nThe offender was transported to
 ↪ Albert Einstein Medical Center, where he was later pronounced deceased as
 ↪ a result of his injuries.\nThere were no other reported injuries as a
 ↪ result of this incident.\n*** Information posted in the original summary
 ↪ reflects a preliminary understanding of what occurred at the time of the
 ↪ incident. This information is posted shortly after the incident and may
 ↪ be updated as the investigation leads to new information. The DA's Office
 ↪ is provided all the information from the PPD's investigation prior to
 ↪ their charging decision.
```

Reading the details of the incidents, we can come up with reasonable fixes to the addresses. OIS 17-08 is not a PPD shooting incident. Some PPD officers were present when New Castle County (Delaware) police officers shot someone. Let's drop this incident. The remaining incidents we can edit based on the contents of the OIS description. We will also tack on ", Philadelphia, PA" to the end of each location to improve geocoding accuracy.

```
ois <- ois |>
  filter(id != "17-08" &    # not a PPD shooting
         id != "16-26") |>  # not really a police shooting
  mutate(location = case_match(id,
           "16-18" ~ "3250 Wellington Street",
           # two locations, let's use the first one
           "16-10" ~ "5750 N. Broad Street",
           # pick the location where the police shooting occurred
           "16-30" ~ "4850 Sansom Street",
           .default=location),
         location = # add the city
           paste0(location,", Philadelphia, PA"))
```

Let's test out the process for just the first location. The code here shows how you can extract

each bit of information that we want from geocoding an address: the coordinates (long,lat), the specific address that the geocoding service translated our requested address to, a quality-of-match score, and location type (e.g. StreetInt, PointAddress, StreetAddress).

```
a <- geocodeARCGIS(ois$location[1])
# collect (long,lat), matched address, address match score, and location type
a$candidates$location$x[1]
```

```
[1] -75.18278
```

```
a$candidates$location$y[1]
```

```
[1] 39.91872
```

```
a$candidates$address[1]
```

```
[1] "2650 S 21st St, Philadelphia, Pennsylvania, 19145"
```

```
a$candidates$score[1]
```

```
[1] 100
```

```
a$candidates$attributes$Addr_type[1]
```

```
[1] "StreetAddress"
```

With that we are ready to run all of our addresses through the ArcGIS geocoder. We could have geocoded all these addresses with the more simple code `lapply(ois$location, geocodeARCGIS)`. However, if the JSON connection to the geocoder fails for even one of the addresses (likely if you have a poor internet connection), then the whole `lapply()` function fails. With the for-loop implementation, if the connection fails, then `ois` still keeps all of the prior geocoding results and you can restart the for-loop at the point where it failed.

```
# takes about 3 minutes
geoInfo <- foreach(i=1:nrow(ois), .combine=bind_rows) %do%
{
  # message(paste0("#", i, " Address: ", ois$location[i]))
  a <- geocodeARCGIS(ois$location[i])

  data.frame(lon       = a$candidates$location$x[1],
             lat       = a$candidates$location$y[1],
             addrmatch = a$candidates$address[1],
             score     = a$candidates$score[1],
             addrtype  = a$candidates$attributes$Addr_type[1])
}
ois <- ois |> bind_cols(geoInfo)
```

Now we should have longitude and latitude for every incident. Let's check that they all look sensible.

```
stem(ois$lat)
```

```
The decimal point is 2 digit(s) to the left of the |

3987 | 7
3988 |
3989 |
3990 |
3991 | 4579
3992 | 4566779
3993 | 02344588
3994 | 2239
3995 | 14667788
3996 | 3345566999
3997 | 11234566669
3998 | 112558999
3999 | 011111122233344455666899
4000 | 000011233567
4001 | 1146678
4002 | 12455888
4003 | 0033558
4004 | 000123889
4005 | 012233445
4006 | 226
4007 | 1
4008 | 0
```

```
stem(ois$lon)
```

```
The decimal point is 2 digit(s) to the left of the |

-7526 | 7
-7524 | 00885444432000
-7522 | 75321030
-7520 | 98769873300
-7518 | 754221194432221
-7516 | 9766633322555444210
-7514 | 87776654443221099666554321
-7512 | 221099873111100
```

```
  -7510 | 98776655521987400
  -7508 | 9786
  -7506 | 9360
  -7504 | 19977
  -7502 | 977
  -7500 | 39
  -7498 | 9
```

All the points have latitude around 39 and 40 and longitude around -75. That is a good sign!

Let's check the "address type". We should worry about addresses geocode to a "StreetName." That means the incident got geocoded to, say, "Market Street" but we are not sure where along Market Street the incident actually occurred. The geocoder most likely placed the incident at the midpoint of the street.

```
ois |> count(addrtype)
```

```
          addrtype  n
1     PointAddress 55
2    StreetAddress 75
3 StreetAddressExt  2
4        StreetInt 10
5       StreetName  3
```

```
ois |>
   filter(addrtype=="StreetName") |>
   select(id, location, addrmatch)
```

```
     id                                                    location
1 25-05 1 Philadelphia International Airport Way, Philadelphia, PA
2 25-01                       650 Chamounix Drive, Philadelphia, PA
3 21-14                      3800 Landsowne Drive, Philadelphia, PA
                                                            addrmatch
1 Philadelphia International Airport, Philadelphia, Pennsylvania, 19153
2                       Chamounix Dr, Philadelphia, Pennsylvania, 19131
3                       Lansdowne Dr, Philadelphia, Pennsylvania, 19104
```

One address should be at the Philadelphia Airport. This one geocoded just fine.

```
ois |>
  filter(id=="25-05") |>
  leaflet(width = 1200, height = 800) |>
    addTiles(
       urlTemplate = "https://api.maptiler.com/maps/streets/{z}/{x}/{y}.png?key={
    key}",
       options = tileOptions(key = Sys.getenv("MAPTILER_API_KEY"),
                             tileSize = 512,
```

```
                              zoomOffset = -1),
     attribution = '© MapTiler © OpenStreetMap contributors') |>
addCircleMarkers(~lon, ~lat,
                 radius=3, stroke=FALSE,
                 fillOpacity = 1) |>
addPopups(~lon, ~lat, ~location)
```

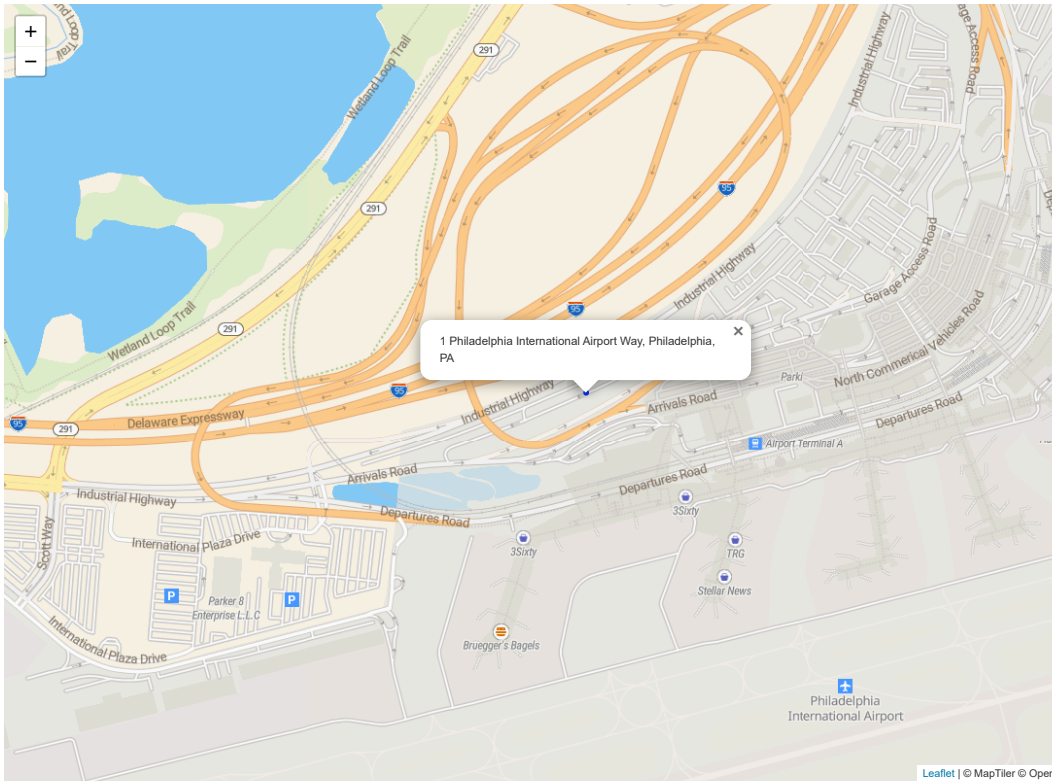

Figure 4: Checking the location of OIS 25-05

OIS 25-01 involves a dog shooting in Fairmount Park. The news stories about the incident place it 1 mile from Belmont/Edgely, close to Chamounix Drive and Ford Ave.

```
ois |>
   filter(id=="25-01") |>
   leaflet(width = 1200, height = 800) |>
   addTiles(
      urlTemplate = "https://api.maptiler.com/maps/streets/{z}/{x}/{y}.png?key={key
    }",
      options = tileOptions(key = Sys.getenv("MAPTILER_API_KEY"),
                            tileSize = 512,
                            zoomOffset = -1),
      attribution = '© MapTiler © OpenStreetMap contributors') |>
   addCircleMarkers(~lon, ~lat,
                    radius=3, stroke=FALSE,
                    fillOpacity = 1) |>
   addPopups(~lon, ~lat, ~location)
```

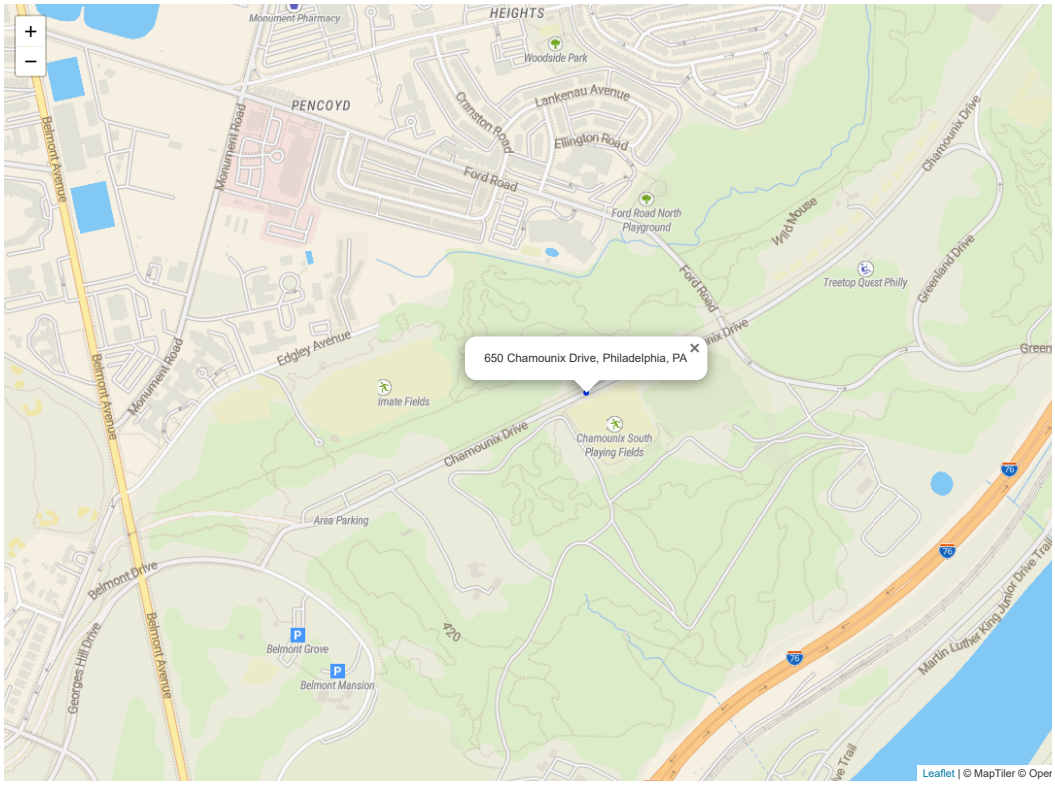

Figure 5: Checking the location of OIS 25-01

```
geocodeARCGIS("Chamounix Drive and Ford Ave, Philadelphia, PA")
```

```
$spatialReference
$spatialReference$wkid
[1] 4326

$spatialReference$latestWkid
[1] 4326


$candidates
                                                    address location.x
1 Chamounix Dr & Ford Rd, Philadelphia, Pennsylvania, 19131  -75.20474
2           Chamounix Dr, Philadelphia, Pennsylvania, 19131  -75.20779
  location.y score                                    attributes.Match_addr
1   39.99711 99.59 Chamounix Dr & Ford Rd, Philadelphia, Pennsylvania, 19131
2   39.99607 90.79           Chamounix Dr, Philadelphia, Pennsylvania, 19131
  attributes.Addr_type extent.xmin extent.ymin extent.xmax extent.ymax
1            StreetInt   -75.20574    39.99611   -75.20374    39.99811
2           StreetName   -75.20879    39.99507   -75.20679    39.99707
```

OIS 21-14 has address "3800 Landsowne Drive". Presumably it intended to find 3800 Lansdowne Drive, but it could not place the 3800 block on Lansdowne Drive. The text describes the incident as occurring behind a school. Let's zoom in and see where this might have occurred. It must have occurred behind the School of the Future. I used Google Maps to find the coordinates behind this school.

```
ois |>
   filter(id=="21-14") |>
   leaflet(width = 1200, height = 800) |>
   addTiles(
      urlTemplate = "https://api.maptiler.com/maps/streets/{z}/{x}/{y}.png?key={key
    }",
      options = tileOptions(key = Sys.getenv("MAPTILER_API_KEY"),
                            tileSize = 512,
                            zoomOffset = -1),
      attribution = '© MapTiler © OpenStreetMap contributors') |>
   addCircleMarkers(~lon, ~lat,
                    radius=3, stroke=FALSE,
                    fillOpacity = 1) |>
   addPopups(~lon, ~lat, ~location)
```

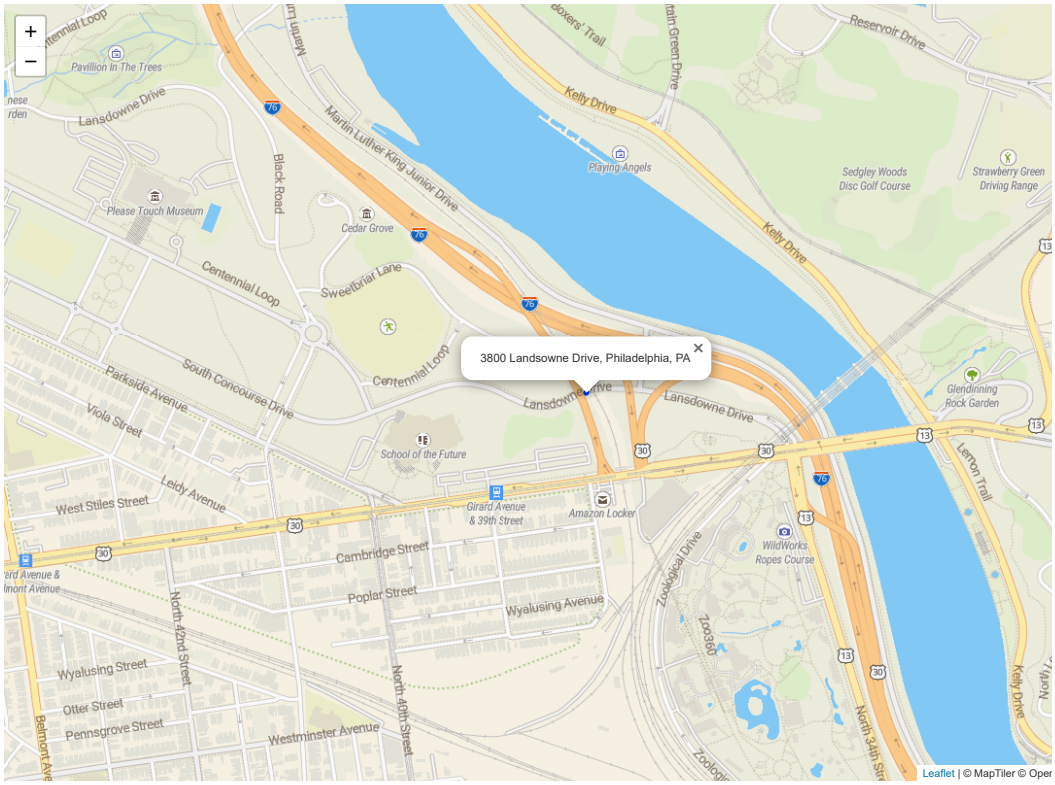

Figure 6: Checking the location of OIS 21-14

Let's record these fixes.

```
ois <- ois |>
  mutate(lat = case_match(id,
                          "21-14" ~ 39.975984,
                          "25-01" ~ 39.99711,
                          .default = lat),
         lon = case_match(id,
                          "21-14" ~ -75.203309,
                          "25-01" ~ -75.20474,
                          .default = lon))
```

Here's a map of all of the incidents. For each incident I have added some pop-up text so that if you click on an incident it will show you the location of the incident and the text describing the incident.

```
ois |>
   leaflet(width = 1200, height = 800) |>
   addTiles(
      urlTemplate = "https://api.maptiler.com/maps/streets/{z}/{x}/{y}.png?key={key
    }",
      options = tileOptions(key = Sys.getenv("MAPTILER_API_KEY"),
                            tileSize = 512,
                            zoomOffset = -1),
      attribution = '© MapTiler © OpenStreetMap contributors') |>
   addCircleMarkers(~lon, ~lat,
                    radius=4, stroke=FALSE,
                    fillOpacity = 1,
                    popup = paste("<b>",ois$location,"</b><br>",ois$text),
                    popupOptions = popupOptions(autoClose = TRUE,
                                                closeOnClick = FALSE))
```

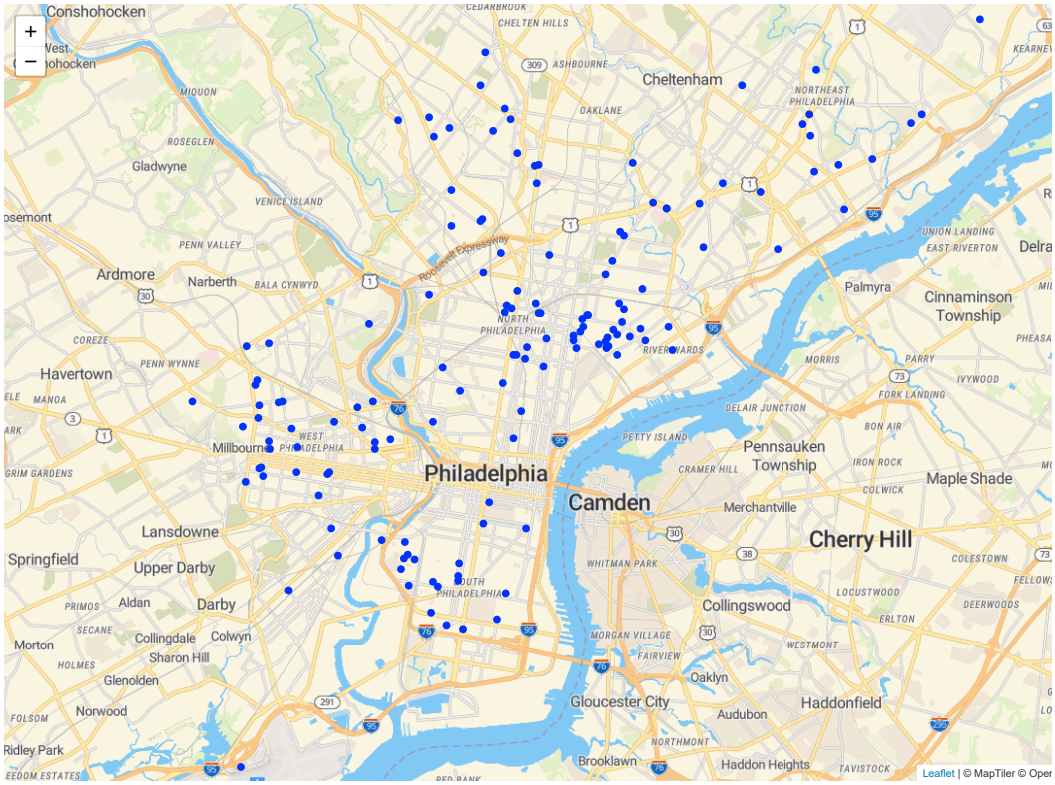

Figure 7: All Philadelphia Officer-involved Shootings

6 Working with shapefiles and coordinate systems

The Philadelphia Police Department divides the city into Police Service Areas (PSAs). The city provides a *shapefile*, a file containing geographic data, that describes the boundaries of the PSAs at Philadelphia's open data site. R can read these files using the `st_read()` function provided in the `sf` (simple features) package. Even though `st_read()` appears to only be accessing `Boundaries_PSA.shp`, you should have all of the `Boundaries_PSA` files in your `10_shapefiles_and_data` folder. The other files have information that `st_read()` needs, like the coordinate system stored in `Boundaries_PSA.prj`. If you do not have all `Boundaries_PSA` files in your folder, then in a few lines you will get errors like "the sfc object should have crs set," meaning that the Coordinate Reference System (CRS) is missing.

```
library(sf)
PPDmap <- st_read("10_shapefiles_and_data/Boundaries_PSA.shp")
```

```
Reading layer `Boundaries_PSA' from data source
  `C:\R4crim\10_shapefiles_and_data\Boundaries_PSA.shp' using driver `ESRI
   ↪ Shapefile'
Simple feature collection with 66 features and 10 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -75.28031 ymin: 39.86701 xmax: -74.95575 ymax: 40.13793
Geodetic CRS:  WGS 84
```

You can also get the same PSA boundaries using geoJSON.

```
library(geojsonsf)
PPDmap <- geojson_sf("https://opendata.arcgis.com/datasets/8
    dc58605f9dd484295c7d065694cdc0f_0.geojson")
```

`PPDmap` is an `sf` (simple features) object. It is not unlike a data frame, but it contains a special `geometry` column containing geographic information associated with a row of data. Here are the two columns in `PPDmap` that are of primary interest.

```
PPDmap |> select(PSA_NUM, geometry)
```

```
Simple feature collection with 65 features and 1 field
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -75.28031 ymin: 39.86701 xmax: -74.95575 ymax: 40.13793
Geodetic CRS:  WGS 84
First 10 features:
   PSA_NUM                       geometry
1      243 POLYGON ((-75.11026 39.9929...
2      144 POLYGON ((-75.16339 40.0769...
```

```
3      151 POLYGON ((-75.0572 39.99865...
4      193 POLYGON ((-75.23857 39.9761...
5      012 POLYGON ((-75.17143 39.9167...
6      053 POLYGON ((-75.21072 40.0413...
7      391 POLYGON ((-75.15955 40.0239...
8      351 POLYGON ((-75.11112 40.0310...
9      072 POLYGON ((-75.04012 40.0686...
10     082 POLYGON ((-74.98683 40.0387...
```

The first column shows the PSA number and the second column shows a truncated description of the geometry associated with this row. In this case, `geometry` contains the coordinates of the boundary of the PSA for each row. Use `st_geometry()` to extract these coordinates.

```
plot(st_geometry(PPDmap))
axis(side=1, cex.axis=0.7) # add x-axis
axis(side=2, cex.axis=0.7) # add y-axis
# extract the center points of each PSA
a <- st_coordinates(st_centroid(st_geometry(PPDmap)))
# add the PSA number to the plot
text(a[,1], a[,2], PPDmap$PSA_NUM, cex=0.5)
```

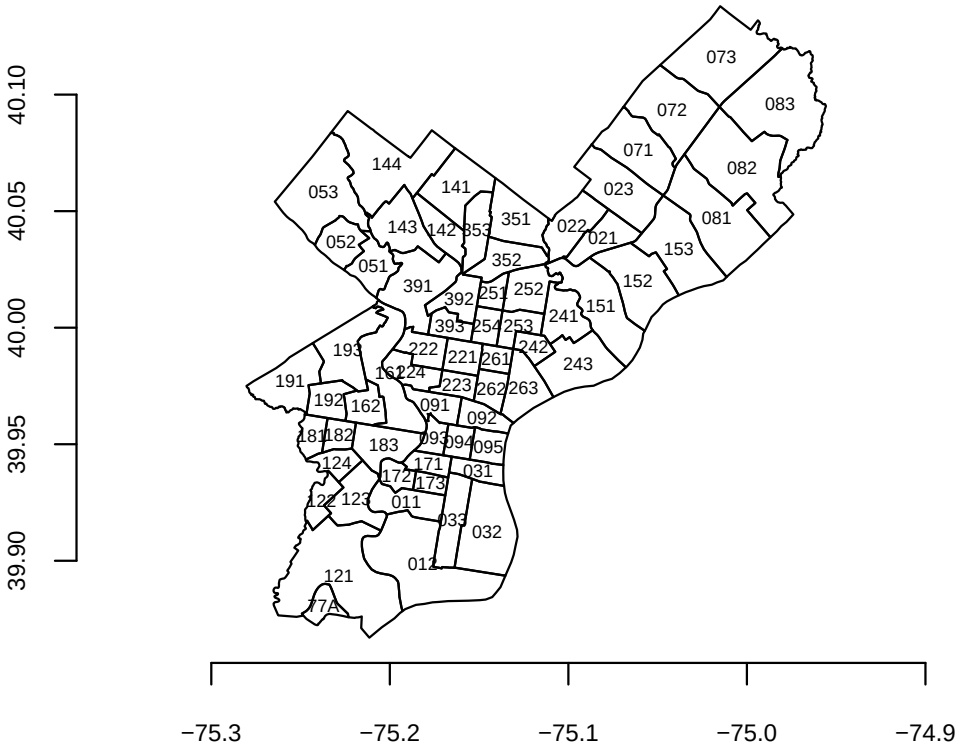

Figure 8: Map of Philadelphia Police Service Areas

We can extract the actual coordinates of one of the polygons if we wish.

```
a <- st_coordinates(PPDmap$geometry[1])
head(a)
```

```
             X        Y L1 L2
[1,] -75.11026 39.99297  1  1
[2,] -75.10992 39.99314  1  1
[3,] -75.10945 39.99337  1  1
[4,] -75.10866 39.99375  1  1
```

```
[5,] -75.10780 39.99421  1  1
[6,] -75.10691 39.99470  1  1
```

```
tail(a)
```

```
               X        Y L1 L2
[139,] -75.10905 39.99158  1  1
[140,] -75.10922 39.99178  1  1
[141,] -75.10934 39.99192  1  1
[142,] -75.10953 39.99213  1  1
[143,] -75.10971 39.99234  1  1
[144,] -75.11026 39.99297  1  1
```

And we can use those coordinates to add additional features to our plot

```
plot(st_geometry(PPDmap))
axis(side=1, cex.axis=0.7)
axis(side=2, cex.axis=0.7)
a <- st_coordinates(st_centroid(st_geometry(PPDmap)))
text(a[,1], a[,2], PPDmap$PSA_NUM, cex=0.5)
a <- st_coordinates(PPDmap$geometry[1])
lines(a[,1], a[,2], col="red", lwd=3)
```

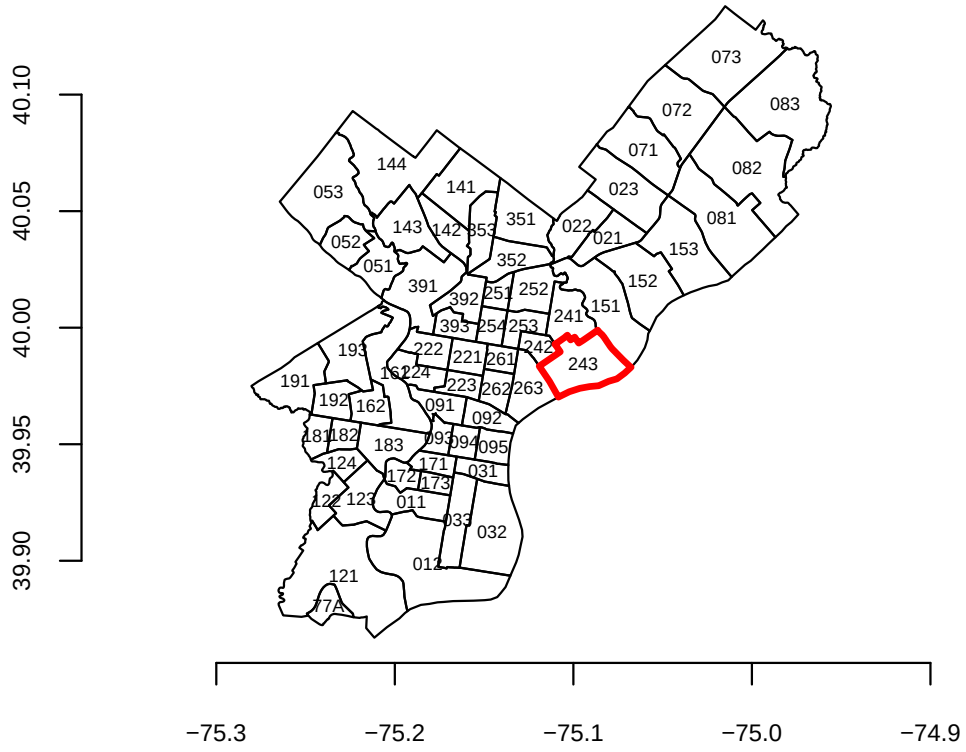

Figure 9: Map of Philadelphia Police Service Areas highlighting the first PSA in `PPDmap`

So this highlighted in red PSA 243.

We can also overlay a leaflet map with the `PPDmap` object.

```
PPDmap |>
   leaflet(width = 1200, height = 800) |>
   addPolygons(weight=1, label=~PSA_NUM) |>
   addTiles(
      urlTemplate = "https://api.maptiler.com/maps/streets/{z}/{x}/{y}.png?key={key
    }",
      options = tileOptions(key = Sys.getenv("MAPTILER_API_KEY"),
```

```
                            tileSize = 512,
                            zoomOffset = -1),
    attribution = '© MapTiler © OpenStreetMap contributors')
```

Figure 10: Map layer with the Philadelphia Police Service Areas (PSA)

6.1 Coordinate systems

Geographic datasets that describe locations on the surface of the earth have a "coordinate reference system" (CRS). Let's extract the CRS for `PPDmap`.

```
st_crs(PPDmap)
```

```
Coordinate Reference System:
  User input: 4326
  wkt:
GEOGCS["WGS 84",
      DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
          AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
      PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
      UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
      AXIS["Latitude",NORTH],
      AXIS["Longitude",EAST],
    AUTHORITY["EPSG","4326"]]
```

The coordinate system used to describe the PPD boundaries is the World Geodetic System 1984 (WGS84) maintained by the United States National Geospatial-Intelligence Agency, one of several standards to aid in navigation and geography. The European Petroleum Survey Group (EPSG) maintains a catalog of different coordinate systems (should be no surprise that oil exploration has driven the development of high quality geolocation standards). They have assigned the standard longitude/latitude coordinate system to be EPSG4326. You can find the full collection of coordinate systems at spatialreference.org. You can see in the output above a reference to EPSG 4326.

Many of us are comfortable with the longitude/latitude angular coordinate systems. However, the distance covered by a degree of longitude shrinks as you move towards the poles and only equals the distance covered by a degree of latitude at the equator. In addition, the earth is not very spherical so the coordinate system used for computing distances on the earth surface might need to depend on where you are on the earth surface.

Almost all web mapping tools (Google Maps, ESRI, OpenStreetMap) use the pseudo-Mercator projection (EPSG3857). Let's convert our PPD map to that coordinate system.

```
PPDmap <- st_transform(PPDmap, crs=3857)
st_crs(PPDmap)
```

```
Coordinate Reference System:
  User input: EPSG:3857
  wkt:
PROJCRS["WGS 84 / Pseudo-Mercator",
    BASEGEOGCRS["WGS 84",
        ENSEMBLE["World Geodetic System 1984 ensemble",
            MEMBER["World Geodetic System 1984 (Transit)"],
            MEMBER["World Geodetic System 1984 (G730)"],
            MEMBER["World Geodetic System 1984 (G873)"],
            MEMBER["World Geodetic System 1984 (G1150)"],
            MEMBER["World Geodetic System 1984 (G1674)"],
            MEMBER["World Geodetic System 1984 (G1762)"],
            MEMBER["World Geodetic System 1984 (G2139)"],
            MEMBER["World Geodetic System 1984 (G2296)"],
            ELLIPSOID["WGS 84",6378137,298.257223563,
                LENGTHUNIT["metre",1]],
            ENSEMBLEACCURACY[2.0]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4326]],
    CONVERSION["Popular Visualisation Pseudo-Mercator",
        METHOD["Popular Visualisation Pseudo Mercator",
            ID["EPSG",1024]],
        PARAMETER["Latitude of natural origin",0,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8801]],
        PARAMETER["Longitude of natural origin",0,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8802]],
        PARAMETER["False easting",0,
            LENGTHUNIT["metre",1],
            ID["EPSG",8806]],
        PARAMETER["False northing",0,
            LENGTHUNIT["metre",1],
            ID["EPSG",8807]]],
    CS[Cartesian,2],
        AXIS["easting (X)",east,
            ORDER[1],
            LENGTHUNIT["metre",1]],
        AXIS["northing (Y)",north,
            ORDER[2],
            LENGTHUNIT["metre",1]],
    USAGE[
```

```
        SCOPE["Web mapping and visualisation."],
        AREA["World between 85.06°S and 85.06°N."],
        BBOX[-85.06,-180,85.06,180]],
    ID["EPSG",3857]]
```

The CRS now indicates that this is a Mercator projection with distance measured in meters (`LENGTHUNIT["metre",1]`). There are special coordinate systems for every part of the world. A useful coordinate system for the Philadelphia area is EPSG2272. Let's convert our PPD map to that coordinate system.

```
PPDmap <- st_transform(PPDmap, crs=2272)
st_crs(PPDmap)
```

```
Coordinate Reference System:
  User input: EPSG:2272
  wkt:
PROJCRS["NAD83 / Pennsylvania South (ftUS)",
    BASEGEOGCRS["NAD83",
        DATUM["North American Datum 1983",
            ELLIPSOID["GRS 1980",6378137,298.257222101,
                LENGTHUNIT["metre",1]]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4269]],
    CONVERSION["SPCS83 Pennsylvania South zone (US survey foot)",
        METHOD["Lambert Conic Conformal (2SP)",
            ID["EPSG",9802]],
        PARAMETER["Latitude of false origin",39.3333333333333,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8821]],
        PARAMETER["Longitude of false origin",-77.75,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8822]],
        PARAMETER["Latitude of 1st standard parallel",40.9666666666667,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8823]],
        PARAMETER["Latitude of 2nd standard parallel",39.9333333333333,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8824]],
        PARAMETER["Easting at false origin",1968500,
            LENGTHUNIT["US survey foot",0.304800609601219],
            ID["EPSG",8826]],
        PARAMETER["Northing at false origin",0,
            LENGTHUNIT["US survey foot",0.304800609601219],
```

```
        ID["EPSG",8827]]],
    CS[Cartesian,2],
        AXIS["easting (X)",east,
            ORDER[1],
            LENGTHUNIT["US survey foot",0.304800609601219]],
        AXIS["northing (Y)",north,
            ORDER[2],
            LENGTHUNIT["US survey foot",0.304800609601219]],
    USAGE[
        SCOPE["Engineering survey, topographic mapping."],
        AREA["United States (USA) - Pennsylvania - counties of Adams;
         ↪ Allegheny; Armstrong; Beaver; Bedford; Berks; Blair; Bucks;
         ↪ Butler; Cambria; Chester; Cumberland; Dauphin; Delaware; Fayette;
         ↪ Franklin; Fulton; Greene; Huntingdon; Indiana; Juniata;
         ↪ Lancaster; Lawrence; Lebanon; Lehigh; Mifflin; Montgomery;
         ↪ Northampton; Perry; Philadelphia; Schuylkill; Snyder; Somerset;
         ↪ Washington; Westmoreland; York."],
        BBOX[39.71,-80.53,41.18,-74.72]],
    ID["EPSG",2272]]
```

This coordinate system is the Lambert Conic Conformal (LCC). This particular projection of the `PPDmap` is tuned to provide good precision for the southern part of Pennsylvania (note the parallel coordinates are at the latitude of southern Pennsylvania and the meridian is a little west of Philadelphia) and distances are measured in feet (note the `LENGTHUNIT["US survey foot",0.304800609601219]` tag in the CRS description).

Let's transform back to longitude/latitude. It really is best to work using a different coordinate system, but I'm going to stick with longitude/latitude so that the values make a little more sense to us.

```
PPDmap <- st_transform(PPDmap, crs=4326)
```

Now both the PPD data and the polygons are on the same scale

```
plot(st_geometry(PPDmap), axes=TRUE, cex.axis=0.7)
points(lat~lon, data=ois, col=rgb(1,0,0,0.5), pch=16)
```

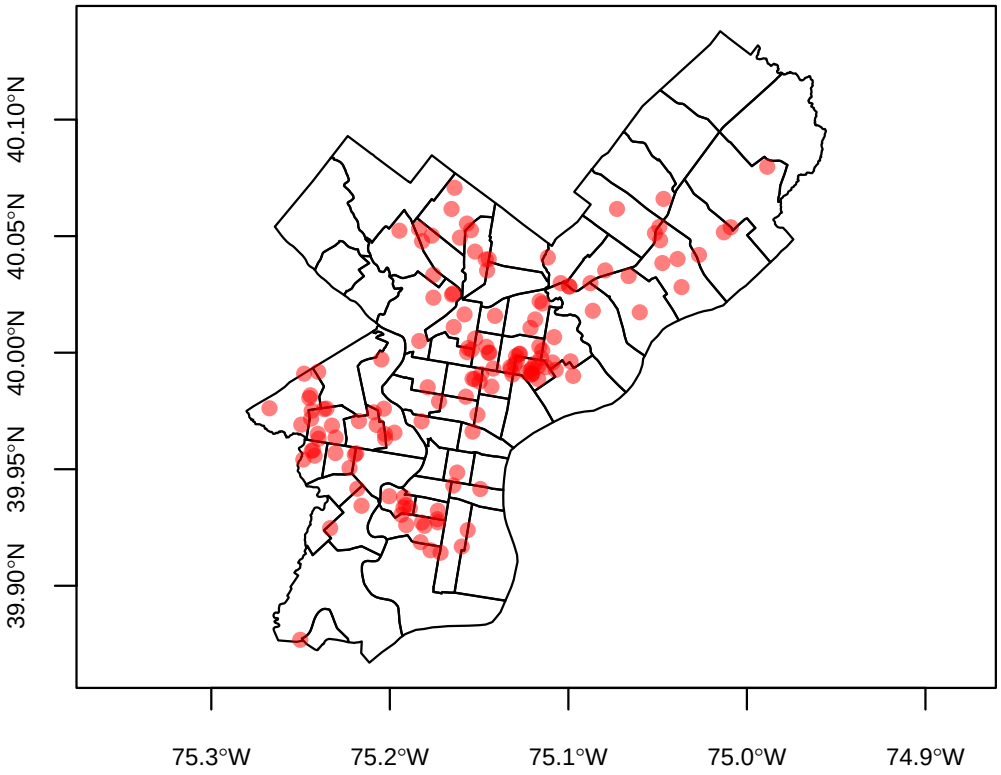

Figure 11: Map of OISs over PSAs

To make the dots a little transparent, I have used the `rgb()` function with which you can mix red, green, and blue colors and set the transparency. The `1` tells `rgb()` to use maximum red. The two `0`s tell `rgb()` to use no green or blue. The 0.5 tells `rgb()` to make the dots halfway transparent.

6.2 Spatial joins

A *spatial join* is the process of linking two data sources by their geography. For the case of the OIS data, we want to know how many OISs occurred in each PSA. To do this we need to

drop each OIS point location into the PSA polygons and have R tell us in which polygon did each OIS land.

First we need to convert our `ois` data frame to an `sf` object, communicating to R that the `lon` and `lat` columns are special. At this stage we also have to communicate in what coordinate system are the `lon` and `lat` values. `st_as_sf()` converts an R object into an `sf` object.

```
ois <- st_as_sf(ois,
                coords=c("lon","lat"),
                crs=4326)
ois |> select(-text, -url, -addrmatch)
```

```
Simple feature collection with 145 features and 8 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -75.26743 ymin: 39.87683 xmax: -74.98862 ymax: 40.07977
Geodetic CRS:  WGS 84
First 10 features:
      id                                                  location subInjury
1  25-14                    2650 South 21st Street, Philadelphia, PA       N/A
2  25-13                  150 West Somerset Street, Philadelphia, PA    Killed
3  25-12                     350 North 65th Street, Philadelphia, PA       N/A
4  25-11                       4650 Roosevelt Blvd, Philadelphia, PA    Killed
5  25-10                         4150 Ogden Street, Philadelphia, PA       N/A
6  25-09                  4650 Roosevelt Boulevard, Philadelphia, PA    Killed
7  25-08                         1650 Moore Street, Philadelphia, PA   Wounded
8  25-06                        2850 Jasper Street, Philadelphia, PA       N/A
9  25-05 1 Philadelphia International Airport Way, Philadelphia, PA    Killed
10 25-04                         4150 Leidy Avenue, Philadelphia, PA       N/A
   subArrest offInjury       date  score      addrtype
1        N/A        No 2025-06-20 100.00 StreetAddress
2        N/A        No 2025-05-21 100.00 StreetAddress
3        N/A       Yes 2025-04-30 100.00 StreetAddress
4        N/A        No 2025-04-28 100.00 StreetAddress
5        N/A        No 2025-03-22 100.00 StreetAddress
6        N/A       Yes 2025-03-20 100.00 StreetAddress
7        Yes        No 2025-03-19 100.00 StreetAddress
8        N/A       Yes 2025-02-04 100.00  PointAddress
9        N/A        No 2025-02-03  95.85    StreetName
10       N/A       Yes 2025-01-27 100.00  PointAddress
                    geometry
1  POINT (-75.18278 39.91872)
2  POINT (-75.13236 39.99267)
3  POINT (-75.24962 39.96913)
4  POINT (-75.09951 40.02845)
```

```
5  POINT (-75.20741 39.96899)
6  POINT (-75.09951 40.02845)
7  POINT (-75.17337 39.92857)
8  POINT (-75.12016 39.99119)
9  POINT (-75.25016 39.87683)
10 POINT (-75.20895 39.97441)
```

You can see that `ois` now has one of those special geometry columns. We can plot the OISs over the PSA map.

```
plot(st_geometry(PPDmap), axes=TRUE, cex.axis=0.7)
plot(st_geometry(ois), add=TRUE, col=rgb(1,0,0,0.5), pch=16)
```

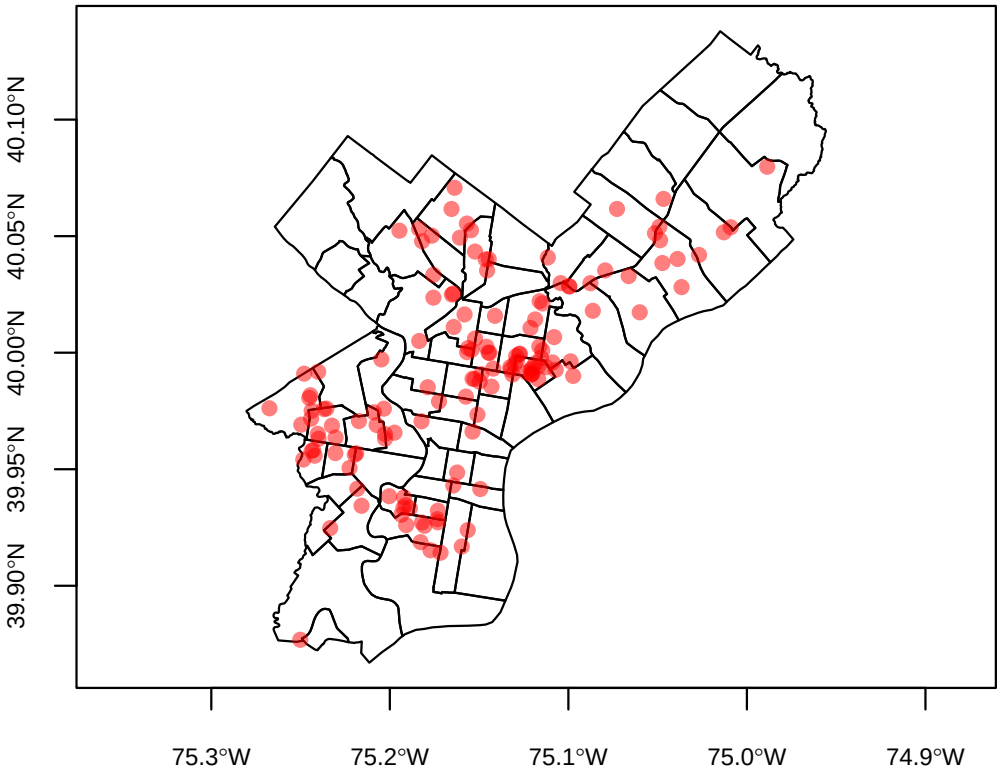

Figure 12: Map of OISs over PSAs using `sf` objects

`st_join()` will match each row in `ois` to each polygon in PSA. When linking a data frame with *point* data (OIS location) with a data frame with *polygon* data (like PSAs), the points will join with the polygons in which they land. I just want to add the `PSA_NUM` column out of the `PPDmap` to our `ois` data.

```
PSAlookup <- ois |>
  st_join(PPDmap |> select(PSA_NUM))
PSAlookup |>
   select(id, date, location, PSA_NUM, geometry) |>
   head()
```

```
Simple feature collection with 6 features and 4 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -75.24962 ymin: 39.91872 xmax: -75.09951 ymax: 40.02845
Geodetic CRS:  WGS 84
     id       date                                    location PSA_NUM
1 25-14 2025-06-20   2650 South 21st Street, Philadelphia, PA     011
2 25-13 2025-05-21 150 West Somerset Street, Philadelphia, PA     253
3 25-12 2025-04-30    350 North 65th Street, Philadelphia, PA     191
4 25-11 2025-04-28      4650 Roosevelt Blvd, Philadelphia, PA     021
5 25-10 2025-03-22        4150 Ogden Street, Philadelphia, PA     162
6 25-09 2025-03-20 4650 Roosevelt Boulevard, Philadelphia, PA     021
                     geometry
1 POINT (-75.18278 39.91872)
2 POINT (-75.13236 39.99267)
3 POINT (-75.24962 39.96913)
4 POINT (-75.09951 40.02845)
5 POINT (-75.20741 39.96899)
6 POINT (-75.09951 40.02845)
```

Now our `PSAlookup` contains everything from `ois` but also adds a new column `PSA_NUM`.

Let's examine the PSA with the most OISs and highlight their incidents on the map.

```
PSAlookup |>
   count(PSA_NUM) |>
   slice_max(n)
```

```
Simple feature collection with 1 feature and 2 fields
Geometry type: MULTIPOINT
Dimension:     XY
Bounding box:  xmin: -75.12886 ymin: 39.98866 xmax: -75.11248 ymax: 39.99641
Geodetic CRS:  WGS 84
  PSA_NUM  n                        geometry
1     242 11 MULTIPOINT ((-75.11248 39.9...
```

```
plot(st_geometry(PPDmap), axes=TRUE, cex.axis=0.7)
PSAlookup |>
   st_join(PSAlookup |>
              count(PSA_NUM) |>
              slice_max(n) |>
              select(-PSA_NUM),
           left=FALSE) |> # right join
  st_geometry() |>
  plot(add=TRUE, col=rgb(0,1,0,0.5), pch=16)
```

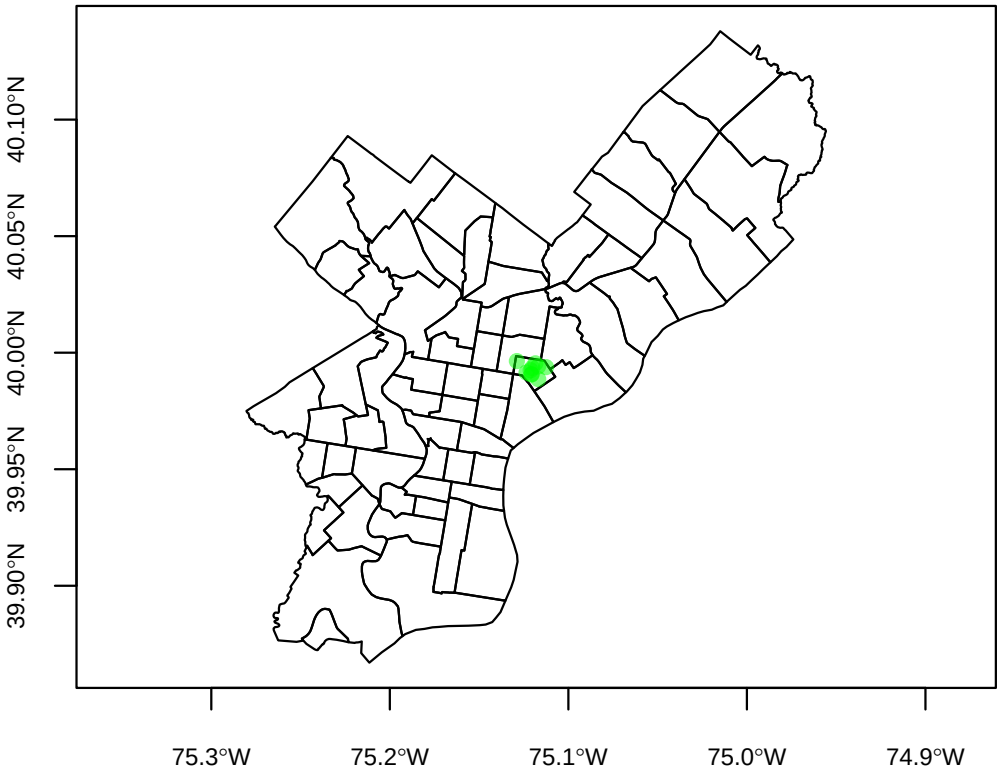

Figure 13: Map of PSA with the largest number of OISs

Let's identify which OISs occurred in the same PSA as the University of Pennsylvania. We've already geocoded Penn and have its coordinates. Let's join it with `PPDmap` to find out which PSA it is in.

```
gcPenn
```

```
                                          address location.x location.y
1 3718 Locust Walk, Philadelphia, Pennsylvania, 19104  -75.19758   39.95227
  score                              attributes.Match_addr
1   100 3718 Locust Walk, Philadelphia, Pennsylvania, 19104
```

```
  attributes.Addr_type extent.xmin extent.ymin extent.xmax extent.ymax
1        StreetAddress   -75.19858    39.95127   -75.19658    39.95327
        lon      lat
1 -75.19758 39.95227
```

```
st_as_sf(gcPenn,
         coords=c("lon","lat"),
         crs=4326) |> # tell R that the coords are lon/lat
   st_join(PPDmap) |>
   select(PSA_NUM)
```

```
Simple feature collection with 1 feature and 1 field
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -75.19758 ymin: 39.95227 xmax: -75.19758 ymax: 39.95227
Geodetic CRS:  WGS 84
  PSA_NUM                   geometry
1     183 POINT (-75.19758 39.95227)
```

Now we see that Penn is in PSA 183 and we can highlight those points on the map.

```
plot(st_geometry(PPDmap), axes=TRUE, cex.axis=0.7)
PSAlookup |>
  filter(PSA_NUM=="183") |>
  st_geometry() |>
  plot(add=TRUE, col="blue", pch=16)
```

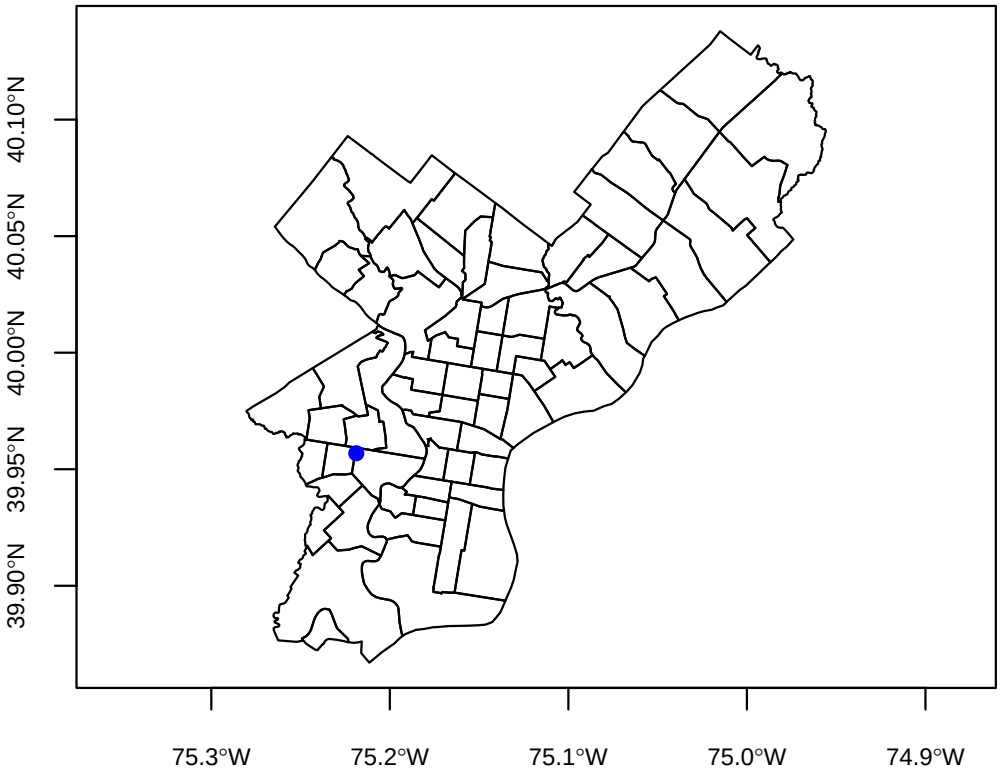

Figure 14: Map of OISs in the same PSA as Penn

We read about this incident earlier when fixing the OIS incident locations in Section 5.

6.3 Coloring a map based on the value of a feature

Lastly, we will tabulate the number of OISs in each PSA and color the map by the number of OISs.

```
# merge the shooting count into the PPDmap data
PPDmap <- PPDmap |>
   left_join(PSAlookup |>
```

```
                count(PSA_NUM) |>
                st_drop_geometry(),
            by=join_by(PSA_NUM)) |>
  rename(nShoot=n) |>
  mutate(nShoot=replace_na(nShoot, 0))

head(PPDmap)
```

```
Simple feature collection with 6 features and 8 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -75.26443 ymin: 39.87838 xmax: -75.0572 ymax: 40.09295
Geodetic CRS:  WGS 84
  OBJECTID SECT_CODE DIV_CODE DIST_NUMC PSA_NUM Shape__Area Shape__Length
1        1      <NA>      EPD        24     243    13256469      16062.12
2        2      <NA>     NWPD        14     144    22663486      26039.57
3        3      <NA>     NEPD        15     151    17388456      22268.67
4        4      <NA>     SWPD        19     193    11696975      18387.39
5        5      <NA>      SPD        01     012    26771567      26695.53
6        6      <NA>     NWPD        05     053    20373551      21860.86
  nShoot                       geometry
1      2 POLYGON ((-75.11026 39.9929...
2      0 POLYGON ((-75.16339 40.0769...
3      3 POLYGON ((-75.0572 39.99865...
4      1 POLYGON ((-75.23857 39.9761...
5      1 POLYGON ((-75.17143 39.9167...
6      0 POLYGON ((-75.21072 40.0413...
```

We can see that `PPDmap` now has a new `nShoot` column. A histogram will show what kinds of counts we observe in the PSAs.

```
hist(PPDmap$nShoot, xlab="Number of OISs", ylab="Number of PSAs", main="")
```

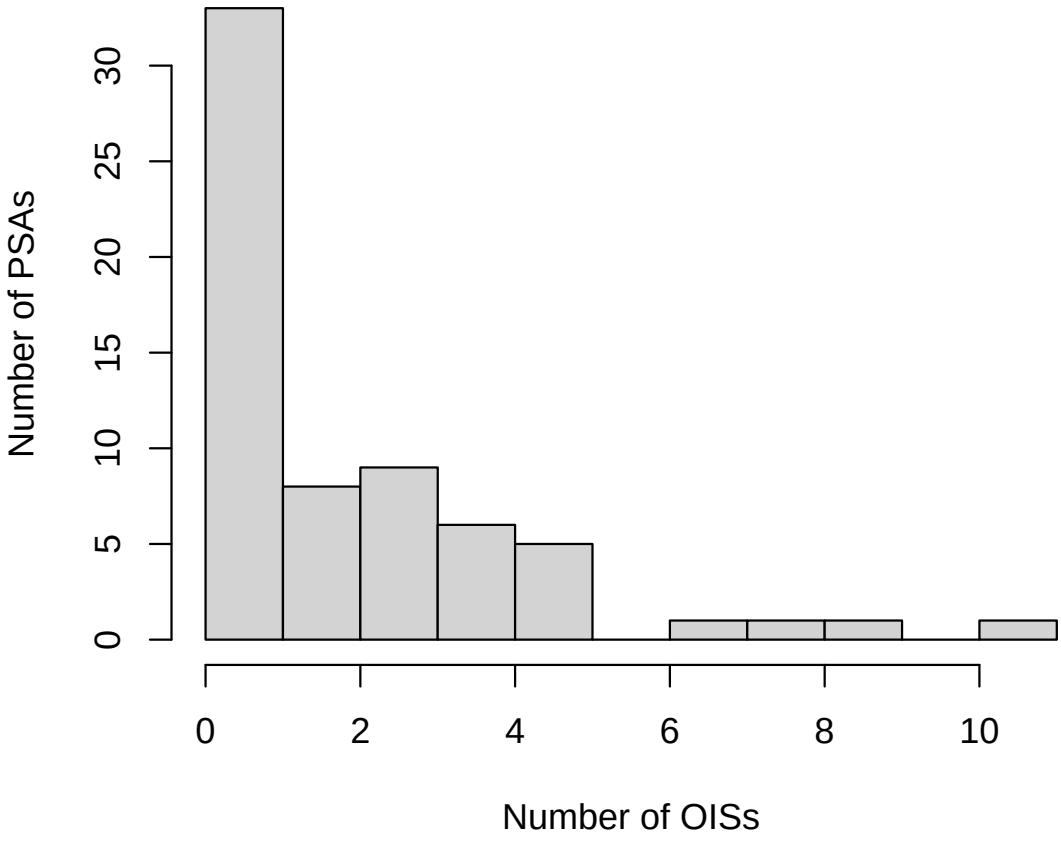

Figure 15: Histogram of OIS incident counts by PSA

Let's discretize the OIS counts into a few categories.

```
PPDmap <- PPDmap |>
  mutate(catShoot =
           cut(nShoot,
               breaks=c(0,1,2,3,4,8,Inf),
               right=FALSE))
```

`cut()` converts all of the individual counts into categories, like [1,5) or [25,30). For each of these categories we will associate a color for the map. `heat.colors()` will generate a sequence of colors in the yellow, orange, red range.

```
a <- data.frame(catShoot = levels(PPDmap$catShoot),
                col      = rev(heat.colors(6,1)))
a
# some other color options
#    col = rev(rainbow(6,1))
#    or generate a range of red colors
#    col = rgb(seq(0,1,length=6),0,0,1)
```

```
  catShoot       col
1    [0,1) #FFFF80FF
2    [1,2) #FFFF00FF
3    [2,3) #FFBF00FF
4    [3,4) #FF8000FF
5    [4,8) #FF4000FF
6  [8,Inf) #FF0000FF
```

These are eight digit codes describing the color. The first two digits correspond to red, digits three and four correspond to green, digits five and six correspond to blue, and the last two digits correspond to transparency. These are hexadecimal numbers (base 16). Hexadecimal numbers use the digits 0-9, like normal decimal system numbers, and then denote 10 as A, 11 as B, on up to 15 as F. So FF as a decimal is $15 \times 16 + 15 = 255$, which is the maximum value for a two digit hexadecimal. The hexadecimal 80 as a decimal is $8 \times 16 + 0 = 128$, which is in the middle of the range 0 to 255. So the first color code, FFFF80FF, means maximum red, maximum green, half blue, and not transparent at all. This mixture is known more commonly as "yellow".

Now we join `PPDmap` with our color lookup table in `a` and plot it.

```
# match the color to category
PPDmap <- PPDmap |>
  left_join(a, by=join_by(catShoot))

PPDmap |>
  st_geometry() |>
  plot(col=PPDmap$col, border="black")
# add the number of shootings
#   warning is reminder that it is collapsing polygon data down to a point
b <- st_coordinates(st_centroid(PPDmap))
```

```
Warning: st_centroid assumes attributes are constant over geometries
```

```
text(b[,1], b[,2], PPDmap$nShoot, cex=0.7)
```

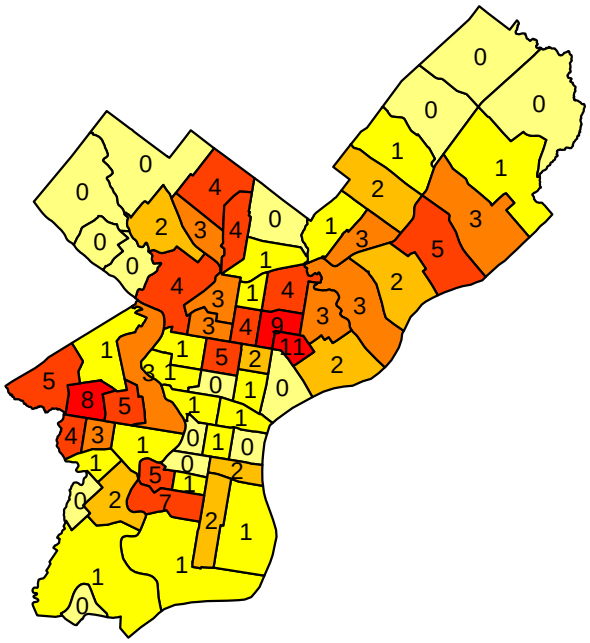

Figure 16: PSAs color-coded by number of OIS incidents

Those PSAs with the least shootings are a very pale yellow. As we examine PSAs with a greater number of OISs, their colors get redder and redder.

`sf` objects have their own default plotting method to accomplish these kind of "heat maps." The default palette is generated with `sf.colors()`.

```
PPDmap |>
  select(nShoot) |>
  plot(main="")
```

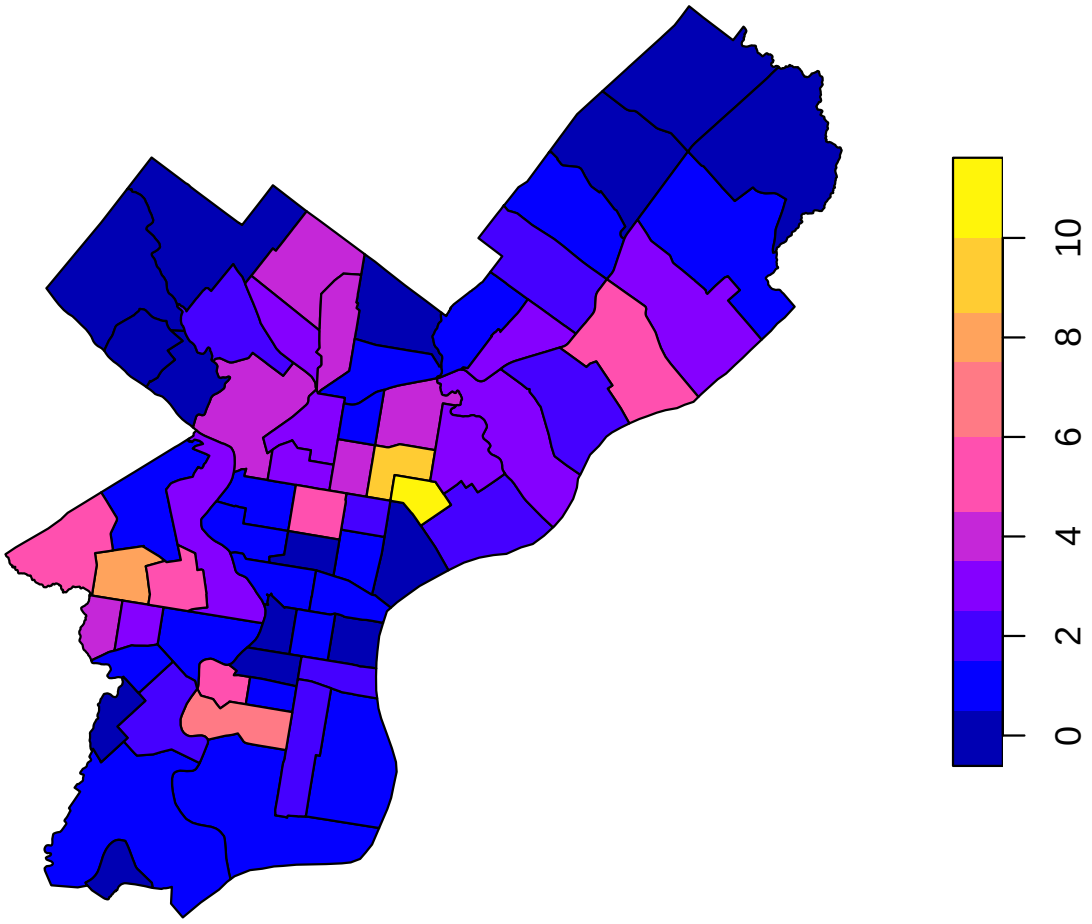

Figure 17: PSAs color-coded by number of OIS incidents, default `sf` colors

You can change the color palette, for example, by using the yellow-orange-red palette from HCL (hue, chroma, luminance) color set.

```
PPDmap |>
  select(nShoot) |>
  plot(main="",
       pal = hcl.colors(6, "YlOrRd", rev = TRUE),
       nbreaks = 6)
```

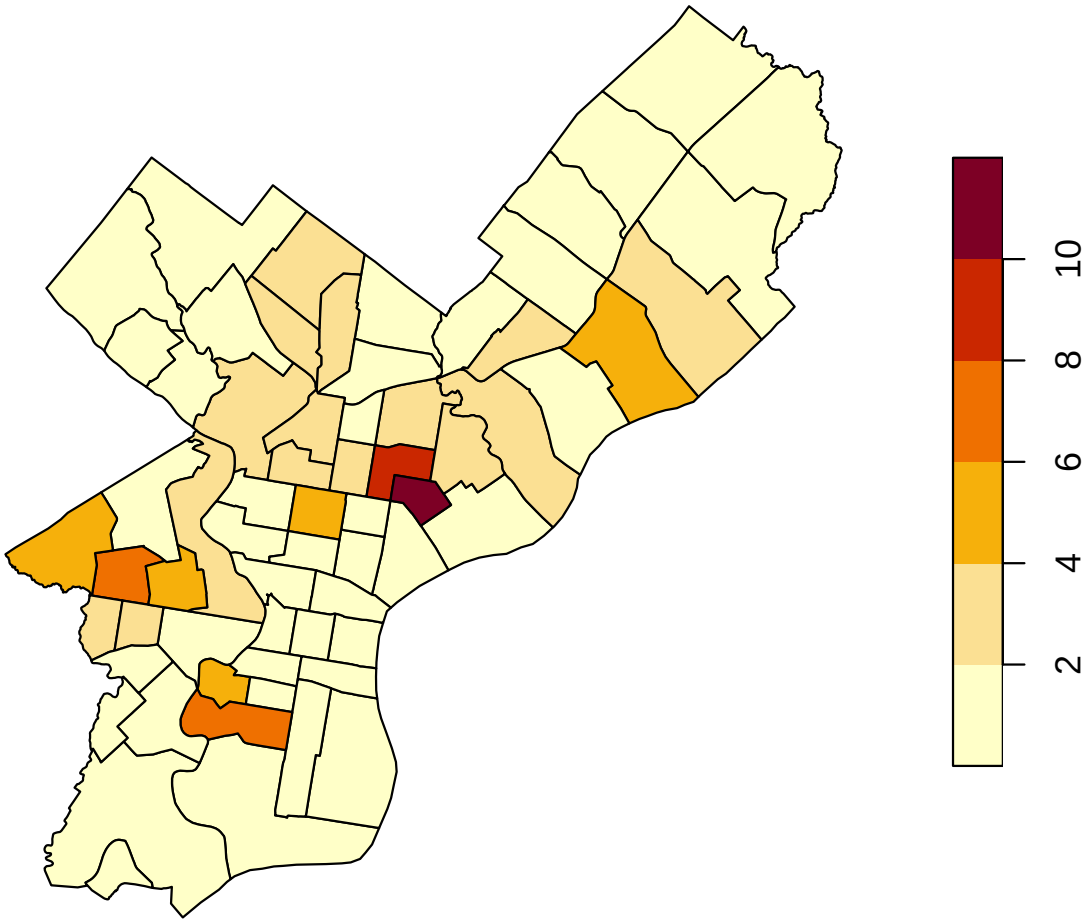

Figure 18: PSAs color-coded by number of OIS incidents, YlOrRd HCL palette

Lastly, I will share the classic *viridis* palette, noted for being color-blind friendly and "perceptually uniform".

```
# classic viridis palette
library(viridis)
```

```
Loading required package: viridisLite
```

```
PPDmap |>
  select(nShoot) |>
  plot(main="",
```

```
      pal = viridis_pal(option="D"),
      nbreaks = 12)
```

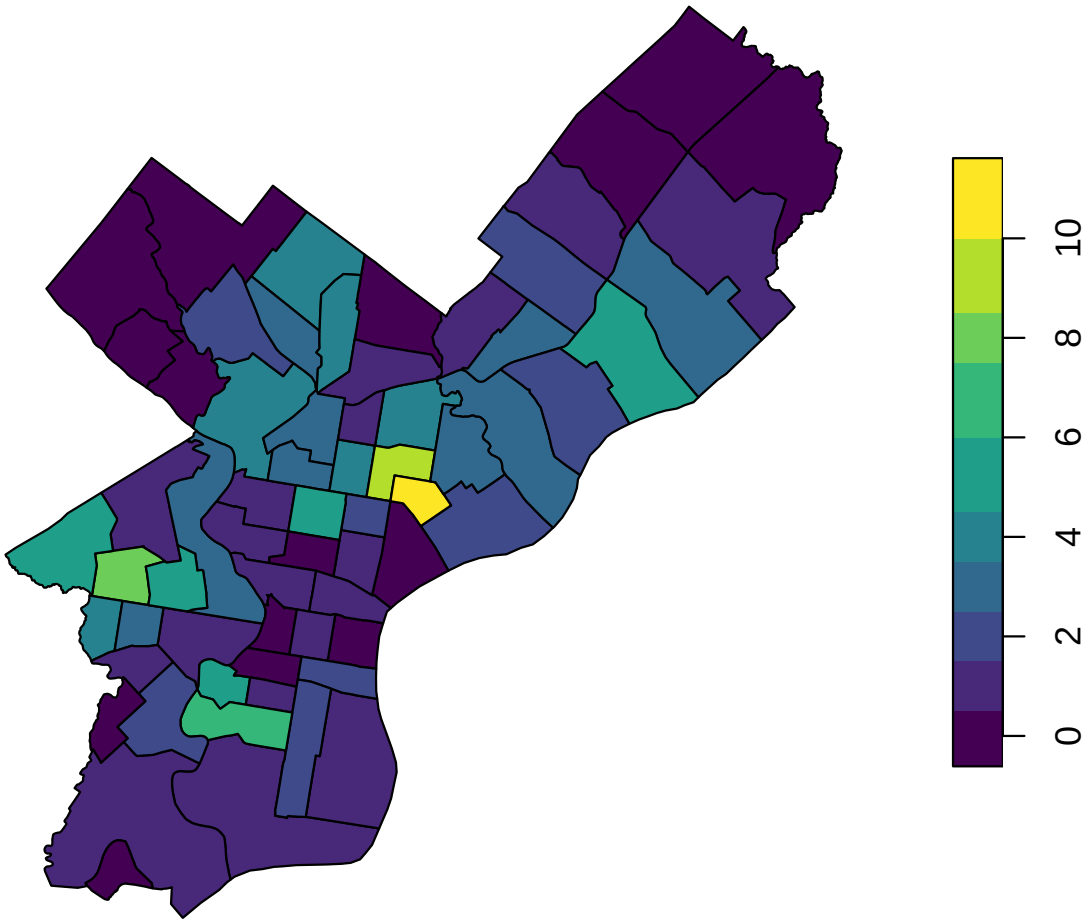

Figure 19: PSAs color-coded by number of OIS incidents, viridis palette

And a leaflet version to end on.

```
PPDmap <- PPDmap |>
  mutate(label = paste("PSA:",PSA_NUM, "Count:",nShoot))

PPDmap |>
   leaflet(width = 1200, height = 800) |>
   addPolygons(weight=1, col=~col, label=~label) |>
```

```
  addTiles(
    urlTemplate = "https://api.maptiler.com/maps/streets/{z}/{x}/{y}.png?key={key
  }",
    options = tileOptions(key = Sys.getenv("MAPTILER_API_KEY"),
                          tileSize = 512,
                          zoomOffset = -1),
    attribution = '© MapTiler © OpenStreetMap contributors')
```

Figure 20: Map of OIS counts by PSA

7 Summary

We started with just a web page linking to a collection of text descriptions. We used a variety of webscraping and regular expressions to extract everything we could from the web page tables. We had R "read" the text descriptions to extract the dates. We geocoded the incidents so that we could put them on a map. Finally, we tabulated by PSA the number of OISs and mapped those as well.

If you have worked through all of this, then I would recommend that you save your objects, using `save(ois, PSAlookup, file="PPDOIS.RData")`. That way you will not have to scrape everything off the web again or redo any geocoding.

8 Exercises

1. Revisit the geocoding section discussing geocoding errors. Examine the OISs that have not been geocoded to specific locations. Fix their addresses and redo the geocoding of these OISs to improve the accuracy of the data.
2. Identify officer-involved shootings that resulted in the offender being transported to the Hospital at the University of Pennsylvania. Create a map marking the location of HUP, the location of officer-involved shootings resulting in the offender being transported to HUP, and the locations of all other shootings.
3. For each shooting determine which hospital treated the offender. Use `st_distance()` to determine what percentage of those shot in an OIS went to the closest hospital.