Intro to data management in R

Dan Weinberger

2/20/2020

Why use R for data management?

- R has a steep learning curve, but…
- Free software
- Can do anything from basic analysis and data management to advanced statistical and mathematical modeling
- Beautiful graphics
- Can make reproducible analysis, unlike spreadsheet software

Step 1: Create an R project

- Organizing your code and data from the beginning of an analysis is critical.
- An R project is a way to organize all of your files in 1 place.

1. On your computer create a folder called “Project 1”. Within this folder create sub-folders called “Data” and “R Code”
2. In RStudio, create a new project by going to File/New project/Existing Directory. Then select your “Project 1” folder
3. “Project 1” is now your working directory. Whenever you want to repoen the project, go to the folder Project 1 and click on the file that ends in .RProj.

- **Protip**: Use a version control program like Git/Github with RStudio to track changes. When you create your project, select “Version control”

Step 2: Load your data into R

- Save your dataset in the Data folder in your project Usually you will have a spreadsheet created in another program. It might be a text file (.csv, .txt) or a MS Excel file (.xlsx, .xls). Or you might even have SPSS or SAS files created in other stats softwares. R can import any of these.

To import a csv file and save it as 'ds1', use this code, substituting the name of your file for "datasetname"

```
ds1 <- read.csv('./Data/datasetname.csv')
```

```
#install.packages('readxl') #run one time only
library(readxl) #run each time you re-open R
ds1 <- read_excel('./Data/datasetname.xlsx')
```

or an R data file (.rds)

```
ds1<-readRDS('./Data/cincinnati.rds')
```

There are also functions to import SAS files (read.sas7bdatin sas7bdat package) and SPSS files (read.sppp in foreign package)

Issues with importing data

Often datasets will have information on the top rows that you don't want to import. For instance, the owner of the data might include information about the source in the top few rows We do not want to import this into R. Let's say there are 5 rows with this type of information that we want to skip, and the row with the column names is column 6. We modify our import statement by using 'skip='

```
ds1 <- read.csv('./Data/datasetname.csv', skip=5)
```

Or import an Excel spreadsheet.

```
library(readxl) #run each time you re-open R
ds1 <- read_excel('./Data/datasetname.xlsx', skip=5)
```

Step 3: check the variable types

- R has 3 basic types of data it stores: numeric, character, and factor. Anything that represents a number should be stored as numeric (e.g., number of cases of disease). Labels made up of letters are typically stored as a character. Factors can be used for discrete categories (e.g., age groups, state names). Often when you import a dataset, by default it will store character variables as factors (and sometime will be confused by numbers as well). Use the str() function to check the structure of the data .

Variable type

This shows we have latitude and longitude stored as numeric variables (num), Time of incident is a date-time(month/day/Year HH:MM) but is stored as a factor variabl. The Cause of the visit is stored as a factor variable. And the neighborhood is stored as a factor variable

```
str(ds1)
```

```
## 'data.frame':    355645 obs. of  6 variables:
##  $ LATITUDE_X                   : num  39.2 39.1 39.2 39.1 39.2 ...
##  $ LONGITUDE_X                  : num  -84.4 -84.4 -84.4 -84.5 -84.4 ...
##  $ CREATE_TIME_INCIDENT         : Factor w/ 324106 levels "1/1/2015 0:02",..: 50921 290002 312127 21
##  $ ARRIVAL_TIME_PRIMARY_UNIT    : Factor w/ 280190 levels "","1/1/2015 0:05",..: 44330 250437 269764
##  $ CFD_INCIDENT_TYPE_GROUP      : Factor w/ 105 levels "","ABDOMINAL PAIN / PROBLEMS",..: 96 96 96 9
##  $ COMMUNITY_COUNCIL_NEIGHBORHOOD: Factor w/ 72 levels "AVONDALE","AVONDALE - NORTH AVONDALE",..: 54
```

Step 4: Convert date variable to a date-time format

Sample formats for dates and datetimes:

- 2010-11-01 “%Y-%m-%d”
- 11/01/2010 “%m/%d/%Y”
- 11/01/10 “%m/%d/%y”
- Nov-01-2010 %b-%d-%Y"
- Nov-01-2010 13:02 %b-%d-%Y %H:%M"

```
ds1$datetime<-as.POSIXlt(ds1$CREATE_TIME_INCIDENT, #variable with the datetime
                         format=c("%m/%d/%Y %H:%M"))
```

Extract the date and hour to a new variable

Whenevr we are dealing with dates, lubridate package has a lot of useful functions

```
ds1$date<-date(ds1$datetime)
```

Extract the hour of the event (0-23) to a new variables

```
ds1$hour<-hour(ds1$datetime)
```

View the formatted dataset

Open the spreadhseet using View(ds1)

```
View(ds1)
```

Or just look at the variables you are interested in, showing first 5 rows

```
ds1[1:5, c('CFD_INCIDENT_TYPE_GROUP','date','hour')]
```

```
##              CFD_INCIDENT_TYPE_GROUP       date hour
## 1 TRAFFIC / TRANSPORTATION INCIDENTS 2018-10-30    8
## 2 TRAFFIC / TRANSPORTATION INCIDENTS 2017-08-03   16
## 3 TRAFFIC / TRANSPORTATION INCIDENTS 2017-09-24   14
## 4 TRAFFIC / TRANSPORTATION INCIDENTS 2017-05-05   14
## 5 TRAFFIC / TRANSPORTATION INCIDENTS 2017-09-10   19
```

Step 5: Explore your data to look for problems

What is the range of values for date and hour? Are there any that don't make sense (e.g. and hour=27?)

```
range(ds1$date)
```

```
## [1] "2015-01-01" "2019-04-08"
```

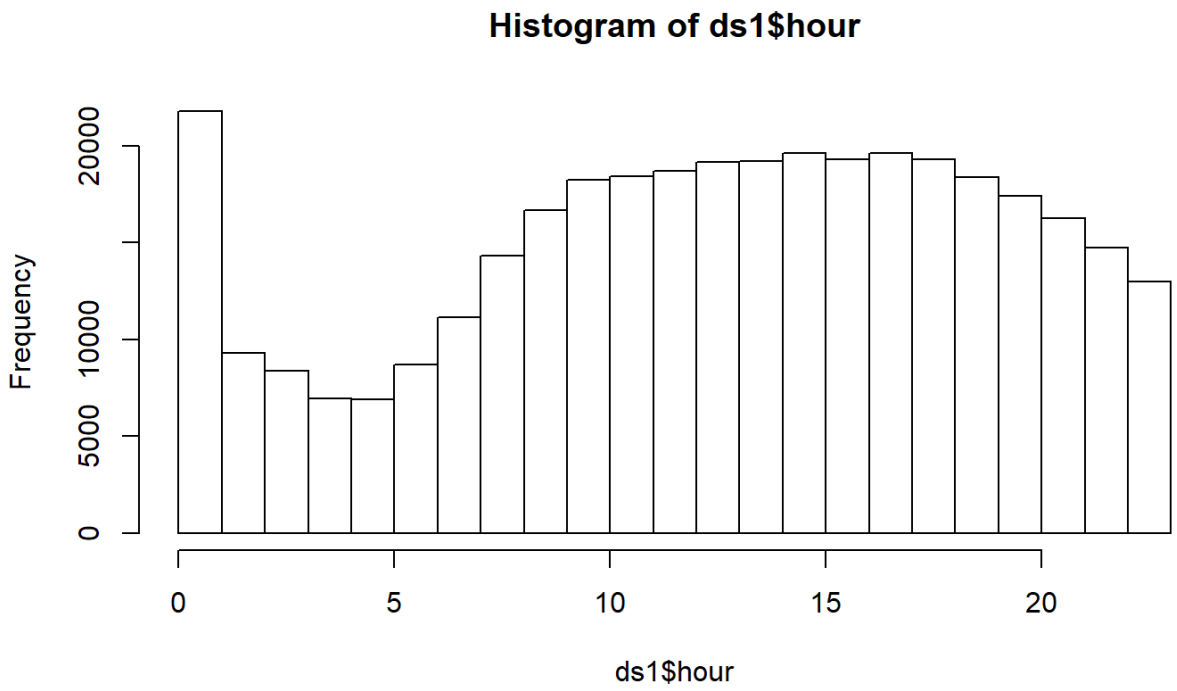
```
range(ds1$hour)
```

```
## [1]  0 23
```

View histograms of key variables

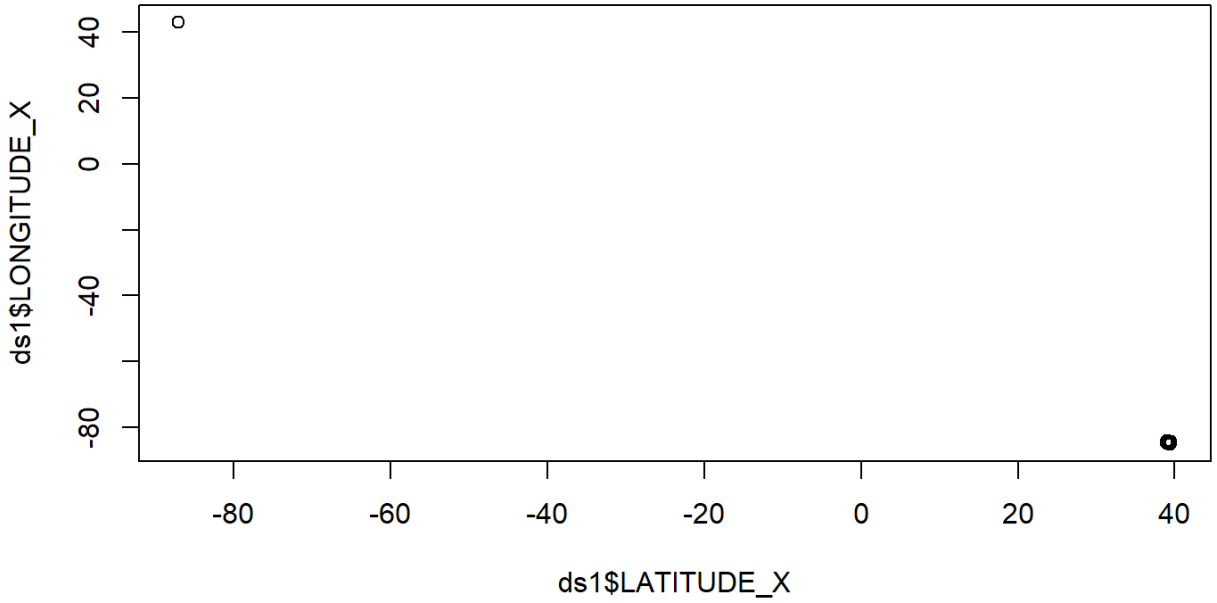
Histogram showing the hour of calls

```
hist(ds1$hour)
```



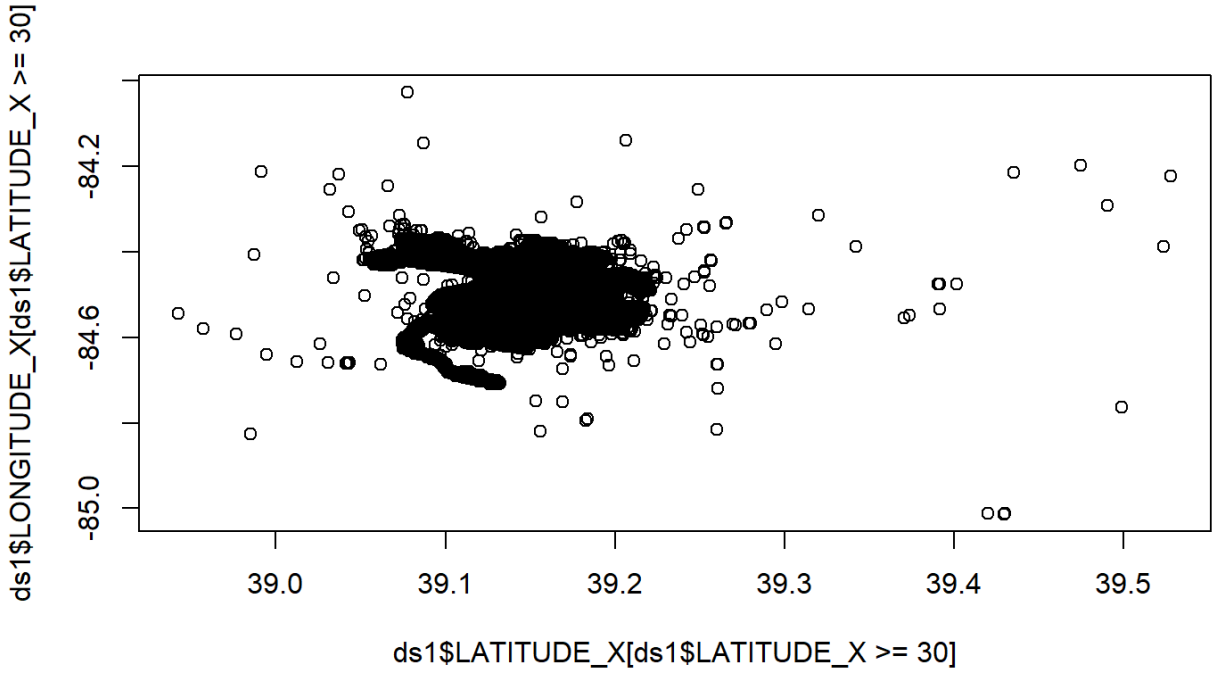
Simple map of coordinates

```
plot(ds1$LATITUDE_X, ds1$LONGITUDE_X)
```



Map again, but exclude outliers

```
plot(ds1$LATITUDE_X[ds1$LATITUDE_X>=30], ds1$LONGITUDE_X[ds1$LATITUDE_X>=30])
```



Step 6: Aggregate your data by day and cause

Create a variable with a value of 1 for every observation. We will then sum this up by day and cause

```
ds1$one<-1
```

sum the 'one' variable by date and cause

```
ds2<-aggregate(ds1$one, by=list('date'=ds1$date, 'cause'=ds1$CFD_INCIDENT_TYPE_GROUP), FUN=sum)
names(ds2)<-c('date', 'cause','count') #rename the columns
```

How do the data look?

Use the 'head' function to get a preview of the dataset

```
head(ds2)
```

```
##            date                    cause count
## 1001 2015-09-26 ABDOMINAL PAIN / PROBLEMS     7
## 1002 2015-09-27 ABDOMINAL PAIN / PROBLEMS     6
## 1003 2015-09-28 ABDOMINAL PAIN / PROBLEMS     6
## 1004 2015-09-29 ABDOMINAL PAIN / PROBLEMS     9
## 1005 2015-09-30 ABDOMINAL PAIN / PROBLEMS     2
## 1006 2015-10-01 ABDOMINAL PAIN / PROBLEMS     3
```

Step 7: Reshape your data from long to wide

The data are currently in th 'long' format, where we have a row for each date and cause. We might want to have it in the wide format, where we have a row for each date and a column for each cause. We can do this with the reshape2 package

```
library(reshape2)
```

First "melt" your data into a format that can be reshapred

```
ds2.m<-melt(ds2[,c('date','cause', 'count')], #just keep the subset of variables we want to analyze
            id.vars= c('date', 'cause')) #specify the id variables we want to use
```

Then "cast" your data into a new format

the rows come before tilda, columns after tilda date~cause gives each date a row and each cause a column

```
ds2.c<-dcast(ds2.m, date~cause)
```

```
head(ds2.c)
```

```
##         date Var.2 ABDOMINAL PAIN / PROBLEMS ACCIDENT WITH INJURY
## 1 2015-01-01    NA                         3                    NA
## 2 2015-01-02    NA                         4                    NA
## 3 2015-01-03    NA                         5                    NA
## 4 2015-01-04    NA                         7                    NA
## 5 2015-01-05    NA                         8                    NA
## 6 2015-01-06    NA                         4                    NA
##   ACCIDENT WITH INJURY - FIRE ONLY AIR CRAFT IN TROUBLE OR DOWN
## 1                                6                           NA
## 2                                8                           NA
## 3                                9                           NA
## 4                               10                           NA
## 5                                7                           NA
## 6                                6                           NA
##   ALLERGIES (REASTIONS) / ENVENOMATIONS (STINGS; BITES)
```

Replacing missing values

You can see there are a lot of NA (missing) values. This is because not every cause is documented on ever day. We could replace all of the NAs with 0. We can use the apply statement to do this. We will ignore column 1 ([,-1]) and then wherever a column has NA, it is replace with 0. Apply goes column by column and tests whether for missing values

```
ds2.c[,-1]<- apply(ds2.c[,-1],
                   2, #search by column (change to 1 to search by row)
                   function(x){  #apply this function to all columns
                     x[is.na(x)]<-0 #if the column enry has an NA, it is replaced with 0
                   return(x)
                   }
                   )
```

Data with NA replaced by 0

```
head(ds2.c)
```

```
##         date Var.2 ABDOMINAL PAIN / PROBLEMS ACCIDENT WITH INJURY
## 1 2015-01-01     0                         3                     0
## 2 2015-01-02     0                         4                     0
## 3 2015-01-03     0                         5                     0
## 4 2015-01-04     0                         7                     0
## 5 2015-01-05     0                         8                     0
## 6 2015-01-06     0                         4                     0
##   ACCIDENT WITH INJURY - FIRE ONLY AIR CRAFT IN TROUBLE OR DOWN
## 1                                6                            0
## 2                                8                            0
## 3                                9                            0
## 4                               10                            0
## 5                                7                            0
## 6                                6                            0
##   ALLERGIES (REASTIONS) / ENVENOMATIONS (STINGS; BITES)
## 1                                                     0
## 2                                                     0
## 3                                                     1
## 4                                                     1
## 5                                                     1
## 6                                                     1
```

Reshape your data from wide to long

We could also reshape the data back to a long format

```
ds2.c.m<-melt(ds2.c, id.vars = 'date')
ds2.c.m.c<- dcast(ds2.c.m, date+variable~.)
names(ds2.c.m.c)<-c('date','cause','count')
```

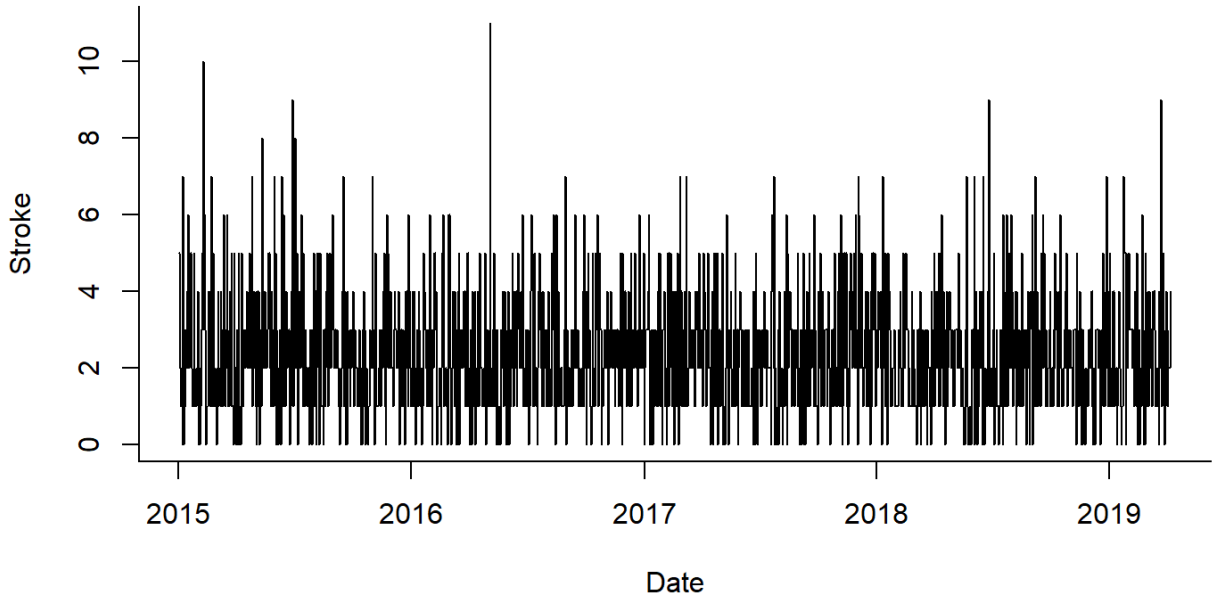
```
head(ds2.c.m.c)
```

```
##         date                                                cause count
## 1 2015-01-01                                                Var.2     0
## 2 2015-01-01                          ABDOMINAL PAIN / PROBLEMS     3
## 3 2015-01-01                               ACCIDENT WITH INJURY     0
## 4 2015-01-01                     ACCIDENT WITH INJURY - FIRE ONLY     6
## 5 2015-01-01                         AIR CRAFT IN TROUBLE OR DOWN     0
## 6 2015-01-01 ALLERGIES (REASTIONS) / ENVENOMATIONS (STINGS; BITES)     0
```

Plot a daily time series

We will look at the aggregated stroke cases to see if it worked OK. Looks good!

```
plot(ds2.c$date,ds2.c$`STROKE (CVA) / TRANSIENT ISCHEMIC ATTACK (TIA)`, type='l',
     ylab='Stroke', xlab='Date',
     bty='l')
```



Reshape directly from line list data to daily time series

Subset to just keep columns you want. Keep all rows. this is indicated by the brackets [rows, columns]. Since we want all rows, we leave the area before the comma blank, and after the comma, we provide a vector of the column names

```
ds1.subset<-ds1[,c("date", "hour","CFD_INCIDENT_TYPE_GROUP"  )]
```

'melt' the dataset ds1.subset. The 'molten' dataframe m1 will be used in the next step

```
m1<- melt(ds1.subset, id.vars=c("CFD_INCIDENT_TYPE_GROUP", 'date', 'hour'))
```

Now 'cast' the molten dataframe m1 to the wide format. Each row is a date, each column is an incident type

```
c1 <- dcast(m1 ,
            date~CFD_INCIDENT_TYPE_GROUP,  # each row is a date, each column is an incident type
            fun.aggregate = length) #counts how many observations we are aggregating.
```

```
## Using hour as value column: use value.var to override.
```

View the cast dataset

```
head(c1)
```

```
##         date Var.2 ABDOMINAL PAIN / PROBLEMS ACCIDENT WITH INJURY
## 1 2015-01-01     0                         3                     0
## 2 2015-01-02     0                         4                     0
## 3 2015-01-03     0                         5                     0
## 4 2015-01-04     0                         7                     0
## 5 2015-01-05     0                         8                     0
## 6 2015-01-06     0                         4                     0
##   ACCIDENT WITH INJURY - FIRE ONLY AIR CRAFT IN TROUBLE OR DOWN
## 1                                 6                           0
## 2                                 8                           0
## 3                                 9                           0
## 4                                10                           0
## 5                                 7                           0
## 6                                 6                           0
##   ALLERGIES (REASTIONS) / ENVENOMATIONS (STINGS; BITES)
## 1                                                     0
## 2                                                     0
## 3                                                     1
## 4                                                     1
## 5                                                     1
## 6                                                     1
```

We could also cast to the long format

Each row is a date/incident type combination

```
c2 <- dcast(m1 ,
            date+CFD_INCIDENT_TYPE_GROUP ~.,  # each row is a date, each column is an incident type
            fun.aggregate = length) #counts how many observations we are aggregating.
```

```
## Using hour as value column: use value.var to override.
```

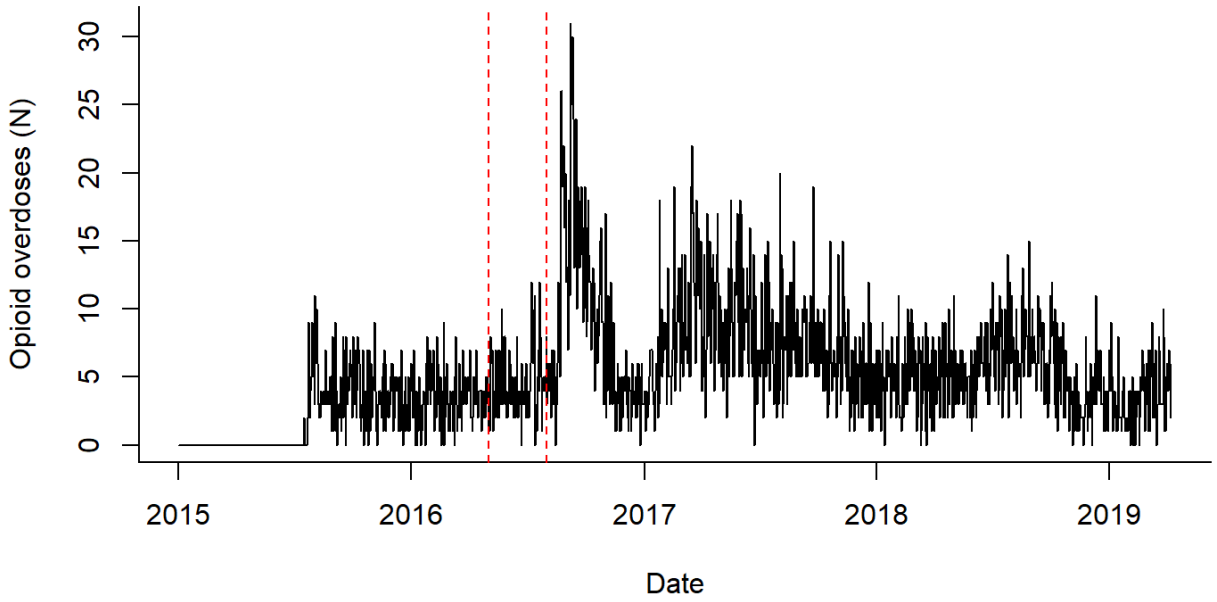
```
##         date                          CFD_INCIDENT_TYPE_GROUP  .
## 1 2015-01-01                       ABDOMINAL PAIN / PROBLEMS  3
## 2 2015-01-01                 ACCIDENT WITH INJURY - FIRE ONLY  6
## 3 2015-01-01                              ASSAULT WITH INJURY 16
## 4 2015-01-01                 AUTO ACCIDENT - CAR HIT BUILDING  2
## 5 2015-01-01                             AUTOMATIC FIRE ALARM 11
## 6 2015-01-01 BACK PAIN (NON-TRAUMATIC OR NON-RECENT TRAUMA)  1
```

Let's make a time series of the opioid overdoses

Daily-aggregated time series

```
plot(c1$date, c1$`HEROIN OVERDOSE`,
     bty='l', #turn off top and right plot border
     type='l', #line plot
     ylab='Opioid overdoses (N)',
     xlab='Date',
     )
     abline(v=as.Date(c('2016-05-01', '2016-08-01')),lty=2, col='red') #Period we examined in SATSCAN
```

Time series


Opioid overdoses (N)
0
5
10
15
20
25
30
2015
2016
2017
2018
2019
Date

Mapping in R

R has tons of mapping functions available.

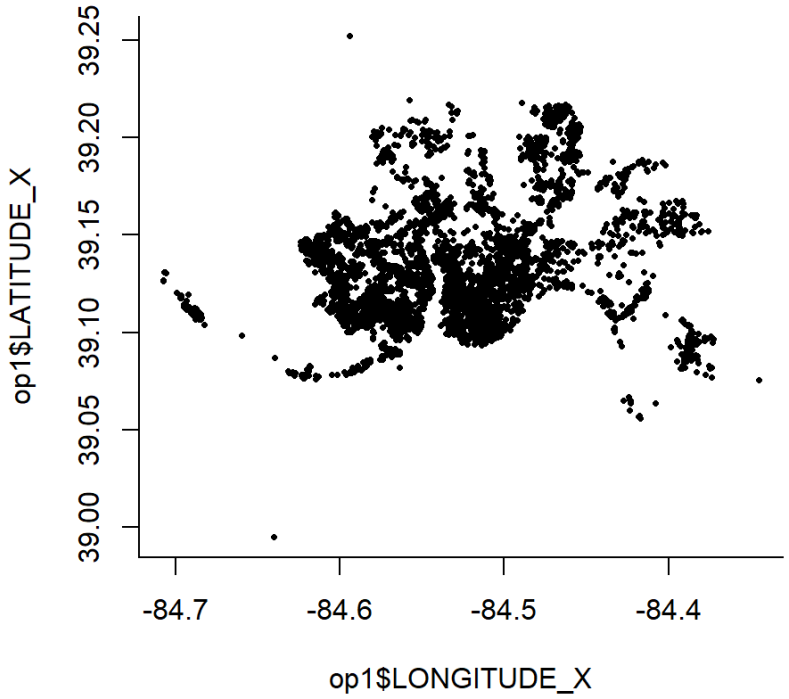
First, let's just make a map of all the locations. Subset the line list data to just breathing problems

```
op1<-ds1[ds1$CFD_INCIDENT_TYPE_GROUP=='HEROIN OVERDOSE',] #pull out rows where incident type is heroin o
```

Since we have each location, we can just make a dot at each latitute/longitude

```
plot(op1$LONGITUDE_X, #x variable
     op1$LATITUDE_X , #Y variable
     pch=16,  #shape of the markers
     cex=0.5, #size of the marker
     col='black',  #specify a black marker
     bty='l')
```

Simple map


op1$LATITUDE_X
39.25
39.20
39.15
39.10
39.05
39.00
-84.7
-84.6
-84.5
-84.4
op1$LONGITUDE_X

Play with color transparency

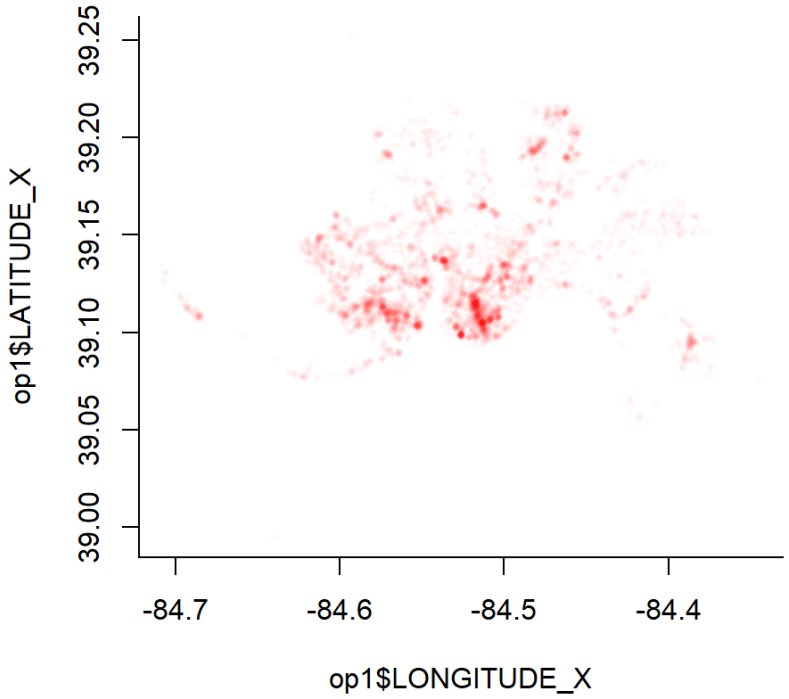
The rgb function crates custom color combinations. There are 4 entries, each range from 0 to 1: red, green, blue, alpha. Alpha controls the opacity of the color. rgb(1,0,0,1) would give you a totally opaque red, rgb(0,1,0,1) would give opaque green, and rgb(0,0,0,1) would give opaque blue. rgb(0,0,1,0.5) would give a partially transparent blue. rgb(0,0,0,0.25) would give a transparent black color) Let's make transparent red markers

```
red.t=rgb(1,0,0,alpha=0.02) #red, with transparency
```

Much better–now we can see hotspots

```
plot(op1$LONGITUDE_X, #x variable
     op1$LATITUDE_X , #Y variable
     pch=16,  #shape of the markers
     cex=0.5, #size of the marker
     col=red.t,  #specify a black marker
     bty='l')
```

Transparent dots



Adding boundaries

To add boundaries to the map, you need a boundary ("shape file"). We have the neighborhood boundaries, which were downloaded from the Zillow website
Read in a file that has map coordinates for neighborhood boundaries

```
plot(shp.cin, bty='l') #plot the boundaries
points(op1$LONGITUDE_X, #x variable
     op1$LATITUDE_X , #Y variable
     pch=16,  #shape of the markers
     cex=0.5, #size of the marker
     col=red.t  #specify a transparent red
    )
```

Boundary map