Extracting data from text and geocoding

Greg Ridgeway (gridge@upenn.edu)

October 31, 2018

library(lubridate) library(pdftools) library(jsonlite) library(ggmap) library(sf)

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 2007. Have a look at their OIS webpage. While a lot of information has been posted to the webpage, more information is buried in pdf files associated with 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 pdf files for any incidents missing incident dates, clean up addresses using regular expressions, geocode the addresses to latitude/longitude using the OpenStreetMap geocoder and the ArcGIS geocoder (using JSON), and then make maps describing the shootings.

Scraping the OIS data

Let's start by grabbing the raw HTML from the PPD OIS webpage

```
a <- scan("http://www.phillypolice.com/ois/", what="", sep="\n")
```

`scan()` is a very simple function that just pulls in text from a file or URL. It does not attempt to do any formatting. `what=""` tells `scan()` to treat what it is reading in as text and `sep="\n"` tells `scan()` to break the text apart whenever it encounters a line feed character.

The first several elements of a are just HTML code setting up the page.

```
a[1:4]
```

```
[1] "<!DOCTYPE html>"
[2] "<html>"
[3] "<head>"
[4] "<title>Officer Involved Shootings | Philadelphia Police Department</title>"
```

But further on down you will find HTML code containing the OIS information that we seek. Let's look at one of the 2018 OISs.

```
i <- grep("id=\"2018-2954", a)
a[i + 0:9]
```

```
 [1] "<tr id=\"2018-2954\">"
 [2] "<td><a href=\"/assets/crime-maps-stats/officer-involved-shootings/18-01.pdf\" class=\"fanc
 [3] "<td>01/13/2018</td> "
 [4] "<td>2800 Block of Kensington Avenue</td> "
 [5] "<td>Wounded</td> "
```

```
 [6] "<td>Yes</td> "
 [7] "<td>No</td> "
 [8] "<td>Pending</td> "
 [9] "<td>Pending</td> "
[10] "</tr>"
```

For the data from 2013-2018, each table row related to an OIS starts with something like `<tr id="2018-2954">`. The very next row contains the URL of the pdf containing more detailed data. The third row contains the date and the fourth row contains the address. There are additional cells indicating injuries and how the shooting was adjudicated, but we will not work with these in this exercise.

Note that if we want the date, it is always two elements after the `<tr>` tag. We can use `gsub()` to strip away the unwanted HTML tags.

```r
a[i + 2]
gsub("<[^>]*>","",a[i + 2])
```

```
[1] "<td>01/13/2018</td> "
[1] "01/13/2018 "
```

We'll use this same strategy for all of the shootings and for the OISs id, date, location, and URL. Start by using `grep()` to find all of the lines of HTML code that start off a row for an OIS form 2013-2018 (the data for OISs before 2013 look a little different).

```r
i <- grep("id=\"201(3|4|5|6|7|8)",a)
ois <- data.frame(
  id      =gsub("<[^>]*>", "", a[i+1]),
  date    =gsub("<[^>]*>", "", a[i+2]),
  location=gsub("<[^>]*>", "", a[i+3]),
  url     =gsub("<td><a href=\"(.*)\" class=.*", "\\1", a[i+1]),
  stringsAsFactors=FALSE)
```

For shootings between 2007-2012, the table provides no incident date and the incident location is 2 elements after the `<tr>` tag rather than 3.

```r
i <- grep("id=\"20(07|08|09|10|11|12)",a)
temp <- data.frame(
  id      =gsub("<[^>]*>","",a[i+1]),
  date    =NA,
  location=gsub("<[^>]*>","",a[i+2]),
  url     =gsub("<td><a href=\"(.*)\" class=.*","\\1",a[i+1]),
  stringsAsFactors=FALSE)
```

Now we can stack all the data from 2007-2018 together and clear out some extra spaces in the `id` column.

```r
ois <- rbind(ois, temp)
ois$id<- gsub(" ","",ois$id)
ois[1:5,]
```

```
     id        date                             location
```

```
1 18-01 01/13/2018  2800 Block of Kensington Avenue
2 18-02 01/29/2018      1300 Block of Bigler Street
3 18-08 04/18/2018     3100 block of N. 33rd Street
4 18-12 06/08/2018     1400 block of Lardner Street
5 18-16 08/06/2018        4800 block of Knox Street
                                                            url
1    /assets/crime-maps-stats/officer-involved-shootings/18-01.pdf
2 /assets/crime-maps-stats/officer-involved-shootings/OIS18-02.pdf
3 /assets/crime-maps-stats/officer-involved-shootings/OIS18-08.pdf
4    /assets/crime-maps-stats/officer-involved-shootings/18-12.pdf
5    /assets/crime-maps-stats/officer-involved-shootings/18-16.pdf
```

Everything from the PPD OIS page is now neatly stored in an R data frame.

We will need the full URL (rather than the relative URL) for all the pdf files.

```
ois$url <- paste0("http://www.phillypolice.com", ois$url)
```

A couple of the entries have problematic `id`, some easily fixed.

```
grep("^[^-]*$", ois$id, value=TRUE)
ois$id[ois$id=="1630"] <- "16-30"
ois$id[ois$id=="1730"] <- "17-30"
ois$id[ois$id=="1822"] <- "18-22"
```

```
[1] "1822" "1730" "1630" ""     ""     ""
```

A few other OISs with missing `id` are a little harder to figure out.

```
subset(ois, id=="")
```

```
    id       date                      location
77     04/22/2014  5100 block of Willows Avenue
275           <NA>      â\200œBâ\200\235 and Ontario St
282           <NA>       â\200œAâ\200\235 and Louden St
                                    url
77  http://www.phillypolice.com/assets/
275 http://www.phillypolice.com/assets/
282 http://www.phillypolice.com/assets/
```

The Willows Avenue OIS has a missing `id`, but the one before it has `id` 14-12 and the one after it has `id` 14-16.

```
i <- grep("5100 block of Willows Avenue", ois$location)
ois[(i-1):(i+1),]
```

```
      id       date                      location
76 14-12 03/25/2014  100 block of W. Louden Street
77       04/22/2014   5100 block of Willows Avenue
78 14-16 04/26/2014     5400 block of Media Street
                                                                                url
76 http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/14-12.pdf
77                                                      http://www.phillypolice.com/assets/
```

```
78 http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/14-16.pdf
```

Let's see if there is a pdf for 14-13, 14-14, or 14-15.

```
a <- try(scan("http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/14
```

```
Warning in file(file, "r"): cannot open URL 'http://www.phillypolice.com/
assets/crime-maps-stats/officer-involved-shootings/14-13.pdf': HTTP status
was '404 Not Found'
```

```
a <- try(scan("http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/14
```

```
Warning in file(file, "r"): cannot open URL 'http://www.phillypolice.com/
assets/crime-maps-stats/officer-involved-shootings/14-14.pdf': HTTP status
was '404 Not Found'
```

```
a <- try(scan("http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/14
```

```
Warning in scan("http://www.phillypolice.com/assets/crime-maps-stats/
officer-involved-shootings/14-15.pdf", : embedded nul(s) found in input
```

Okay! So the OIS's id must be 14-15. We'll fix the URL to the associated pdf file too.

```
ois$id[ois$location=="5100 block of Willows Avenue " & ois$date=="04/22/2014 "] <- "14-15"
ois$url[ois$id=="14-15"] <- "http://www.phillypolice.com/assets/crime-maps-stats/officer-involve
```

The problem with the remaining two addresses are the smart quotes around "A" and "B" in the street name.

```
i <- which(ois$id=="")
ois[c((i[1]-1):(i[1]+1),(i[2]-1):(i[2]+1)), 1:3]
```

```
       id date                    location
274 10-60 <NA> 1200 block of W. Harold St
275       <NA>     â\200œBâ\200\235 and Ontario St
276 10-67 <NA>    5200 block of Marlow St
281 10-74 <NA>           3300 Brighton St
282       <NA>      â\200œAâ\200\235 and Louden St
283 10-78 <NA>   2900 block of Island Ave
```

Before 2013, the names of the pdf files combined the OIS id and the location. But smart quotes can cause problems in URLs and file names. This is what likely caused a problem here. I made some guesses about what the pdf file name must be and found pdfs for "10-65 B and Ontario St.pdf" and "10-76 A and Louden St.pdf". So, the `id` right after OIS 10-60 is 10-65 and the `id` right after OIS 10-74 is 10-76. We can clean up the locations and the URLs too.

```
ois$id[1 + which(ois$id=="10-60")] <- "10-65"
ois$id[1 + which(ois$id=="10-74")] <- "10-76"

ois$location[ois$id=="10-65"] <- "B St and Ontario St"
ois$location[ois$id=="10-76"] <- "A St and Louden St"

ois$url[ois$id=="10-65"] <- "http://www.phillypolice.com/assets/crime-maps-stats/officer-involve
```

```
ois$url[ois$id=="10-76"] <- "http://www.phillypolice.com/assets/crime-maps-stats/officer-involve
```

Many of the locations have odd looking characters.

```
grep("&", ois$location, value=TRUE)
```

```
 [1] "49th &amp; Walnut Streets "
 [2] "near 16th Street &amp; Allegheny Ave "
 [3] "4800, 4900 &amp; 5100 blocks of Sansom "
 [4] "32nd &amp; Susquehanna Ave "
 [5] "&quot;A&quot; &amp; Somerset Streets "
 [6] "Haverford Ave &amp; Drexel Road "
 [7] "22nd &amp; Morris Streets "
 [8] "Island Ave &amp; Elmwood Street "
 [9] "Devon &amp; Locust Streets "
[10] "Robbins Street &amp; Castor Avenue "
[11] "5600 block of Lansdowne Avenue &amp; 58th Street "
[12] "Cambria &amp; Warnock Streets "
[13] "21st &amp; Spencer St "
[14] "Cobbs Creek &amp; Spruce St "
[15] "Broad &amp; Olney Ave "
[16] "Kennsington Ave &amp; Sommerset "
[17] "Old York Rd. &amp; Tabor Rd "
[18] "Haverford Ave. &amp; Sherwood Road "
[19] "17th &amp; Westmoreland St "
[20] "Oxford Ave &amp; Benner St "
[21] "Brown &amp; Sloan St "
[22] "Greenway Ave &amp; Avondale St "
[23] "19th &amp; Cheltenham Ave "
[24] "Watts St &amp; Cambridge St "
```

HTML represents certain special characters with special codes. For example, “&” is the ampersand, “”" is a quote, and “ ” is a non-breaking space, a space that always keeps the word before and after the space on the same line. HTML special characters always start with a & and end with ;. Let’s clean up these HTML codes and also remove any extra leading or trailing spaces from the location.

```
ois$location <- gsub("&quot;",  "",    ois$location)
ois$location <- gsub("&amp;",   "and", ois$location)
ois$location <- gsub("^ *| *$", "",    ois$location)
```

Some of the URLs have HTML codes in them too. These need to be replaced with & or R will not be able to find these pdf files.

```
# An example URL with an HTML codes in it
ois$url[ois$id=="07-53"]
ois$url <- gsub("&amp;", "&", ois$url)
```

```
[1] "http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/2007/07-53 W
```

Lastly, let’s reformat the dates using the `lubridate` package.

```r
library(lubridate)
```

```
Attaching package: 'lubridate'

The following object is masked from 'package:base':

    date
```

```r
ois$date <- mdy(ois$date)
```

Now our data frame `ois` should have all its IDs, with properly formatted dates, clean locations, and correct URLs indicating the location of the pdf files with details about the shootings.

```r
head(ois)
```

```
     id       date                         location
1 18-01 2018-01-13 2800 Block of Kensington Avenue
2 18-02 2018-01-29     1300 Block of Bigler Street
3 18-08 2018-04-18    3100 block of N. 33rd Street
4 18-12 2018-06-08    1400 block of Lardner Street
5 18-16 2018-08-06       4800 block of Knox Street
6 18-17 2018-08-09     2000 block of Snyder Avenue
                                                                                 url
1    http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/18-01.pdf
2 http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/OIS18-02.pdf
3 http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/OIS18-08.pdf
4    http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/18-12.pdf
5    http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/18-16.pdf
6    http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootings/18-17.pdf
```

Extracting data from pdf files

All of the OISs in 2012 and earlier are missing the incident dates.

```r
# pick the first two OISs in each year
a <- ois$id[ois$id<"13-00"]
a <- sapply(split(a, substr(a, 1, 2)), function(x) x[1:2])
subset(ois, id %in% sort(as.character(a)))[,1:3]
```

```
       id date                location
139 12-01 <NA>            2500 N 32nd St
140 12-02 <NA>          1900 W. Erie Ave
198 11-02 <NA>            2400 Ridge Ave
199 11-04 <NA>      5200 Westminster Ave
242 10-01 <NA>             600 Willow St
243 10-03 <NA>            6600 Dicks Ave
293 09-01 <NA>            2200 S 56th St
294 09-02 <NA>         3900 Fairmount Ave
354 08-01 <NA>    5600 block of Boyer St
```

```
355 08-02 <NA> 900 block of W. Butler St
396 07-01 <NA>              4400 N 17 ST
397 07-02 <NA>              2400 N 10 ST
```

However, the pdf documents describing the incident contains the date of the incident. Rather than reading all the pdf files and transcribing the dates, we are going to have R do all the work.

The package `pdftools` includes functions for exploring pdf files. Let's load the library and have a look at one of the pdf files describing incident 07-01.

```
library(pdftools)
pdfFilename0701 <- "http://www.phillypolice.com/assets/crime-maps-stats/officer-involved-shootin
pdfText0701 <- pdf_text(pdfFilename0701)
pdfText0701
```

```
[1] "PS# 07-01\r\n1/1/07\r\nOn 1/1/07, at approximately 12:02 A.M., uniformed officers were trav
```

`pdf_text()` extracts all the raw text from the pdf file. Right at the beginning of the file we can see a date, 1/1/07. That's what we want. Note that there are scattered `\r\n` throughout. These are carriage return (`\r`) and line feed (`\n`) characters that signal the end of a line. The old printers would look for these characters to move the printer head back to the beginning of a line (carriage return) and advance the page to the next line (line feed). Nowadays, those same characters are still used to denote the end of a line. However, PCs use `\r\n`, Unix systems use `\n`, older Macs used `\r`, but Mac OS X adopted the Unix standard `\n`. Expect any of these combinations in data files. I have this running on a PC, so I am going to use those `\r\n` to separate the lines and isolate the date.

```
a <- strsplit(pdfText0701, split = "\r\n")[[1]]
a
```

```
 [1] "PS# 07-01"
 [2] "1/1/07"
 [3] "On 1/1/07, at approximately 12:02 A.M., uniformed officers were traveling east on"
 [4] "Wingohocking Street from 18th Street when they heard multiple gunshots coming from the 440
 [5] "block of N.17th Street. When approaching the intersection of 17th and Wingohocking Street,
 [6] "officers observed a male discharging an assault rifle into the air on the 4400 block of N.
 [7] "Street."
 [8] "The officers exited their patrol vehicle, identified themselves as police officers, and or
 [9] "offender to put the weapon down. The offender then turned and pointed the weapon in the"
[10] "officers' direction. One of the officers discharged his weapon at the offender. The offend
[11] "dropped the weapon and fled into a residence on the 4400 block of N. 17th Street where he
[12] "apprehended after a brief struggle."
[13] "A .223 caliber rifle loaded with four live rounds was recovered at the scene. There were n
[14] "reported injuries resulting from the police discharge. Two additional apprehensions were m
[15] "the scene for assault on police and related offenses."
```

Now let's apply the `mdy()` function to `a`. Now most of the text looks nothing like dates, so for those `mdy()` will just give us an `NA`. But for the properly formatted dates, we will get a date object back.

```
a <- mdy(a)
```

```
Warning: 14 failed to parse.
```

```
a
```

```
 [1] NA           "2007-01-01" NA           NA           NA
 [6] NA           NA           NA           NA           NA
[11] NA           NA           NA           NA           NA
```

In this case, only one of the lines had a proper date, but it is possible that some document might have more than one. So we will record just the first one using `sort(a)[1]`. By default, `sort()` tosses all the `NA`s.

Let's put this all together. We will look through each OIS, read its pdf file, save the text in a new column in `ois`.

```
ois$text <- NA
for(i in 1:nrow(ois))
{
   a <- pdf_text(ois$url[i])
   # in case there is more than one page, collapse all pages together
   a <- paste(a, collapse="\r\n")
   ois$text[i] <- a
}
```

Now our `ois` data frame has columns: id, date, location, url, text.

Right now the row for OIS 07-01 has a missing date.

```
subset(ois, id=="07-01")[,1:3]
```

```
       id date     location
396 07-01 <NA> 4400 N 17 ST
```

For any OIS that is missing the date, extract the date from the pdf text.

```
for(i in which(is.na(ois$date)))
{
   # try to find dates, take the first valid date
   a <- strsplit(ois$text[i], split = "\r\n")[[1]]
   a <- sort(mdy(a))[1]

   ois$date[i] <- as.character(a)
}
```

And let's check what happened to the date for OIS 07-01.

```
subset(ois, id=="07-01")[,1:3]
```

```
       id       date     location
396 07-01 2007-01-01 4400 N 17 ST
```

And let's check that we have all valid dates

```
sum(is.na(ymd(ois$date)))
```

```
[1] 0
```

As another check, let's make sure that all the OISs with `id` starting with 07 have dates in 2007, OISs with `id` starting with 08 have dates in 2008, and so on.

```
aggregate(year(date) ~ substr(id,1,2),
          data=ois,
          FUN=unique)
```

```
   substr(id, 1, 2) year(date)
1                07       2007
2                08       2008
3                09       2009
4                10       2010
5                11       2011
6                12       2012
7                13       2013
8                14       2014
9                15       2015
10               16       2016
11               17       2017
12               18       2018
```

Yes, every incidents' `id` matches with the year derived from the date. This doesn't guarantee that all the dates were extracted correctly, but it is a good check to catch any obvious errors.

Without having to transcribe dates from pdf files, we have now filled in all the dates!

Geocoding the OIS locations

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

Geocoding is the process of converting a text description of a location (typically and 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 understand very general descriptions of locations. You can ask for the coordinates of something like "bobbys burger palace near UPenn" and it will understand that "UPenn" means the University of Pennsylvania and that "bobbys burger palace" is celebrity chef Bobby Flay's fast food burger joint. Unfortunately, as of June 2018 Google Maps now 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.

These technologies are still rapidly evolving, so it is most important to learn how to learn to use these tools. We will use the OpenStreetMap geocoder and the ArcGIS geocoder to give you the sense of how to work with them.

Many web data sources use a standardized language for providing data. JSON (JavaScript Object Notation) is quite common and both OpenStreetMap and ArcGIS use JSON.

The URL for OpenStreetMap has the form `http://nominatim.openstreetmap.org/search/3718%20Locust%20Wal`

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("http://nominatim.openstreetmap.org/search/3718%20Locust%20Walk,%20Philadelphia,%20PA?forma
     what="", sep="\n")
```


```
[1] "[{\"place_id\":\"228278154\",\"licence\":\"Data Â© OpenStreetMap contributors, ODbL 1.0. ht
```


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 pr ocess fast and easily.

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

```
library(jsonlite)
fromJSON("http://nominatim.openstreetmap.org/search/3718%20Locust%20Walk,%20Philadelphia,%20PA?f
```

```
   place_id
1 228278154
                                                                     licence
1 Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright
  osm_type   osm_id
1      way 32108143
                                                                boundingbox
1 39.952309653061, 39.952409653061, -75.198505469388, -75.198405469388
               lat               lon
1 39.9523596530612 -75.1984554693878
                                                                                          display_nam
1 3718, Locust Walk, University City, Philadelphia, Philadelphia County, Pennsylvania, 19104, US
  class  type importance address.house_number address.footway
1 place house      0.421                 3718     Locust Walk
  address.neighbourhood address.city      address.county address.state
1       University City Philadelphia Philadelphia County  Pennsylvania
  address.postcode address.country address.country_code
1            19104             USA                   us
```

`fromJSON()` converts the results from the OpenStreetMap geocoder to a row in a data frame. The JSON tags turn into column names and the values are placed as data in a row.

The major drawback of the OpenStreetMap geocoder is that it does not handle intersections. If we try to geocode "38th St and Walnut St", the results come back empty.

```
fromJSON("http://nominatim.openstreetmap.org/search/38th%20St%20and%20Walnut%20St,%20Philadelphi
```

```
list()
```

This is where the ArcGIS geocoder comes in handy. The ArcGIS geocoder also works with JSON but the URL construction is a little different. Here we get several results, but clearly the first one is the one that we want. It is also the one that has the highest `score`.

```r
fromJSON("https://geocode.arcgis.com/arcgis/rest/services/World/GeocodeServer/findAddressCandida
```

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

$spatialReference$latestWkid
[1] 4326


$candidates
                                                         address
1        S 38th St & Walnut St, Philadelphia, Pennsylvania, 19104
2     E State St & Walnut St, Kennett Square, Pennsylvania, 19348
3                    Walnut St, Philadelphia, Pennsylvania, 19102
4                    Walnut St, Philadelphia, Pennsylvania, 19107
5                    Walnut St, Philadelphia, Pennsylvania, 19139
6                    Walnut St, Philadelphia, Pennsylvania, 19104
7                    Walnut St, Philadelphia, Pennsylvania, 19106
8                    Walnut St, Philadelphia, Pennsylvania, 19103
9   E State St & N Walnut St, Kennett Square, Pennsylvania, 19348
10             State Rd & Walnut Ln, Telford, Pennsylvania, 18969
11         State Rd & Walnut Ave E, Bensalem, Pennsylvania, 19020
12 E Street Rd & N Walnut Rd, Kennett Square, Pennsylvania, 19348
   location.x location.y score
1   -75.19868   39.95361 99.61
2   -75.70500   39.84916 92.11
3   -75.16610   39.94956 89.84
4   -75.15921   39.94872 89.84
5   -75.22935   39.95743 89.84
6   -75.19625   39.95331 89.84
7   -75.14849   39.94735 89.84
8   -75.17451   39.95063 89.84
9   -75.70560   39.84894 88.18
10  -75.32272   40.33865 87.85
11  -74.97302   40.06046 87.54
12  -75.70581   39.87565 87.23
                                             attributes.Match_addr
1        S 38th St & Walnut St, Philadelphia, Pennsylvania, 19104
2     E State St & Walnut St, Kennett Square, Pennsylvania, 19348
3                    Walnut St, Philadelphia, Pennsylvania, 19102
4                    Walnut St, Philadelphia, Pennsylvania, 19107
5                    Walnut St, Philadelphia, Pennsylvania, 19139
6                    Walnut St, Philadelphia, Pennsylvania, 19104
7                    Walnut St, Philadelphia, Pennsylvania, 19106
8                    Walnut St, Philadelphia, Pennsylvania, 19103
9   E State St & N Walnut St, Kennett Square, Pennsylvania, 19348
```

```
10             State Rd & Walnut Ln, Telford, Pennsylvania, 18969
11         State Rd & Walnut Ave E, Bensalem, Pennsylvania, 19020
12 E Street Rd & N Walnut Rd, Kennett Square, Pennsylvania, 19348
   attributes.Addr_type extent.xmin extent.ymin extent.xmax extent.ymax
1              StreetInt   -75.19968    39.95261   -75.19768    39.95461
2              StreetInt   -75.70600    39.84817   -75.70400    39.85016
3             StreetName   -75.16710    39.94856   -75.16510    39.95056
4             StreetName   -75.16021    39.94772   -75.15821    39.94972
5             StreetName   -75.23035    39.95643   -75.22835    39.95843
6             StreetName   -75.19725    39.95231   -75.19525    39.95431
7             StreetName   -75.14949    39.94635   -75.14749    39.94835
8             StreetName   -75.17551    39.94963   -75.17351    39.95163
9              StreetInt   -75.70660    39.84794   -75.70460    39.84994
10             StreetInt   -75.32372    40.33765   -75.32172    40.33965
11             StreetInt   -74.97402    40.05946   -74.97202    40.06146
12             StreetInt   -75.70681    39.87465   -75.70481    39.87665
```

To make geocoding using these two services a little more convenient, we can create two functions that automate the process of taking an address, filling in `%20` for spaces in the appropriate URL, and retrieving the JSON results from the geocoding service.

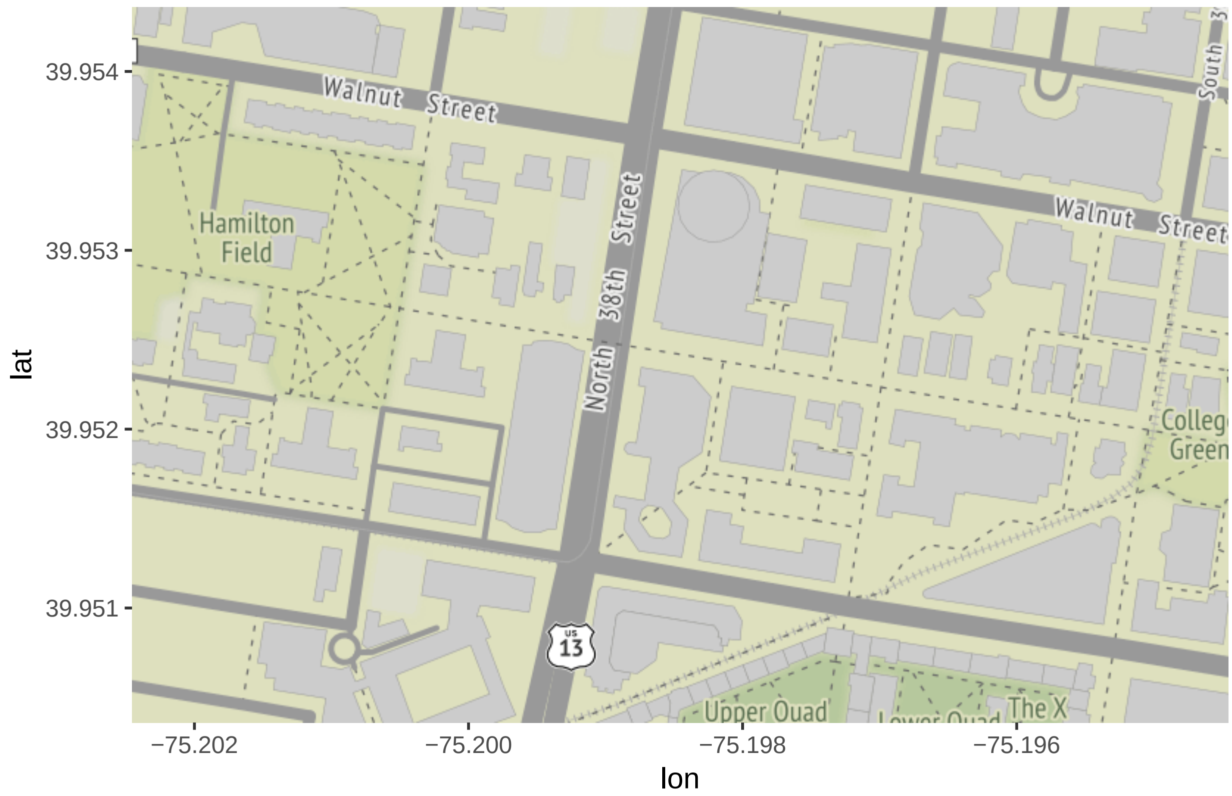
```r
geocodeOSM <- function(address)
{
   a <- gsub(" +", "\\%20", address)
   a <- paste0("http://nominatim.openstreetmap.org/search/",
               a,
               "?format=json&addressdetails=0&limit=1")
   return( fromJSON(a) )
}

geocodeARCGIS <- function(address)
{
   a <- gsub(" +", "\\%20", address)
   a <- paste0("https://geocode.arcgis.com/arcgis/rest/services/World/GeocodeServer/findAddressC
               a,
               "&outFields=Match_addr,Addr_type")
   return( fromJSON(a) )
}
```

Let's test out `geocodeOSM()` by pulling up a up a map of the geocoded coordinates.

```r
library(ggmap)
gcPenn <- geocodeOSM("3718 Locust Walk, Philadelphia, PA")
boundbox <- c(as.numeric(gcPenn$lon)-0.004,
              as.numeric(gcPenn$lat)-0.002,
              as.numeric(gcPenn$lon)+0.004,
              as.numeric(gcPenn$lat)+0.002)
b <-  get_map(location = boundbox,
              maptype = "terrain", # or "toner" or "watercolor"
```

```
                  source = "stamen")
ggmap(b)
```



This map shows a map of the western part of the Penn campus with 3718 Locust Walk (the square building near the center of the map) near the center of the map.

We are almost ready to throw all of our addresses at these geocoders, but let's first make sure the addresses look okay. Several OISs are missing locations.

```
i <- which(ois$location %in% c("","withheld","Withheld"))
ois$text[i]
```

```
 [1] "PS#1618\r\n5/31/16\r\nOn Tuesday, May 31, 2016, at approximately 1:12 PM, an off-
duty officer,\r\nin civilian attire, arrived home at his residence. Upon entering the front\r\nd
duty detective\r\napprehended the other offender near Brighton and Hawthorne Streets.\r\nThere w
 [2] "PS# 1626\r\n9/05/16\r\nOn Monday, September 5, 2016, at approximately 6:28 P.M., an off-
duty\r\nofficer, in plainclothes, became involved in a verbal and physical altercation\r\nwith h
Torresdale Hospital for treatment.\r\nThe officer's firearm, a .40 caliber semi-
automatic pistol, loaded with three\r\nlive rounds, was recovered at the scene.\r\nThere were no
 [3] "PS#10-06\r\n1/20/10\r\nOn 1/20/10, at approximately 7:36 P.M., plainclothes officers trave
 [4] "PS# 09-25\r\n3/21/09\r\nOn 3/21/09, at approximately 10:17 P.M., an officer, on-duty and i
 [5] "PS# 09-27\r\n4/7/09\r\nOn 4/7/09, at approximately 6:52 P.M., uniformed officers responded
 [6] "\t\r  \r\nPS# 09-76\r\n10/25/09\r\nOn 10/25/09, at approximately 2:15 A.M., a uniformed of
 [7] "PS# 08-06\r\n1/11/08\r\nOn 1/11/08, between approximately 8:50 PM and 9:20 PM, plainclothe
 [8] "PS# 08-18\r\n2/25/08\r\nOn 02/25/08, at approximately 11:06 AM, uniform officers responded
```

```
 [9] "PS# 08-30\r\n4/2/08\r\nOn 4/2/08, at approximately 8:32 P.M., uniformed officers responded
[10] "PS# 08-35\r\n5/1/08\r\nOn 5/1/08, at approximately 12:54 P.M., uniformed bicycle officers
[11] "PS# 08-40\r\n6/13/08\r\nOn 6-13-08, at approximately 8:42 P.M., plainclothes officers rece
[12] "PS# 08-60\r\n10/7/08\r\nOn 10/7/08, at approximately 2:46 P.M., uniformed officers respond
[13] "PS #08-70\r\n12/5/08\r\nOn 2/5/08, at approximately 4:25 P.M., a call was received at Poli
[14] "PS# 08-74\r\n12/27/08\r\nOn12/27/08, at approximately 11:14 A.M., uniformed officers respo
[15] "PS# 07-27\r\n4/21/07\r\nOn 4/21/07, at approximately 12:57 A.M., a uniformed officer stopp
```

Several of these text descriptions of the incidents contain the locaiton information. Let's fill those in

```
ois$location[ois$id=="16-18"] <- "3200 block of Wellington Street"
ois$location[ois$id=="10-06"] <- "Howard and Grange Street"
ois$location[ois$id=="08-06"] <- "200 block of Clapier Street"
ois$location[ois$id=="08-18"] <- "900 block of E. Slocum Street"
ois$location[ois$id=="08-30"] <- "700 block of W. Rockland Street"
ois$location[ois$id=="08-40"] <- "5400 Jefferson Street"
ois$location[ois$id=="08-60"] <- "3000 Memphis Street"
ois$location[ois$id=="08-70"] <- "1300 block of S. 29th Street"
ois$location[ois$id=="08-74"] <- "5600 block of N. Mascher Street"
```

While browsing locations sometimes you might come across ones that aren't quite right, this one for example. fixing.

```
# This one was just "51st Arch"
ois$location[ois$id=="07-19"] <- "51st St and Arch"
```

The text of this incident indicates that Philadelphia PD officers were not involved in the shooting.

```
ois$text[ois$id=="17-08"]
ois <- subset(ois, id != "17-08")
```

```
[1] "OIS# 1708 (March 29, 2017)\r\nOn Wednesday, March 29, 2017, at approximately 5:39 PM, two u
```

Several of the addresses are of the form "5400 block of Erdick St". I want these to get geocoded to the middle of the block. So I'm going to change addresses like these to be like "5450 Erdick St".

```
# put "blocks" at the midpoint
ois$location <- gsub("00 block( of)?", "50", ois$location, ignore.case=TRUE)
ois$location <- gsub("unit bl(oc)?k( of)?", "50", ois$location, ignore.case=TRUE)
# and one additional cleanup "Rear Alley of 300 block of N. 55th Street"
ois$location <- gsub("Rear Alley of |near ", "", ois$location, ignore.case = TRUE)
```

Now let's run all of the addresses through the OpenStreetMap geocoder. We could have geocoded all these addresses with the more simple code `lapply(a, geocodeOSM)`. However, if the JSON connection to the OpenStreetMap website 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 `gcOIS` still keeps all of the prior geocoding results and you can restart the for-loop at the point where it failed.

```
# add city and state to each address to improve geocoding accuracy
a <- paste0(ois$location,", Philadelphia, PA")
```

```r
# create a list to store the geocoding results
gcOIS <- vector("list", nrow(ois))
for(i in 1:nrow(ois))
{
   gcOIS[[i]] <- geocodeOSM(a[i])
   if(length(gcOIS[[i]]) == 0)
   {
      cat("Could not geocode address #",i,":",a[i],"\n")
   }
}
```

```
Could not geocode address # 9 : 6450 Lambert Street, Philadelphia, PA
Could not geocode address # 19 : 49th and Walnut Streets, Philadelphia, PA
Could not geocode address # 23 : Loudon and D streets, Philadelphia, PA
Could not geocode address # 25 : 5700 N. Park street/5700 N. Broad street, Philadelphia, PA
Could not geocode address # 26 : 50 Salford street, Philadelphia, PA
Could not geocode address # 29 : 16th Street and Allegheny Ave, Philadelphia, PA
Could not geocode address # 32 : Withheld, Philadelphia, PA
Could not geocode address # 35 : 4800, 4900 and 5150s of Sansom, Philadelphia, PA
Could not geocode address # 39 : 550 N. 56 Street, Philadelphia, PA
Could not geocode address # 45 : 32nd and Susquehanna Ave, Philadelphia, PA
Could not geocode address # 46 : A and Somerset Streets, Philadelphia, PA
Could not geocode address # 53 : 5050 W. Master, Philadelphia, PA
Could not geocode address # 57 : 1250 S. Carlisle, Philadelphia, PA
Could not geocode address # 71 : 650 Creighton Street, Philadelphia, PA
Could not geocode address # 85 : 7350 W. Passyunk Avenue, Philadelphia, PA
Could not geocode address # 88 : 2250 Delhi Street, Philadelphia, PA
Could not geocode address # 95 : 50 Reger Street, Philadelphia, PA
Could not geocode address # 97 : 2550 Sydenham Street, Philadelphia, PA
Could not geocode address # 99 : 22nd and Morris Streets, Philadelphia, PA
Could not geocode address # 101 : 650 Glenwood Street, Philadelphia, PA
Could not geocode address # 109 : 2450 North Edgely Street, Philadelphia, PA
Could not geocode address # 113 : Devon and Locust Streets, Philadelphia, PA
Could not geocode address # 115 : 2300 Oxford Street, Philadelphia, PA
Could not geocode address # 116 : Robbins Street and Castor Avenue, Philadelphia, PA
Could not geocode address # 123 : 5350 Columbia Avenue, Philadelphia, PA
Could not geocode address # 124 : 250 Montana Street, Philadelphia, PA
Could not geocode address # 125 : 1850 Bucknell Street, Philadelphia, PA
Could not geocode address # 126 : 5650 Lansdowne Avenue and 58th Street, Philadelphia, PA
Could not geocode address # 127 : 2200 Glenwood Avenue, Philadelphia, PA
Could not geocode address # 129 : Cambria and Warnock Streets, Philadelphia, PA
Could not geocode address # 130 : 4650 Frandford Avenue, Philadelphia, PA
Could not geocode address # 135 : Levick and Vandike Streets, Philadelphia, PA
Could not geocode address # 146 : 39th and Chestnut Sts, Philadelphia, PA
Could not geocode address # 148 : Margaret and Tackawanna Sts, Philadelphia, PA
Could not geocode address # 150 : Harbison Ave -Phila, Philadelphia, PA
Could not geocode address # 151 : 16th St and Lehigh Ave, Philadelphia, PA
```

```
Could not geocode address # 158 : 2100 N. 31st, Philadelphia, PA
Could not geocode address # 162 : 10th and Ontario Sts, Philadelphia, PA
Could not geocode address # 163 : 52nd and jefferson Sts, Philadelphia, PA
Could not geocode address # 167 : 1000 Norris St, Philadelphia, PA
Could not geocode address # 169 : 17th St and JFK Blvd, Philadelphia, PA
Could not geocode address # 172 : 2nd and Nedro Sts, Philadelphia, PA
Could not geocode address # 175 : 5th and York Sts, Philadelphia, PA
Could not geocode address # 178 : 65th St Phila, Philadelphia, PA
Could not geocode address # 180 : D St and Wyoming Ave, Philadelphia, PA
Could not geocode address # 182 : 9th St and Hunting Park Ave, Philadelphia, PA
Could not geocode address # 183 : 1300 Girard Ave, Philadelphia, PA
Could not geocode address # 185 : 500 Conestoga St, Philadelphia, PA
Could not geocode address # 186 : 32nd and Oxford Sts, Philadelphia, PA
Could not geocode address # 187 : 1900 Hemberger St, Philadelphia, PA
Could not geocode address # 188 : germantown Ave and Pike St, Philadelphia, PA
Could not geocode address # 189 : 4200 Whittakher Ave, Philadelphia, PA
Could not geocode address # 191 : 61st and Market Sts, Philadelphia, PA
Could not geocode address # 192 : 3400 N. Braddock St, Philadelphia, PA
Could not geocode address # 200 : 2300 Susquehanna Ave, Philadelphia, PA
Could not geocode address # 205 : 4500 Melrose Ave, Philadelphia, PA
Could not geocode address # 208 : 400 Sanger St, Philadelphia, PA
Could not geocode address # 215 : 2100 Monmouth St, Philadelphia, PA
Could not geocode address # 217 : 1400 Indiana Ave, Philadelphia, PA
Could not geocode address # 218 : 5100 Duffield Ave, Philadelphia, PA
Could not geocode address # 220 : 500 Lehigh Ave, Philadelphia, PA
Could not geocode address # 221 : 2800 Judson St, Philadelphia, PA
Could not geocode address # 223 : 1000 W Cumberland Ave, Philadelphia, PA
Could not geocode address # 225 : 5900 Kemble St, Philadelphia, PA
Could not geocode address # 230 : 9th and Pike Sts, Philadelphia, PA
Could not geocode address # 232 : Kiem and Ontario Sts, Philadelphia, PA
Could not geocode address # 236 : 2500 W Oxford, Philadelphia, PA
Could not geocode address # 237 : Broad and Mcferron Sts, Philadelphia, PA
Could not geocode address # 238 : 6000 W Oxford, Philadelphia, PA
Could not geocode address # 240 : 2600 Berks St, Philadelphia, PA
Could not geocode address # 243 : Howard and Grange Street, Philadelphia, PA
Could not geocode address # 244 : 21st and Spencer St, Philadelphia, PA
Could not geocode address # 254 : 1650 Mentor St, Philadelphia, PA
Could not geocode address # 255 : 60th Street and Springfield Ave, Philadelphia, PA
Could not geocode address # 262 : 2950 Girard Ave, Philadelphia, PA
Could not geocode address # 266 : 2250 Edgeley St, Philadelphia, PA
Could not geocode address # 270 : Cobbs Creek and Spruce St, Philadelphia, PA
Could not geocode address # 274 : B St and Ontario St, Philadelphia, PA
Could not geocode address # 275 : 5250 Marlow St, Philadelphia, PA
Could not geocode address # 281 : A St and Louden St, Philadelphia, PA
Could not geocode address # 285 : block of Lancaster Ave, Philadelphia, PA
Could not geocode address # 291 : Marston and Diamond St, Philadelphia, PA
Could not geocode address # 299 : 7300 N. 20th, Philadelphia, PA
Could not geocode address # 303 : Broad and Olney Ave, Philadelphia, PA
```

```
Could not geocode address # 304 : Kennsington Ave and Sommerset, Philadelphia, PA
Could not geocode address # 305 : 1000 W. Indiana St, Philadelphia, PA
Could not geocode address # 307 : 4200 Grissom, Philadelphia, PA
Could not geocode address # 311 : 16th Warton St, Philadelphia, PA
Could not geocode address # 316 : 5700 N. Mascher, Philadelphia, PA
Could not geocode address # 318 : 2400 N. Colorado, Philadelphia, PA
Could not geocode address # 328 : 1200 Hazzard St, Philadelphia, PA
Could not geocode address # 330 : Haverford Ave. and Sherwood Road, Philadelphia, PA
Could not geocode address # 338 : 2400 N. Bancroft S, Philadelphia, PA
Could not geocode address # 339 : 17th and Westmoreland St, Philadelphia, PA
Could not geocode address # 341 : 2900 Oakdale St, Philadelphia, PA
Could not geocode address # 343 : Oxford Ave and Benner St, Philadelphia, PA
Could not geocode address # 348 : Brown and Sloan St, Philadelphia, PA
Could not geocode address # 350 : 1800 S. Ringgold, Philadelphia, PA
Could not geocode address # 351 : 1100 Venango St, Philadelphia, PA
Could not geocode address # 358 : Gratz and Cumberland Streets, Philadelphia, PA
Could not geocode address # 361 : 53rd and Ludlow Streets, Philadelphia, PA
Could not geocode address # 364 : 27th and Cumberland Streets, Philadelphia, PA
Could not geocode address # 367 : 30th Street and Lehigh Ave, Philadelphia, PA
Could not geocode address # 375 : 2100 Norris Street, Philadelphia, PA
Could not geocode address # 377 : Greenway Ave and Avondale St, Philadelphia, PA
Could not geocode address # 379 : 450 Hobart St, Philadelphia, PA
Could not geocode address # 380 : Broad and Thompson St, Philadelphia, PA
Could not geocode address # 381 : Broad and Ruscomb Streets, Philadelphia, PA
Could not geocode address # 384 : 1250 Oakdale Street, Philadelphia, PA
Could not geocode address # 387 : 56th Street at Media Street, Philadelphia, PA
Could not geocode address # 390 : Lawrence and Cambria Streets, Philadelphia, PA
Could not geocode address # 392 : 23rd and Tasker Streets, Philadelphia, PA
Could not geocode address # 395 : 4400 N 17 ST, Philadelphia, PA
Could not geocode address # 396 : 2400 N 10 ST, Philadelphia, PA
Could not geocode address # 397 : 1700 N 59 ST, Philadelphia, PA
Could not geocode address # 404 : 5100 S 11 ST, Philadelphia, PA
Could not geocode address # 405 : 13th and Pike Streets, Philadelphia, PA
Could not geocode address # 406 : Wayne Avenue and Apsley St, Philadelphia, PA
Could not geocode address # 411 : 51st St and Arch, Philadelphia, PA
Could not geocode address # 413 : 250 W. Shawmont St, Philadelphia, PA
Could not geocode address # 414 : 1900 Gleenwood Ave, Philadelphia, PA
Could not geocode address # 418 : 54th Wyalusing Ave, Philadelphia, PA
Could not geocode address # 427 : 1650 Taney Street, Philadelphia, PA
Could not geocode address # 428 : 52nd Street and Greenway Ave, Philadelphia, PA
Could not geocode address # 433 : 6850 N. 19th, Philadelphia, PA
Could not geocode address # 434 : Watts St and Cambridge St, Philadelphia, PA
Could not geocode address # 438 : Kensington Ave and York St, Philadelphia, PA
Could not geocode address # 440 : 2350 W. Passyunk Ave, Philadelphia, PA
Could not geocode address # 443 : 22nd and Sansom St, Philadelphia, PA
Could not geocode address # 445 : Summerdale Avenue and Augusta St, Philadelphia, PA
```

Geocoding failed on several addresses. Some of them are fairly predictable as they are formatted poorly (e.g. Harbison Ave -Phila, Philadelphia, PA). Many others fail because they are intersections and OpenStreeMap does not handle intersections (yet). So let's send the addresses that OpenStreetMap could not geocode to the ArcGIS geocoder.

```
for(i in which(sapply(gcOIS, length)==0))
{
   gcOIS[[i]] <- geocodeARCGIS(a[i])
}
```

Now `gcOIS` has some of the geocoding results from OpenStreetMap and some from ArcGIS. The results are in different formats. The ArcGIS results have a component named `candidates`. So we'll use the presence/absence of a `candidates` component to figure out which geocoding service delivered the results. Then we can extract the longitude, latitude, and some additional features. The ArcGIS geocoder gives numerous results, but we will just take the top scoring one, which is the most likely match.

```
b <- lapply(gcOIS, function(x)
{
   if(is.null(x$candidates)) # OSM
   {
      a <- data.frame(lon=as.numeric(x$lon),
                      lat=as.numeric(x$lat),
                      score=as.numeric(x$importance),
                      loctype=paste(x$class, x$type, sep=":"),
                      method="osm",
                      addressGeo=x$display_name,
                      stringsAsFactors = FALSE)
   } else # ArcGIS
   {
      a <- data.frame(lon=x$candidates$location[1,"x"],
                      lat=x$candidates$location[1,"y"],
                      score=x$candidates$score[1],
                      loctype=x$candidates$attributes$Addr_type[1],
                      method="arcgis",
                      addressGeo=x$candidates$attributes$Match_addr[1],
                      stringsAsFactors = FALSE)
   }
   return(a)
})
gcOIS <- do.call(rbind, b)
# add a column containing the original address
gcOIS <- cbind(gcOIS, addressOrig = a)
head(gcOIS)
```

```
        lon      lat score     loctype method
1 -75.12119 39.99229 0.521 place:house    osm
2 -75.17101 39.91406 0.521 place:house    osm
3 -75.18356 40.00508 0.521 place:house    osm
4 -75.07958 40.03548 0.421 place:house    osm
```

```
5 -75.16452 40.02539 0.421 place:house    osm
6 -75.18072 39.92571 0.421 place:house    osm

1                    2850, Kensington Avenue, Tioga Park, Philadelphia, Philadelphia County, Pe
2    1350, Bigler Street, Packer Park, South Philadelphia, Philadelphia, Philadelphia County, Pe
3                3150, North 33rd Street, Allegheny West, Philadelphia, Philadelphia County, Pe
4                        1450, Lardner Street, Frankford, Philadelphia, Philadelphia County, Pe
5                      4850, Knox Street, Wayne Junction, Philadelphia, Philadelphia County, Pe
6 2050, Snyder Avenue, Girard Estates, South Philadelphia, Philadelphia, Philadelphia County, Pe
                               addressOrig
1 2850 Kensington Avenue, Philadelphia, PA
2     1350 Bigler Street, Philadelphia, PA
3    3150 N. 33rd Street, Philadelphia, PA
4    1450 Lardner Street, Philadelphia, PA
5       4850 Knox Street, Philadelphia, PA
6     2050 Snyder Avenue, Philadelphia, PA
```

Now it appears that we have longitude and latitude for every incident. We should check that they all look sensible.

```r
stem(gcOIS$lat)
stem(gcOIS$lon)
```

```

  The decimal point is at the |

  32 | 88
  33 |
  33 |
  34 |
  34 |
  35 |
  35 |
  36 |
  36 |
  37 |
  37 |
  38 |
  38 |
  39 |
  39 | 99999999999999999999999999999999999999999999999999999999999999999999
  40 | 00000000000000000000000000000000000000000000000000000000000000000000+302


  The decimal point is at the |

  -89 | 11
  -88 |
  -87 |
```

```
  -86 |
  -85 |
  -84 |
  -83 |
  -82 |
  -81 |
  -80 |
  -79 |
  -78 |
  -77 |
  -76 |
  -75 | 33333222222222222222222222222222222222222222222222222222222222222222+374
```

While almost all the points have latitude around 39 and 40 and longitude around -75, several incidents appear to occur at (-89.11, 32.88). Look that up on Google Maps and you will find Philadelphia, Mississippi.

```r
subset(gcOIS, lon < -80)
```

```
          lon      lat score            loctype method
133 -89.10462 32.76987  0.41 highway:residential    osm
374 -89.10462 32.76987  0.41 highway:residential    osm
                                                           addressGeo
133 Jefferson Street, Philadelphia, Neshoba County, Mississippi, 39350, USA
374 Jefferson Street, Philadelphia, Neshoba County, Mississippi, 39350, USA
                                addressOrig
133 5250 Jefferson Street, Philadelphia, PA
374 5400 Jefferson Street, Philadelphia, PA
```

Maybe the ArcGIS geocoder can do a better job.

```r
i <- which(gcOIS$lon < -80)
b <- lapply(gcOIS$addressOrig[i], geocodeARCGIS)

b <- lapply(b, function(x)
{
   data.frame(lon=x$candidates$location[1,"x"],
              lat=x$candidates$location[1,"y"],
              score=x$candidates$score[1],
              loctype=x$candidates$attributes$Addr_type[1],
              method="arcgis",
              addressGeo=x$candidates$attributes$Match_addr[1],
              stringsAsFactors = FALSE)
})
b <- do.call(rbind, b)
# replace just some of the columns with the values in b
gcOIS[i, names(b)] <- b
gcOIS[i,]
```

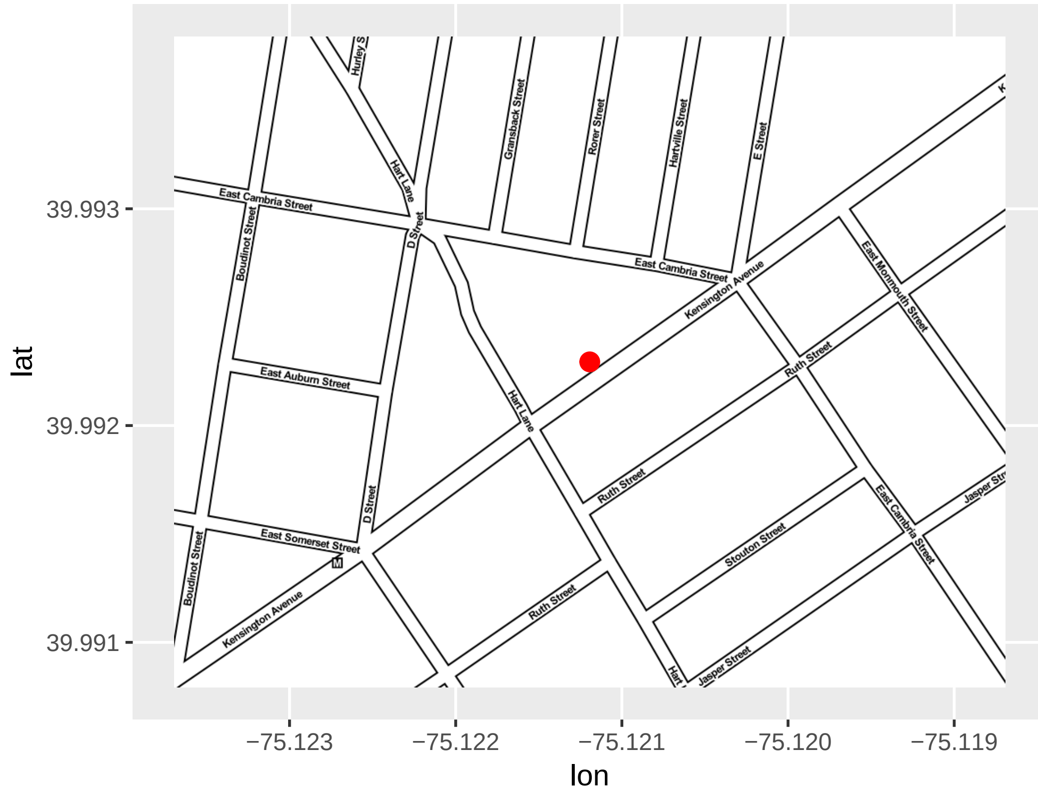
```
          lon      lat score      loctype method
```

```
133 -75.22697 39.98000  99.5  PointAddress arcgis
374 -75.23095 39.97906  99.5 StreetAddress arcgis
                                                    addressGeo
133 5250 W Jefferson St, Philadelphia, Pennsylvania, 19131
374 5400 W Jefferson St, Philadelphia, Pennsylvania, 19131
                                   addressOrig
133 5250 Jefferson Street, Philadelphia, PA
374 5400 Jefferson Street, Philadelphia, PA
```

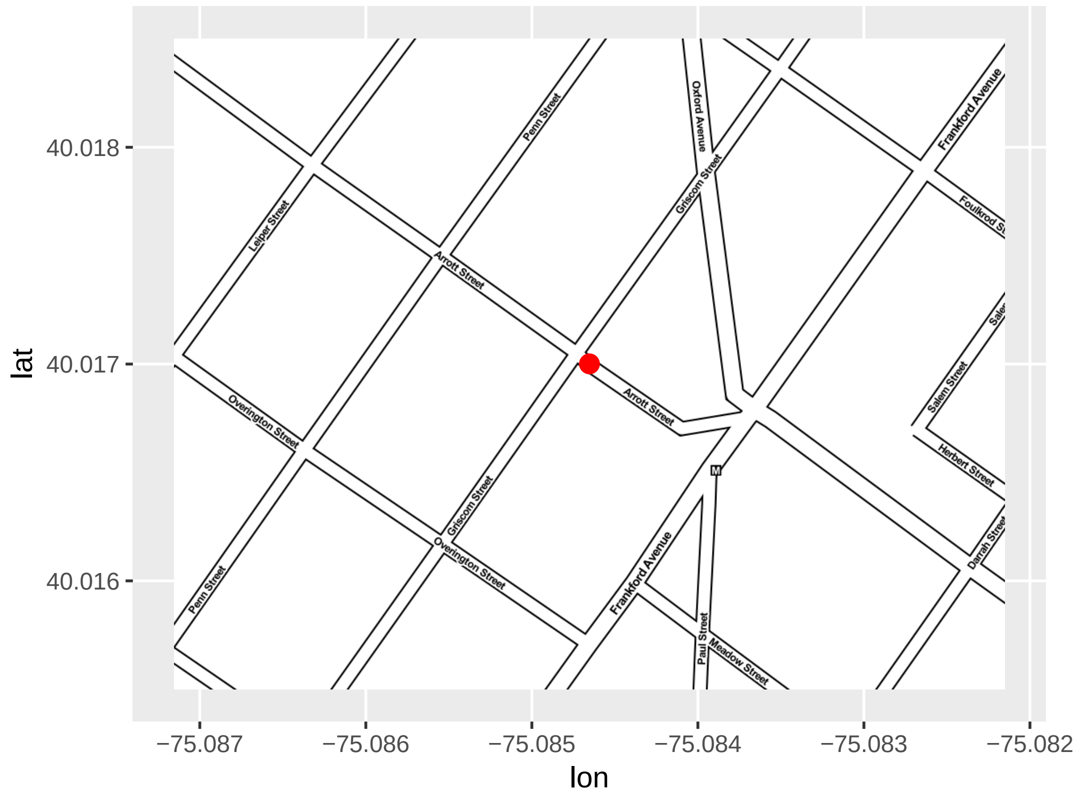
There's no perfect way to check the geocoding results aside from checking each individual point and verify with a map that it lands on the right spot. We can do things like check for strange outliers like we did just now to find that some incidents were mapped to Mississippi. We can spot check a few locations with a map.

```
iMap <- which(ois$id %in% c("18-01","13-14","07-13"))
for(i in iMap)
{
   boundbox <- with(gcOIS[i,],
                    c(lon-0.0025, lat-0.0015, lon+0.0025, lat+0.0015))
   b <-  get_map(location = boundbox,
                 scale = "auto",
                 maptype = "toner",
                 source = "stamen")
   print(ggmap(b,extent="normal") +
            geom_point(aes(x=lon,y=lat), color="red", size=3, data=gcOIS[i,]) +
            ggtitle(ois$location[i]))
}
```

2850 Kensington Avenue



Arrott Street Frankford Avenue



The first and last of these were geocoded correctly. I looked up 2850 Kensington in Google Maps to verify its location and you can see that the red dot in the last map is right at the intersection of 13th and Pike. However, the middle one is off by a bit. You can see Arrott St and Frankford St, but the dot is not at their intersection. The address shown in the map title is missing an "and" between "Arrott Street" and "Frankford Avenue". So where did geocoding place out point?

```
gcOIS[iMap,]
```

```
          lon      lat  score             loctype method
1   -75.12119 39.99229  0.521         place:house    osm
107 -75.08465 40.01700  0.520 highway:residential    osm
405 -75.14837 40.01204 98.650           StreetInt arcgis
                                                                                                                   addres
1   2850, Kensington Avenue, Tioga Park, Philadelphia, Philadelphia County, Pennsylvania, 19134,
107            Arrott Street, Frankford, Philadelphia, Philadelphia County, Pennsylvania, 19124,
405                                           N 13th St & W Pike St, Philadelphia, Pennsylvania, 1
                                          addressOrig
1           2850 Kensington Avenue, Philadelphia, PA
107 Arrott Street Frankford Avenue, Philadelphia, PA
405           13th and Pike Streets, Philadelphia, PA
```

Note that the `loctype` for the first address is "house" and `loctype` is "StreetInt" for the last address. Those are indications that the geocoding was accurate to very specific locations. But for the second address the `loctype` is "residential". It found Arrott St in the Frankford neighborhood of Philadelphia, because OpenStreetMap did not understand that we were looking for an intersection.

Ideally we want geocoding to get us to a house, intersection, or address. If `loctype` is residential, locality, city, or streetname then this indicates that the geocoding did not get us to a very specific location.

```
sort(table(gcOIS$loctype))
```

```
           Locality   office:government    shop:convenience 
                  1                   1                   1 
    highway:service     railway:station        building:yes 
                  2                   2                   3 
      highway:trunk place:neighbourhood   highway:secondary 
                  3                   3                   5 
         place:city          StreetName    highway:tertiary 
                  6                   6                   8 
    highway:primary highway:residential        PointAddress 
                 10                  15                  26 
      StreetAddress           StreetInt         place:house 
                 42                  57                 265 
```

Several of these are very specific locations (office, shop, building, station, house, PointAddress, StreetAddress, StreetInt). Many others are not specific at all (highway, neighborhood, city, Locality, StreetName). Each of these needs to be revisited. Let's examine one these here. Let's look at those geocodeddown to `loctype="Streetname"`.

```
i <- which(gcOIS$loctype=="StreetName")
gcOIS[i,]
```

```
          lon      lat  score    loctype method
150 -75.06512 40.01764 100.00 StreetName arcgis
178 -75.23325 39.92467  99.46 StreetName arcgis
232 -75.10595 39.99590  85.86 StreetName arcgis
285 -75.20031 39.96204 100.00 StreetName arcgis
311 -75.15674 39.93270  86.58 StreetName arcgis
418 -75.21110 39.97135  95.33 StreetName arcgis
                                          addressGeo
150  Harbison Ave, Philadelphia, Pennsylvania, 19135
178      S 65th St, Philadelphia, Pennsylvania, 19142
232   E Ontario St, Philadelphia, Pennsylvania, 19134
285  Lancaster Ave, Philadelphia, Pennsylvania, 19104
311     Wharton St, Philadelphia, Pennsylvania, 19147
418  Wyalusing Ave, Philadelphia, Pennsylvania, 19104
                                  addressOrig
150    Harbison Ave -Phila, Philadelphia, PA
178           65th St Phila, Philadelphia, PA
232   Kiem and Ontario Sts, Philadelphia, PA
285 block of Lancaster Ave, Philadelphia, PA
311          16th Warton St, Philadelphia, PA
418      54th Wyalusing Ave, Philadelphia, PA
```

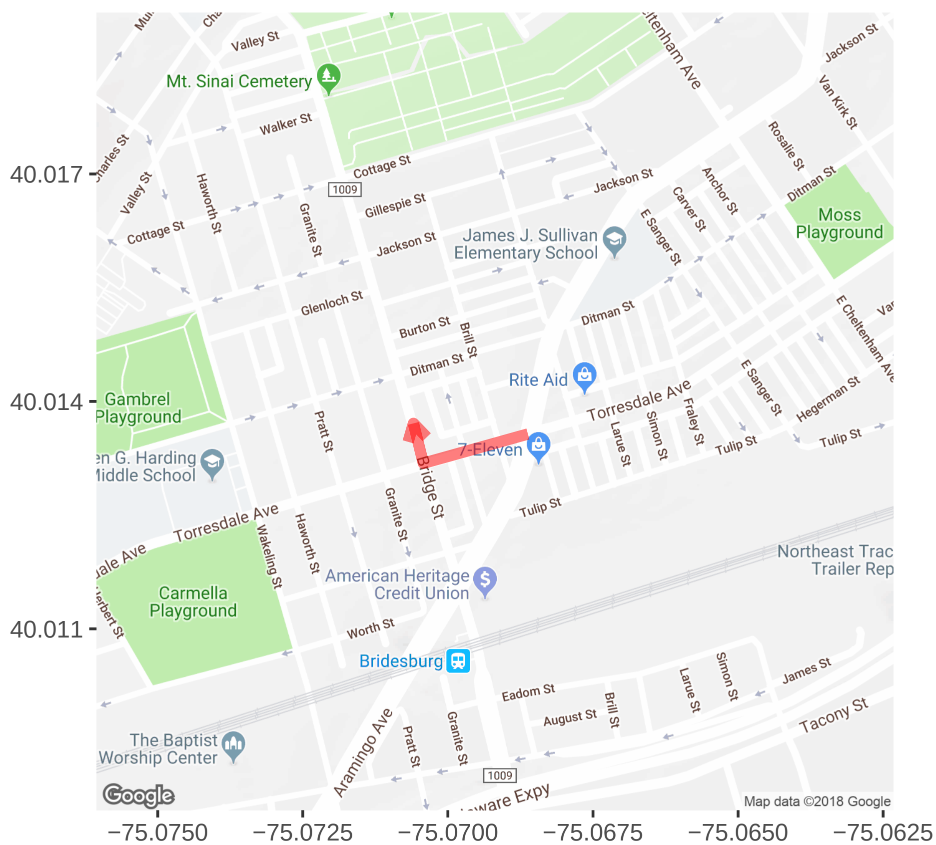
We can see that the problem for many of these is simply bad addresses. Some are intersections that are missing the "and" between the two streets. Let's read into the incident at Harbison Ave to see if we can learn more.

```
ois$text[ois$location=="Harbison Ave -Phila"]
```

```
[1] "\t\r  \r\n\t\r  \r\nPS#12-20\r\n3/22/12\r\nOn 3/22/12, at approximately 12:42 AM, uniformed
```

The text describes the path of the suspect moving south on Torresdale and turning west on Bridge. Let's pull up a map of this area. I've overlayed the path of the suspect on top of the map. The officers describe the suspect moving "south" on Torresdale and "west" on Bridge because the I95 and the Delaware River are just south of this area and run more east-west here. So thinking that I95 run north-south and that the Delaware River is the east boundary of Philadelphia, officers may decribe someone running "west" on Bridge St as running away from I95 and the Delaware River.

```
ggmap(get_map(c(-75.0691987,40.0138755), zoom = 16)) +
   geom_path(aes(x=x, y=y),
             data=data.frame(x=c(-75.068621,-75.070423,-75.070594),y=c(40.013570,40.013184,40.01
             color="red", size=2, alpha=0.5,
             arrow=arrow(length=unit(0.1,"inches"))) +
   theme(plot.title = element_blank(),
         axis.title.x = element_blank(),
         axis.title.y = element_blank())
```



It took some investigation, but we can fix this address too. Here's how we can fix the two addresses that we've identified fixes for so far. The rest are left as an exercise. `textConnection()` is a nice trick for making a small data frame right inside a script. I'm going to make two columns, one with the original address and one with the correction or improved address.

```
a <- textConnection(
"addressOrig,addressFix
Arrott Street Frankford Avenue,Arrott Street and Frankford Avenue
Harbison Ave -Phila,Bridge St and Ditman St")
a <- read.csv(a, stringsAsFactors = FALSE)
# make sure the original addresses match, no NAs!
i <- match(a$addressOrig, ois$location)
i
```

```
[1] 107 150
```

```
b <- lapply(paste0(a$addressFix,", Philadelphia, PA"), geocodeARCGIS)
b <- lapply(b, function(x)
{
   data.frame(lon=x$candidates$location[1,"x"],
              lat=x$candidates$location[1,"y"],
              score=x$candidates$score[1],
              loctype=x$candidates$attributes$Addr_type[1],
              method="arcgis",
              addressGeo=x$candidates$attributes$Match_addr[1],
              stringsAsFactors = FALSE)
})
b <- do.call(rbind, b)
gcOIS[i,names(b)] <- b
gcOIS[i,]
```

```
          lon      lat score   loctype method
107 -75.08367 40.01676   100 StreetInt arcgis
150 -75.07084 40.01434   100 StreetInt arcgis
                                                       addressGeo
107 Arrott St & Frankford Ave, Philadelphia, Pennsylvania, 19124
150     Bridge St & Ditman St, Philadelphia, Pennsylvania, 19124
                                          addressOrig
107 Arrott Street Frankford Avenue, Philadelphia, PA
150            Harbison Ave -Phila, Philadelphia, PA
```

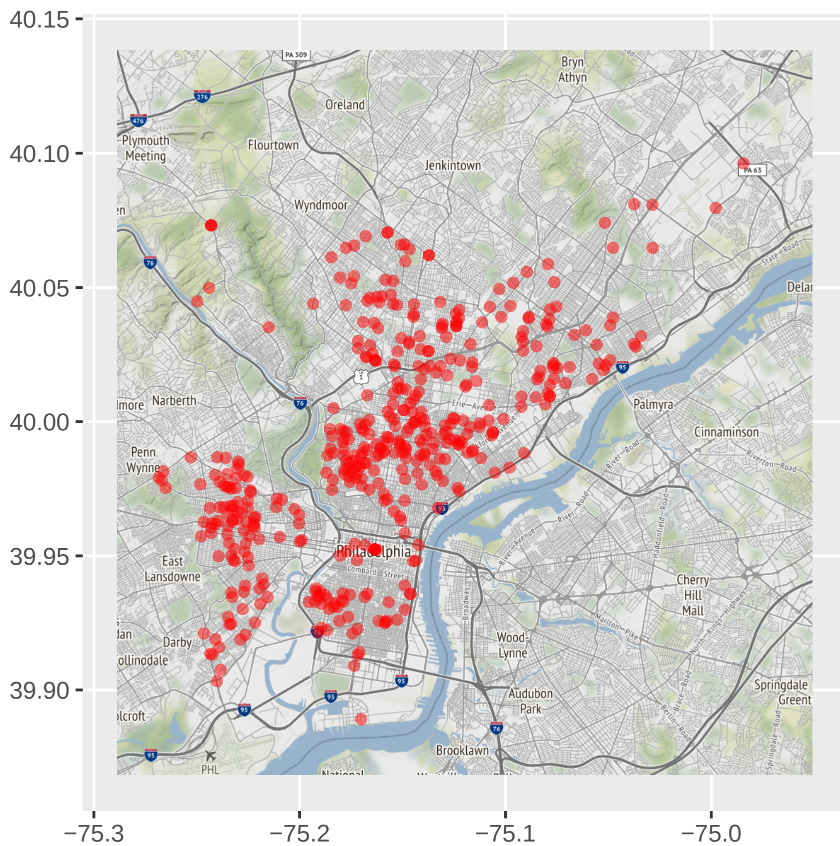
Both appear to be fixed. `addressGeo` appears correct and the `loctype` is now `StreetInt`. All good signs.

Let's create new longitude/latitude columns in our `ois` data frame so that one object contains all of our essential data.

```
ois$lon <- gcOIS$lon
ois$lat <- gcOIS$lat
```

We'll proceed as if we're satisfied with our geocoding even though you know you have more work to do to fix some of those non-specific geocoding problems. We close this section with a map of the city of Philadelphia and the locations of all officer involved shootings.

```r
mapPhilly <- get_map(c(-75.288486,39.868285,-74.950965,40.138251),
                     source ="stamen")
ggmap(mapPhilly, extent="normal") +
  geom_point(aes(x=lon,y=lat), data=gcOIS,
             color="red",
             alpha=0.5) +
   theme(axis.title.x = element_blank(),
         axis.title.y = element_blank())
```



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.

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

```
Reading layer `Boundaries_PSA' from data source `Z:\Penn\CRIM602\notes\R4crim\10_shapefiles_and_
Simple feature collection with 66 features and 10 fields
geometry type:  POLYGON
dimension:      XY
```

```
bbox:           xmin: -75.28031 ymin: 39.86701 xmax: -74.95575 ymax: 40.13793
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

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

```
PPDmap[,c("PSA_NUM","geometry")]
```

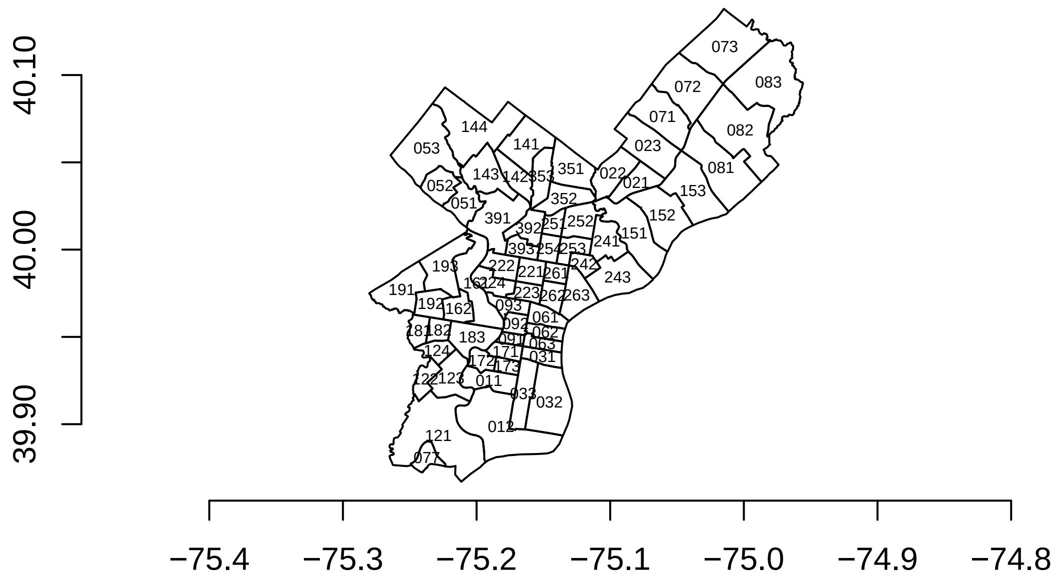
```
Simple feature collection with 66 features and 1 field
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: -75.28031 ymin: 39.86701 xmax: -74.95575 ymax: 40.13793
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
First 10 features:
   PSA_NUM                       geometry
1      077 POLYGON ((-75.2338 39.88977...
2      011 POLYGON ((-75.19724 39.9294...
3      012 POLYGON ((-75.17305 39.9105...
4      021 POLYGON ((-75.05888 40.0405...
5      022 POLYGON ((-75.08306 40.0454...
6      023 POLYGON ((-75.05773 40.0416...
7      051 POLYGON ((-75.21618 40.0415...
8      052 POLYGON ((-75.2192 40.04439...
9      053 POLYGON ((-75.21215 40.0476...
10     061 POLYGON ((-75.13226 39.9580...
```

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 the polygons to make a plot.

```
plot(st_geometry(PPDmap))
axis(side=1) # add x-axis
axis(side=2) # add y-axis
# extra the center points of each PSA
a <- st_coordinates(st_centroid(st_geometry(PPDmap)))
```

```
Warning in st_centroid.sfc(st_geometry(PPDmap)): st_centroid does not give
correct centroids for longitude/latitude data
```

```
# add the PSA number to the plot
text(a[,1], a[,2], PPDmap$PSA_NUM, cex=0.5)
```



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

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

```
             X        Y L1 L2
[1,] -75.23380 39.88977  1  1
[2,] -75.23380 39.88976  1  1
[3,] -75.23343 39.88919  1  1
[4,] -75.23325 39.88893  1  1
[5,] -75.23278 39.88815  1  1
[6,] -75.23223 39.88710  1  1
```

```
tail(a)
```

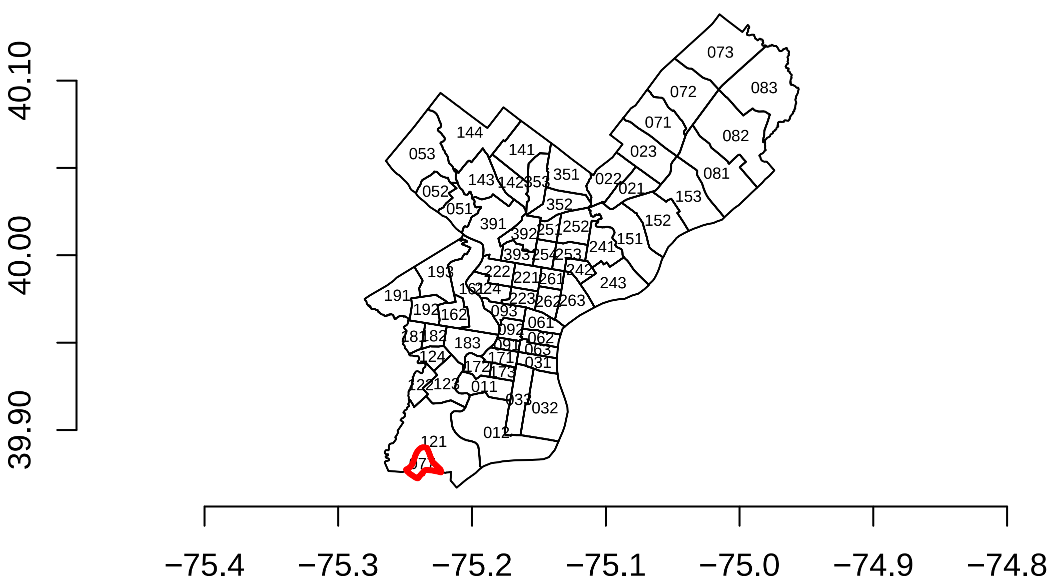
```
               X        Y L1 L2
[135,] -75.23675 39.89006  1  1
[136,] -75.23639 39.89008  1  1
[137,] -75.23594 39.89007  1  1
[138,] -75.23539 39.89001  1  1
[139,] -75.23503 39.88996  1  1
[140,] -75.23380 39.88977  1  1
```

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

```r
plot(st_geometry(PPDmap))
axis(side=1)
axis(side=2)
a <- st_coordinates(st_centroid(st_geometry(PPDmap)))
```

```
Warning in st_centroid.sfc(st_geometry(PPDmap)): st_centroid does not give
correct centroids for longitude/latitude data
```

```r
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)
```



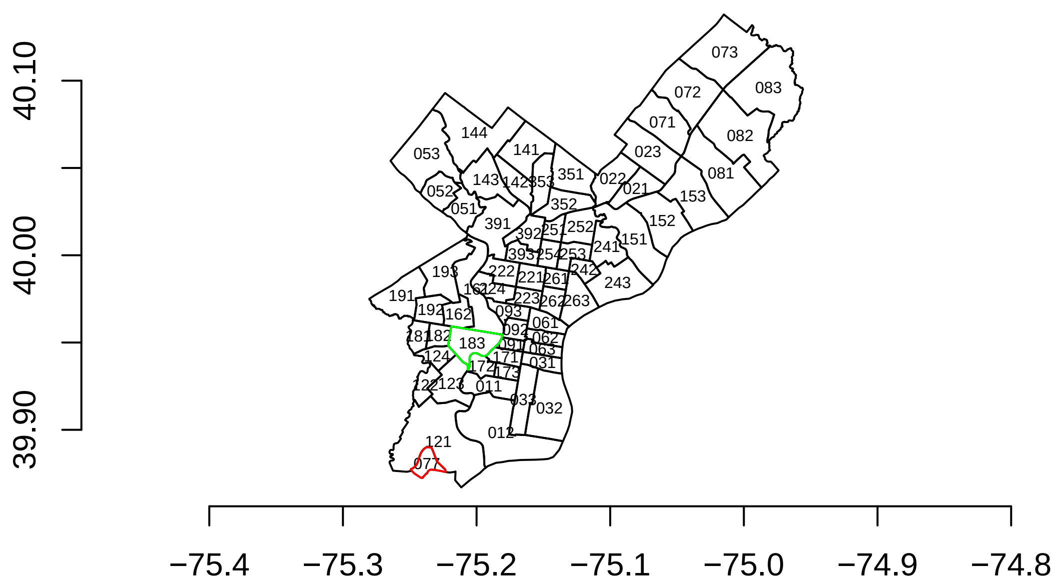
So this highlighted in red PSA 77 in the southern end of Philadelphia. Note that R issued some warnings about our centroid locations. We will return to that in a moment.

Rather than extracting coordinates to add a feature to a plot, `subset()` provides an easier method.

```r
plot(st_geometry(PPDmap))
axis(side=1)
axis(side=2)
a <- st_coordinates(st_centroid(st_geometry(PPDmap)))
```

```
Warning in st_centroid.sfc(st_geometry(PPDmap)): st_centroid does not give
correct centroids for longitude/latitude data
```

```
text(a[,1], a[,2], PPDmap$PSA_NUM, cex=0.5)
plot(st_geometry(subset(PPDmap, PSA_NUM=="077")),
     add=TRUE, border="red")
plot(st_geometry(subset(PPDmap, PSA_NUM=="183")),
     add=TRUE, border="green")
```



Setting `add=TRUE` in the last two calls to `plot()` asks R to overlay the current plot with these additional objects.

Now, back to those warnings we received about calculating centroids with longitude/latitude data. 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:
  EPSG: 4326
  proj4string: "+proj=longlat +datum=WGS84 +no_defs"
```

The `proj4string` tells us that the coordinate system used to describe the PPD boundaries is longitude/latitude. Specifically, it uses 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]((http://spatialreference.org/ref/epsg/4326/). You can find the full collection of coordinate systems at spatialreference.org.

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, OpenStreetMaps) use the pseudo-Mercator projection (EPSG3857). Let's convert our PPD map to that coordinate system.

```
PPDmap <- st_transform(PPDmap, crs=st_crs("+init=epsg:3857"))
st_crs(PPDmap)
```

```
Coordinate Reference System:
  EPSG: 3857
  proj4string: "+proj=merc +lon_0=0 +lat_ts=0 +x_0=0 +y_0=0 +a=6378137 +b=6378137 +nadgrids=@nul
```

Note that the `proj4string` now indicates that this is a Mercator projection with distance measured in meters (`+units=m`). Now if we ask for the centroids of the PSAs, we get more accurate centroids and no warnings from R.

```
st_centroid(st_geometry(PPDmap))
```

```
Geometry set for 66 features
geometry type:  POINT
dimension:      XY
bbox:           xmin: -8377443 ymin: 4848633 xmax: -8346907 ymax: 4882882
epsg (SRID):    3857
proj4string:    +proj=merc +lon_0=0 +lat_ts=0 +x_0=0 +y_0=0 +a=6378137 +b=6378137 +nadgrids=@nul
First 5 geometries:
```

We can use the same projection, but modify it so that distances are measured in feet.

```
PPDmap <- st_transform(PPDmap, crs=st_crs("+init=epsg:3857 +units=us-ft"))
st_crs(PPDmap)
```

```
Coordinate Reference System:
  No EPSG code
  proj4string: "+proj=merc +lon_0=0 +lat_ts=0 +x_0=0 +y_0=0 +a=6378137 +b=6378137 +nadgrids=@nul
```

There is a special coordinate system 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=st_crs("+init=epsg:2272"))
st_crs(PPDmap)
```

```
Coordinate Reference System:
  EPSG: 2272
  proj4string: "+proj=lcc +lat_1=40.96666666666667 +lat_2=39.93333333333333 +lat_0=39.3333333333
```

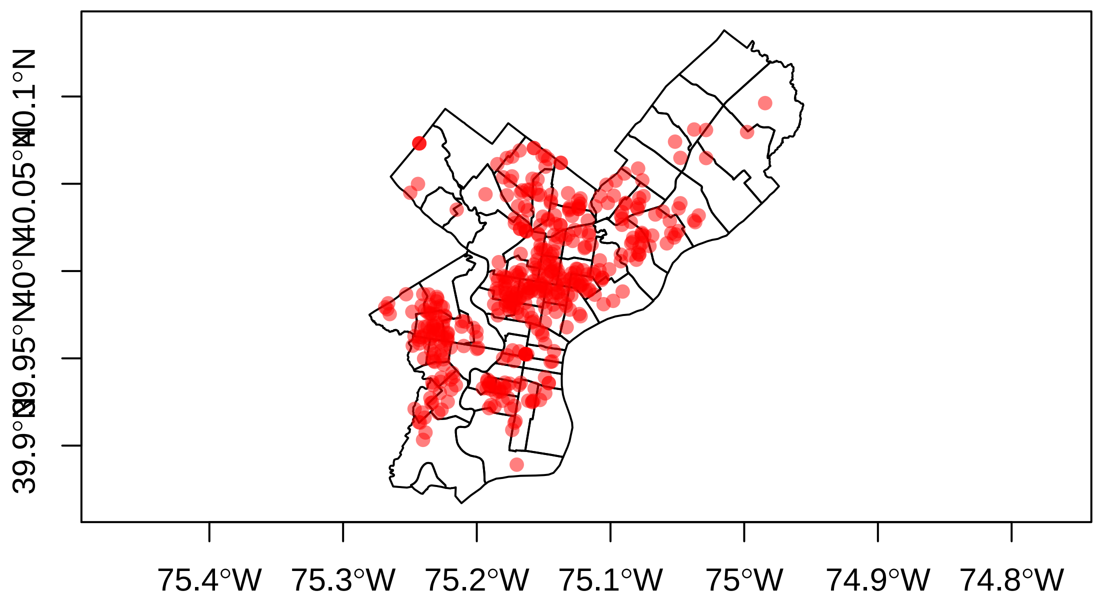
This coordinate system is the Lambert Conic Conformal (LCC). This particular projection of the `PPDmap` is tuned to provide good precision for the southern part of Pennsylvania and distances are measured in feet (note the `+units=us-ft` tag in the `proj4string`).

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. Also at the scale of Philadelphia, we're just using the centroid calculation to figure out where to put labels.

```
PPDmap <- st_transform(PPDmap, crs=st_crs("+init=epsg:4326"))
```

Now both PPD data and polygons are on the same scale

```
plot(st_geometry(PPDmap), axes=TRUE)
points(lat~lon, data=gcOIS, col=rgb(1,0,0,0.5), pch=16)
```



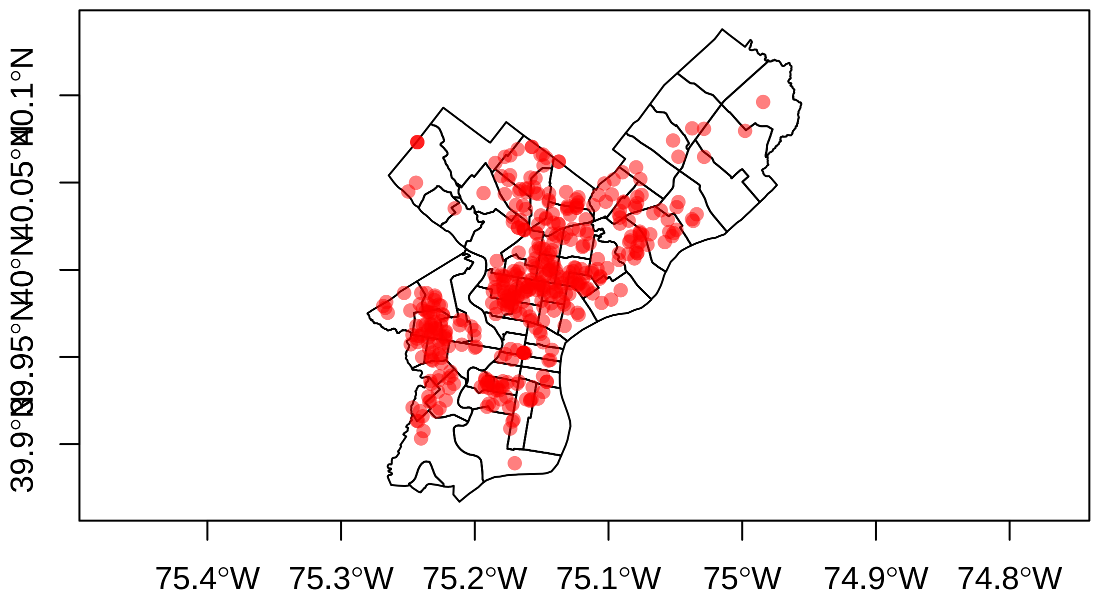
To make the dots a little transparent, I've used the `rgb()` function with which you can mix red, green, and blue colors and set the transparency. The `1` tells `rgb()` to use maximum red. The two `0`s tell `rgb()` to use no green or blue. The 0.5 tells `rgb()` to make the dots halfway transparent.

Spatial joins

Spatial joins 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=st_crs("+init=epsg:4326"))
# check that everything looks okay
plot(st_geometry(PPDmap), axes=TRUE)
plot(st_geometry(ois), add=TRUE, col=rgb(1,0,0,0.5), pch=16)
```



st_join() will match each row in ois to each polygon in PSA. I just want the PSA_NUM column out of the PPDmap.

```
PSAlookup <- st_join(ois, PPDmap[,"PSA_NUM"])
PSAlookup[1:3, c("id","date","location","PSA_NUM","geometry")]
```

```
Simple feature collection with 3 features and 4 fields
geometry type:  POINT
dimension:      XY
bbox:           xmin: -75.18356 ymin: 39.91406 xmax: -75.12119 ymax: 40.00508
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
     id       date              location PSA_NUM
1 18-01 2018-01-13 2850 Kensington Avenue     242
2 18-02 2018-01-29      1350 Bigler Street     033
3 18-08 2018-04-18    3150 N. 33rd Street     391
```

```
                    geometry
1 POINT (-75.12119 39.99229)
2 POINT (-75.17101 39.91406)
3 POINT (-75.18356 40.00508)
```

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

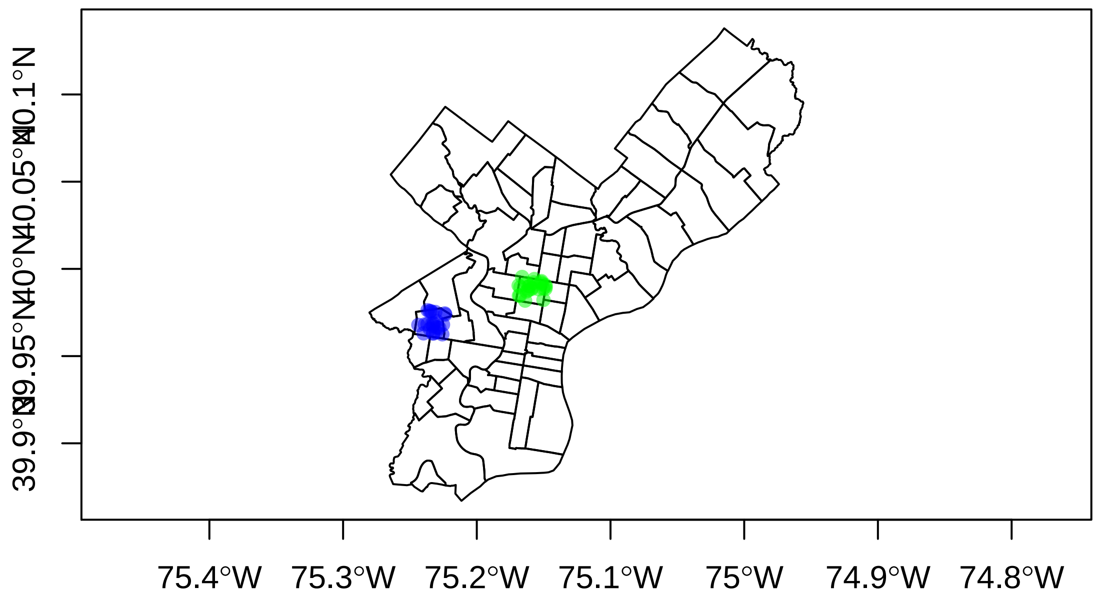
Let's examine the PSAs with the most OISs and highlight them on the map.

```
a <- rev(sort(table(PSAlookup$PSA)))
a
```

```

221 192 222 151 254 141 352 242 182 351 224 391 253 252 172 152 123 021
 24  23  21  18  15  14  13  12  12  11  11  10  10  10  10  10  10  10
392 261 162 393 241 193 191 173 142 092 033 353 251 223 181 022 011 263
  9   9   9   8   8   8   8   8   8   8   8   7   7   7   7   7   7   6
122 262 153 053 243 124 121 061 183 161 091 031 012 143 082 071 063 062
  6   5   5   5   4   4   4   4   3   3   3   3   3   2   2   2   2   2
032 023 144 083 081 072 051 171 093 077 073 052
  2   2   1   1   1   1   1   0   0   0   0   0
```

```
plot(st_geometry(PPDmap), axes=TRUE)
# plot the OISs in the PSA with the most OISs
i <- which(PSAlookup$PSA_NUM==names(a)[1])
plot(st_geometry(PSAlookup[i,]), add=TRUE, col=rgb(0,1,0,0.5), pch=16)
# plot the OISs in the PSA with the second most OISs
i <- which(PSAlookup$PSA_NUM==names(a)[2])
plot(st_geometry(PSAlookup[i,]), add=TRUE, col=rgb(0,0,1,0.5), pch=16)
```



Let's identify which OISs occurred in the same PSA as the University of Pennsylvania. We've already geocoded Penn and have its coordinates.

```
gcPenn
```

```
   place_id
1 228278154
                                                                    licence
1 Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright
  osm_type   osm_id
1      way 32108143
                                                              boundingbox
1 39.952309653061, 39.952409653061, -75.198505469388, -75.198405469388
             lat                 lon
1 39.9523596530612 -75.1984554693878
                                                                                                    display_nam
1 3718, Locust Walk, University City, Philadelphia, Philadelphia County, Pennsylvania, 19104, US
  class  type importance
1 place house      0.421
```

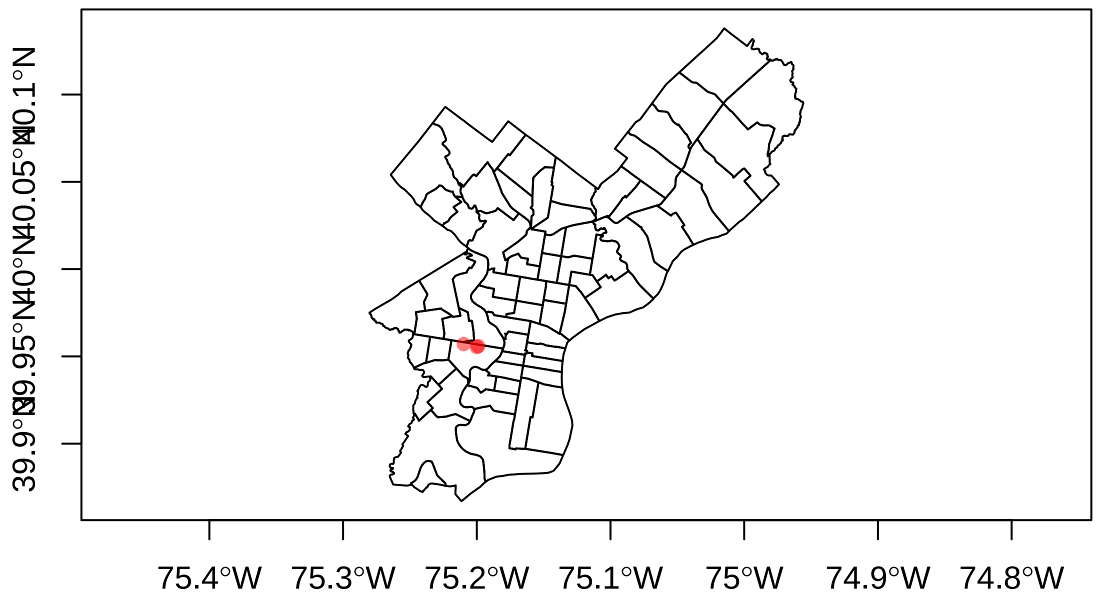
```
#  map them to the spatial coordinates
gcPenn$lon <- as.numeric(gcPenn$lon)
gcPenn$lat <- as.numeric(gcPenn$lat)
st_join(st_as_sf(gcPenn,
```

```
                 coords=c("lon","lat"),
                 crs=st_crs("+init=epsg:4326")),
        PPDmap)
```

```
Simple feature collection with 1 feature and 19 fields
geometry type:  POINT
dimension:      XY
bbox:           xmin: -75.19846 ymin: 39.95236 xmax: -75.19846 ymax: 39.95236
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
   place_id
1 228278154
                                                                  licence
1 Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright
  osm_type   osm_id
1      way 32108143
                                                            boundingbox
1 39.952309653061, 39.952409653061, -75.198505469388, -75.198405469388
                                                                                              display_nam
1 3718, Locust Walk, University City, Philadelphia, Philadelphia County, Pennsylvania, 19104, US
  class  type importance OBJECTID AREA PERIMETER PSACOV_ PSACOV_ID ID
1 place house      0.421       39 <NA>      <NA>    <NA>      <NA> 40
  DISTRICT__ PSA_NUM    OLD_SECTOR DESCRIPT                  geometry
1       <NA>     183 A,B,C,D,E,F,J     <NA> POINT (-75.19846 39.95236)
```

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

```
plot(st_geometry(PPDmap), axes=TRUE)
i <- which(PSAlookup$PSA_NUM=="183")
plot(st_geometry(PSAlookup[i,]), add=TRUE, col=rgb(1,0,0,0.5), pch=16)
```



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

```
# how many shootings in each PSA?
a <- table(PSAlookup$PSA_NUM)
a
```

```
011 012 021 022 023 031 032 033 051 052 053 061 062 063 071 072 073 077
  7   3  10   7   2   3   2   8   1   0   5   4   2   2   2   1   0   0
081 082 083 091 092 093 121 122 123 124 141 142 143 144 151 152 153 161
  1   2   1   3   8   0   4   6  10   4  14   8   2   1  18  10   5   3
162 171 172 173 181 182 183 191 192 193 221 222 223 224 241 242 243 251
  9   0  10   8   7  12   3   8  23   8  24  21   7  11   8  12   4   7
252 253 254 261 262 263 351 352 353 391 392 393
 10  10  15   9   5   6  11  13   7  10   9   8
```

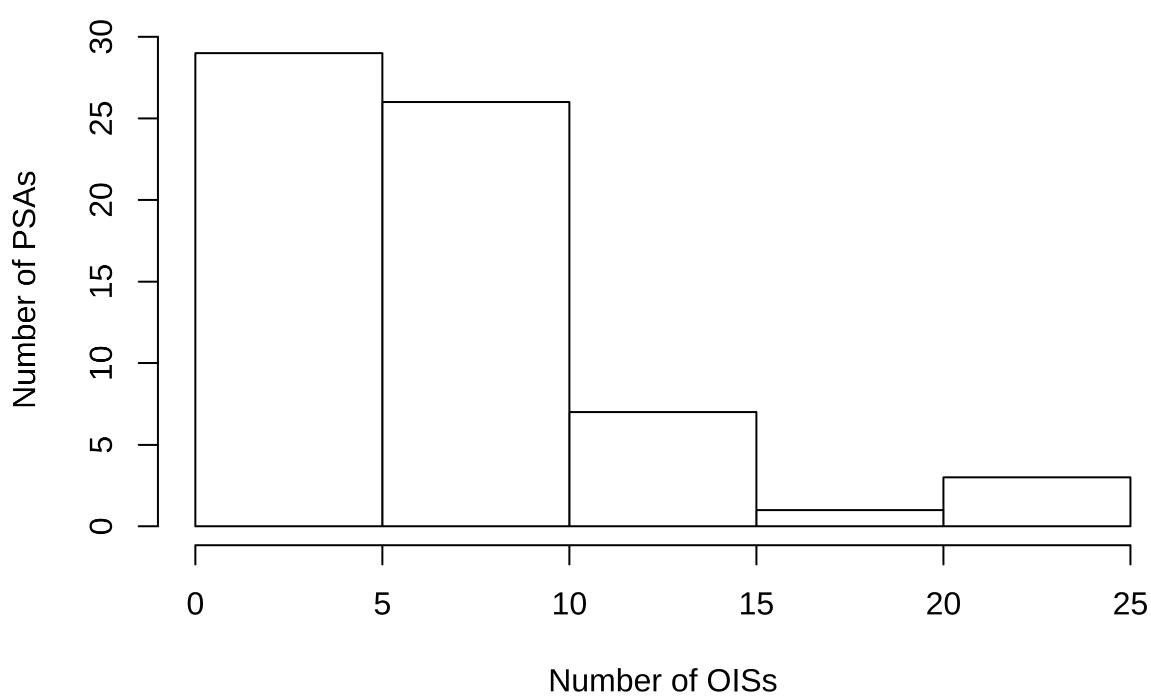
```
# merge the shooting count into the PPDmap data
i <- match(PPDmap$PSA_NUM, names(a))
PPDmap$nShoot <- a[i]
PPDmap[1:3,]
```

```
Simple feature collection with 3 features and 11 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: -75.24925 ymin: 39.87239 xmax: -75.13535 ymax: 39.93435
```

```
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
  OBJECTID AREA PERIMETER PSACOV_ PSACOV_ID ID DISTRICT__ PSA_NUM
1        1 <NA>      <NA>    <NA>      <NA>  1       <NA>     077
2        2 <NA>      <NA>    <NA>      <NA>  2       <NA>     011
3        3 <NA>      <NA>    <NA>      <NA>  3       <NA>     012
     OLD_SECTOR DESCRIPT                          geometry nShoot
1             A     <NA> POLYGON ((-75.2338 39.88977...      0
2 A,B,C,D,E,F,J     <NA> POLYGON ((-75.19724 39.9294...      7
3 G,H,I,K,L,M,N     <NA> POLYGON ((-75.17305 39.9105...      3
```

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(a, xlab="Number of OISs", ylab="Number of PSAs", main="")
```



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

```
a <- cut(PPDmap$nShoot,
         breaks=c(0,1,5,10,15,20,25,30),
         right=FALSE)
a
```

```
 [1] [0,1)   [5,10)  [1,5)   [10,15) [5,10)  [1,5)   [1,5)   [0,1)
 [9] [5,10)  [1,5)   [1,5)   [1,5)   [1,5)   [1,5)   [0,1)   [1,5)
```

```
[17] [1,5)   [1,5)   [1,5)   [5,10)  [0,1)   [1,5)   [5,10)  [10,15)
[25] [1,5)   [10,15) [5,10)  [1,5)   [1,5)   [10,15) [5,10)  [1,5)
[33] [5,10)  [0,1)   [10,15) [5,10)  [5,10)  [10,15) [1,5)   [20,25)
[41] [20,25) [5,10)  [10,15) [5,10)  [10,15) [1,5)   [5,10)  [5,10)
[49] [5,10)  [10,15) [5,10)  [10,15) [5,10)  [5,10)  [5,10)  [20,25)
[57] [5,10)  [1,5)   [5,10)  [1,5)   [15,20) [10,15) [15,20) [10,15)
[65] [10,15) [5,10)
Levels: [0,1) [1,5) [5,10) [10,15) [15,20) [20,25) [25,30)
```

`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.

```
col <- rev(heat.colors(7,1))
col
```

```
[1] "#FFFF80FF" "#FFFF00FF" "#FFCC00FF" "#FF9900FF" "#FF6600FF" "#FF3300FF"
[7] "#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".

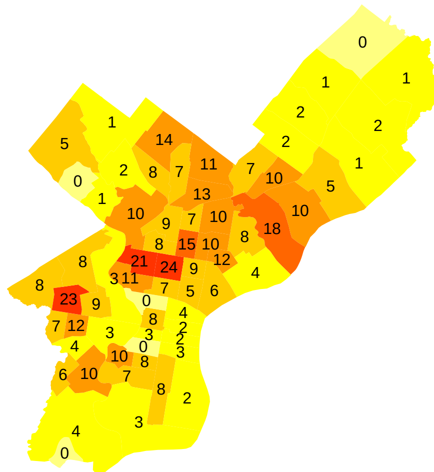
Now we need to select the right color for each PSA. If I apply `as.numeric()` to `a`, then all the [0,1) will convert to 1, the [1,5) will convert to 2, and so on up to [25,30) converting to 7. So `col[as.numeric(a)]` will pick out the right color for each PSA. Now create the map coloring each PSA using the right color.

```
plot(st_geometry(PPDmap), col=col[as.numeric(a)], border=NA)
# add the number of shootings to the map
a <- st_coordinates(st_centroid(PPDmap))
```

```
Warning in st_centroid.sf(PPDmap): st_centroid assumes attributes are
constant over geometries of x
```

```
Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
of_largest_polygon): st_centroid does not give correct centroids for
longitude/latitude data
```

```
text(a[,1], a[,2], PPDmap$nShoot, cex=0.5)
```



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.

Summary

We started with just a web page linking to a collection of pdf files. We used regular expressions to extract everything we could from the web page tables. We had R "read" the pdf files to extract the dates that were not readily available. We geocoded the stops so that we could put them on a map. Finally, we could tabulate by PSA the number of OISs and map those as well.

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

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.