Massachusetts Institute of Technology
Department of Urban Studies and Planning

11.188: Urban Planning and Social Science Laboratory
11.520: Workshop on GIS (2nd half-semester)

Lab Exercise 6 (Part 2)
Exploring Google Maps mobility data

April 12, 2021

Aims of this lab

- Leverage data tracked by Google to understand mobility patterns
- Merge datasets to gain insights into urban phenomena
- Use R for spatial analysis

As the COVID-19 pandemic raged over the last year, many restaurants have suffered from lack of footfall while customers find their food options limited. An added complication is reduced transit options for individuals, so many now either rely on delivery services or can only walk for takeout. As a researcher/urban planner, how can you understand what kind of access people have to restaurants in different areas?

Through this lab you will utilize data from Yelp and Google to understand the food choices of one such individual under lockdown, who was social distancing in Central Square in March, 2020.

Setting up your machine

We will be using the "loc2020.csv" file, which is a CSV file of location tracks that we have provided, for this lab. This file is present in the data package for this lab on Stellar. Make sure to copy it to the same folder R is using for this session. You can find out which folder is being used as the working directory by using command:

```
#Find where R is storing this file and any output you produce later on
getwd()
```

If you would like to change your working directory you can do so by adding the file path of your folder below. Remember R only uses forward slash **'/' not "\"** in pathnames e.g. my pathname was **"C:/Users/rounaq/lab6"**

```
#set working directory by pasting pathname with forward slash in “”
setwd('….')
```

Now we can install and load packages and libraries we will need for this session. You only have to install packages once but you have to load them into your session using the library command every session.

```r
#install packages
install.packages("googleway")
install.packages("jsonlite")
install.packages("leaflet")
install.packages("leaflet.extras")
install.packages("viridis")
install.packages("stats")
install.packages("dplyr")
install.packages("plyr")
#load libraries
library(googleway)
library(jsonlite)
library(leaflet)
library(leaflet.extras)
library(viridis)
library(stats)
library(dplyr)
library(plyr)
```

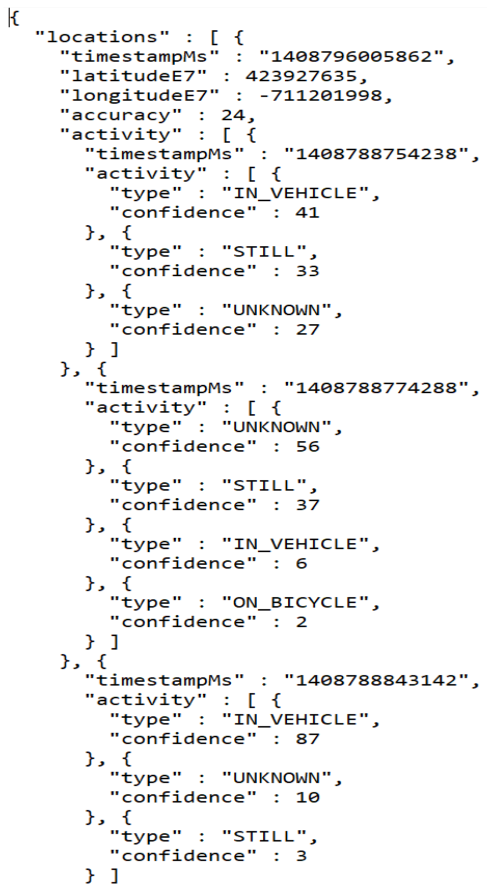
Understanding Google Tracking

While we have provided you with a csv file of tracks you will be using for this lab, it is important to understand how this file was generated. You do not need to attempt this section, but should read through it.

If you have turned on location history on your cell phone and you use Google Maps, then Google has been tracking/storing information on your movement patterns. You can see this history on Google's own interface by going to https://www.google.com/maps/timeline?pb. Zoom in to the various places you have visited and see how specific the movement patterns which Google saves on you. While this is a handy way to view the history, it does not let you analyse your own movement patterns.

We can access all the data google has stored associated with your account by going to https://takeout.google.com/. We are not asking you to download your data, but instead have provided you with a csv file which already has a segment of one individual's tracks. We have partially cleaned the data for you but will detail now how we did so.

When you download your Google location history data you get a zipfolder with multiple files in it. You use the one called Location History.json which is in a JavaScript Object Notation (JSON) format. It is basically a large, tagged list, with all the relevant information stored and looks like this:

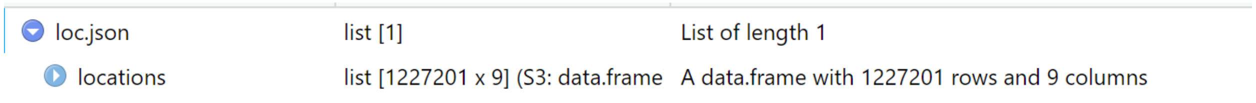


```
{
  "locations" : [ {
    "timestampMs" : "1408796005862",
    "latitudeE7" : 423927635,
    "longitudeE7" : -711201998,
    "accuracy" : 24,
    "activity" : [ {
      "timestampMs" : "1408788754238",
      "activity" : [ {
        "type" : "IN_VEHICLE",
        "confidence" : 41
      }, {
        "type" : "STILL",
        "confidence" : 33
      }, {
        "type" : "UNKNOWN",
        "confidence" : 27
      } ]
    }, {
      "timestampMs" : "1408788774288",
      "activity" : [ {
        "type" : "UNKNOWN",
        "confidence" : 56
      }, {
        "type" : "STILL",
        "confidence" : 37
      }, {
        "type" : "IN_VEHICLE",
        "confidence" : 6
      }, {
        "type" : "ON_BICYCLE",
        "confidence" : 2
      } ]
    }, {
      "timestampMs" : "1408788843142",
      "activity" : [ {
        "type" : "IN_VEHICLE",
        "confidence" : 87
      }, {
        "type" : "UNKNOWN",
        "confidence" : 10
      }, {
        "type" : "STILL",
        "confidence" : 3
      } ]
```

To inspect this file further we imported it into R using a library called JSONlite ***[remember you are not expected to do this part, but it is important to read through to understand how this file was made].***

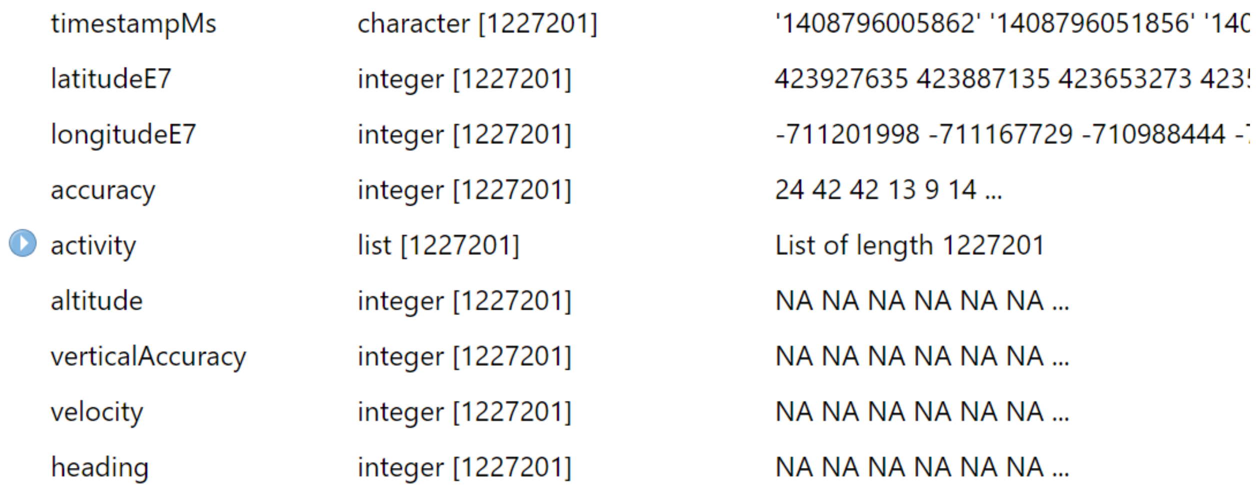
```
#get json file. remember $locations only exist in full history not monthly hi
story
loc.json= fromJSON(txt = "Location History.json")
```

When inspected, this object has the following data structure:



loc.json	list [1]	List of length 1
locations	list [1227201 x 9] (S3: data.frame	A data.frame with 1227201 rows and 9 columns

It is basically a list of 1227201 data points with 9 columns. Exploring it further the columns are:



timestampMs	character [1227201]	'1408796005862' '1408796051856' '140
latitudeE7	integer [1227201]	423927635 423887135 423653273 423
longitudeE7	integer [1227201]	-711201998 -711167729 -710988444 -
accuracy	integer [1227201]	24 42 42 13 9 14 ...
activity	list [1227201]	List of length 1227201
altitude	integer [1227201]	NA NA NA NA NA NA ...
verticalAccuracy	integer [1227201]	NA NA NA NA NA NA ...
velocity	integer [1227201]	NA NA NA NA NA NA ...
heading	integer [1227201]	NA NA NA NA NA NA ...

Google provides a little bit of metadata on the columns which we include below:

The JSON Location History file describes device location signals and associated metadata collected while you were opted into Location History which you have not subsequently deleted.

- **timestampMs(int64):** Timestamp (UTC) in milliseconds for the recorded location.
- **latitudeE7(int32):** The latitude value of the location in E7 format (degrees multiplied by 10**7 and rounded to the nearest integer).
- **longitudeE7(int32):** The longitude value of the location in E7 format (degrees multiplied by 10**7 and rounded to the nearest integer).
- **accuracy(int32):** Approximate location accuracy radius in meters.
- **velocity(int32):** Speed in meters per second.
- **heading(int32):** Degrees east of true north.
- **altitude(int32):** Meters above the WGS84 reference ellipsoid.
- **verticalAccuracy(int32):** Vertical accuracy calculated in meters.
- **activity:** Information about the activity at the location.
- **timestampMs(int64):** Timestamp (UTC) in milliseconds for when the datapoint was recorded
- **type:** Description of the activity type.
- **confidence(int32):** Confidence associated with the specified activity type.

We then cleaned up this file *[see appendix for instructions]* and created the csv file which you can now read into R.

Mapping Google Tracks

In your R studio session, read

- the loc2020 CSV file we have provided
- the csv file of restaurants in a 1000m radius of Central Square that you saved in the last part of the lab.

MAKE SURE BOTH FILES ARE IN THE SAME WORKING DIRECTORY THAT R IS USING IN THIS SESSION

```
loc2020<-read.csv('loc2020.csv')


yelp<-read.csv('insert name of 1000m csv.csv')
```

loc2020 is a csv which has all GPS locations recorded by Google in February and March 2020, as well as additional variables created by us [see appendix for code of how these variables were created].

We want to be able to distinguish the quarantine mobility from pre-quarantine. So we can extract data from **March 12, 2020** onwards from the larger dataset. Below we show you how to extract data for February 2020, and you can edit the code to create a dataset for March 12, 2020 onwards.

Notice the syntax, we are asking R to create a new dataset called locfeb2020 which only has the rows where month==2.

How would you change this to create a condition which only has data where month is March AND where the date is greater than 11? The symbol for AND in R is ***&*** *which would go in before the parenthesis*

Make sure you open the dataset to inspect the variable names you need

```
#create further subsets with only Feb2020 data


locfeb2020<-loc2020[which(loc2020$month==2), ]
```

We can now use an interactive leaflet map to visualize the mobility patterns.

Leaflet is an opensource Javascript library for mapping which allows the creation of powerful, interactive, highly customizable maps with very little code. R has a certain library called leaflet which we've installed above, that allows us to talk to the Leaflet library, initialize a leaflet map widget and display our own data on the map widget. You can read more about Leaflet for R here: https://rstudio.github.io/leaflet/

We will demonstrate how to make leaflet maps using the February 2020 data but you should edit this code and make your own maps for the dataset representing dates starting March 12,2020.

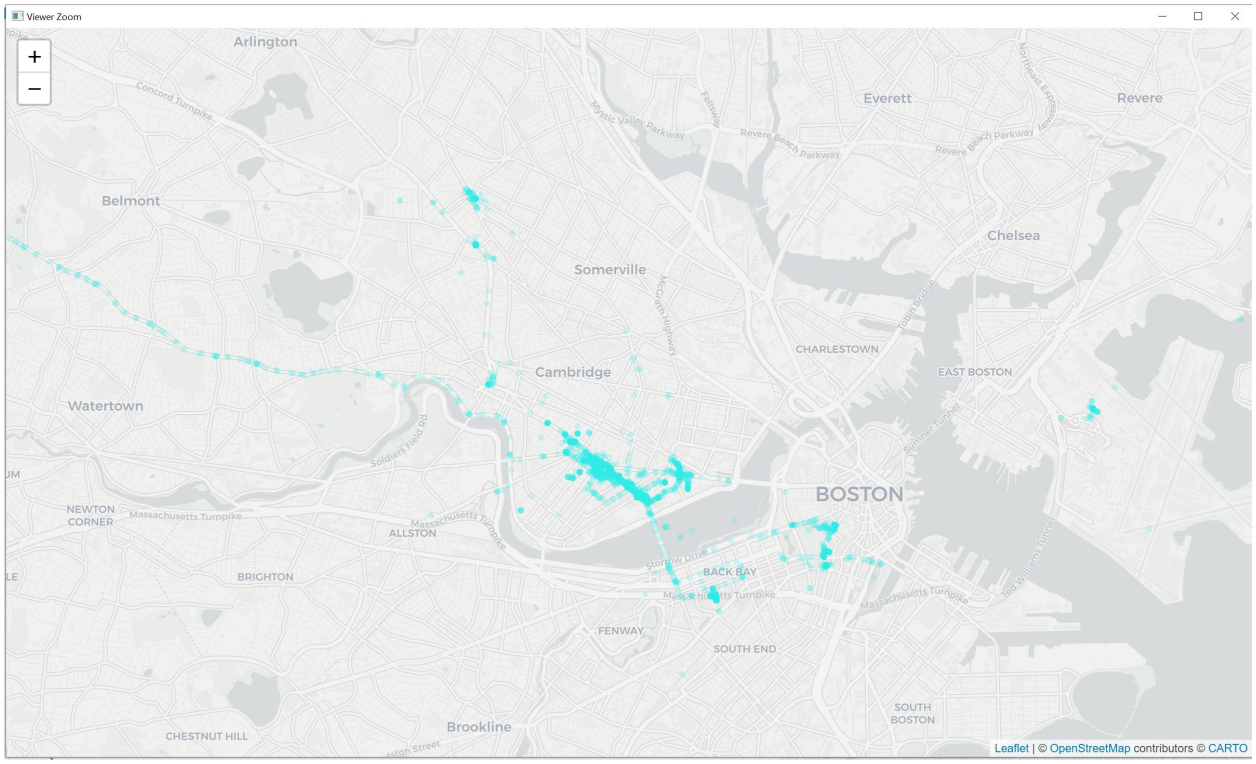
First we set some parameters like colors and radius of our circle markers which will represent our mobility tracks. Then we will call the leaflet library and layer on certain options.

- *Leaflet() initializes a leaflet map widget which stores the data and options which we will define*
- *addProviderTiles* sets a basemap which can be changed to many options (some of which we'll show in the next sections)
- *fitbounds* sets the bounding box or 'frame' of the map by adding the latitude and longitude of its 'extent'.
- *addCircleMarkers* is like symbology in ArcMap. We tell leaflet what data we want to symbolize by parsing in the latitude and longitude columns in it and also define the aesthetics
- The *%>%* symbols you see are called 'pipes' and you can think of these as pipes that connect one layer of the code to the other. The greater than symbol '>' indicates that the result to the left is sent to the next operation on the right, even if it is shown on the next line.
- So the next section defines four variables and then executes one multi-line command to build our map in 'leafmap.feb'. Then the command leafmap.feb display the map that you constructed.

```
#set parameters like color radius of circles
r.goog<-3
col.goog<-"#34ebe5"
r.yelp<-2
col.yelp<-'#f54299'

#making leaflet map for February 2020
leafmap.feb<- leaflet(locfeb2020) %>%
  addProviderTiles(providers$CartoDB.Positron,
                   options = providerTileOptions(opacity = 0.5)) %>%

  fitBounds(  ~min(-71.144427), ~min(42.346422), ~max(-71.048083), ~max(42.39
8743))%>%

  addCircleMarkers( #adding google tracks
    stroke = FALSE, fillOpacity = .3,
    radius= r.goog,
    color= col.goog,
    lng = ~locfeb2020$lonGPS, lat = ~locfeb2020$latGPS
  )
```

```
leafmap.feb
```

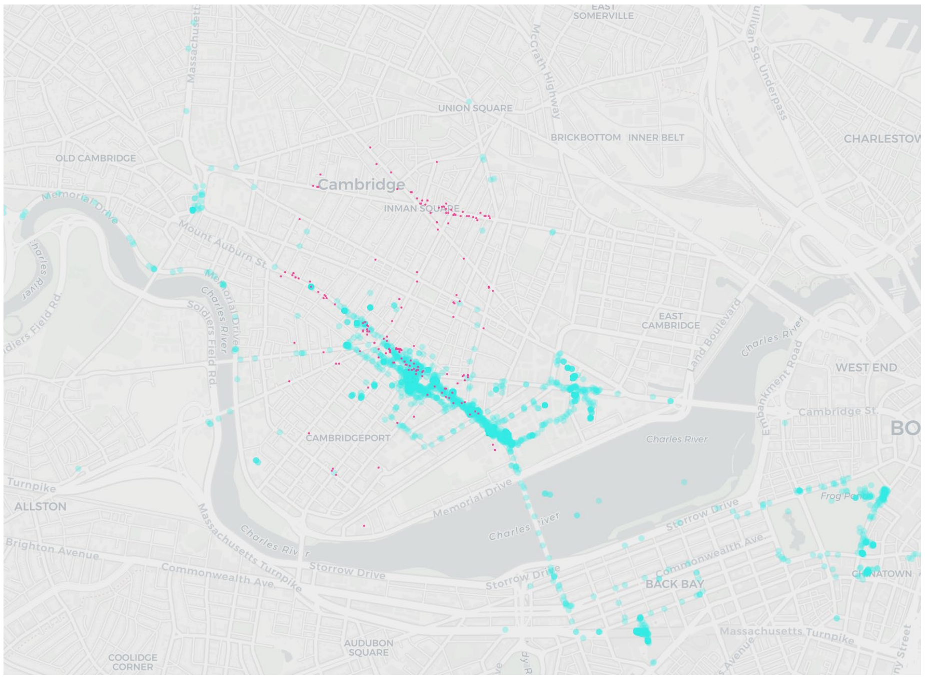
This is what your map should look like:



Remember, this map is interactive so you can zoom in or out as desired.

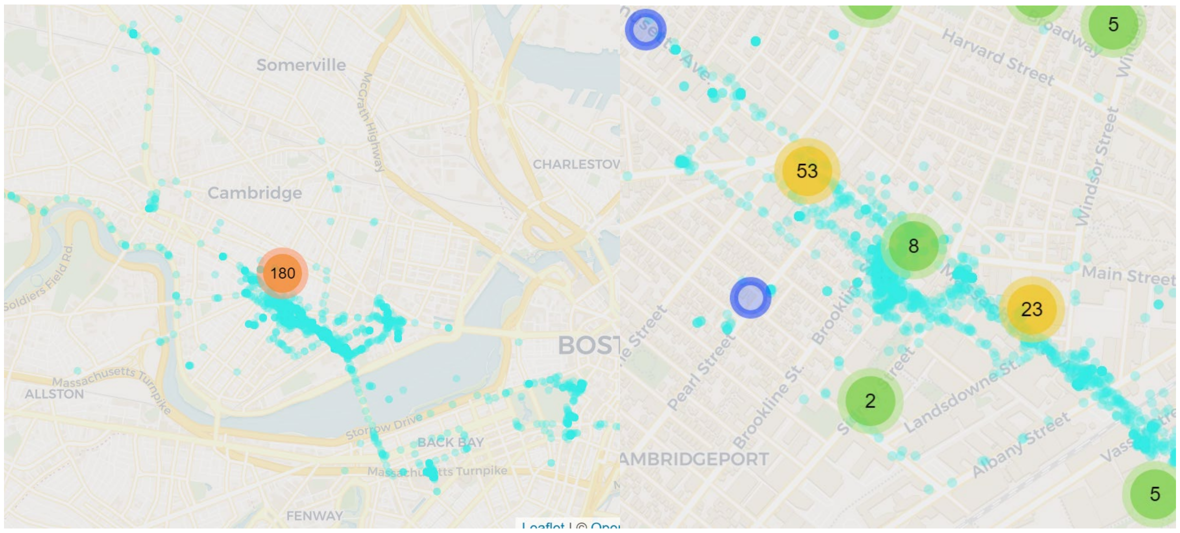
We can also add a restaurant layer to this map. There are a couple of ways to visualize this data. We can either use point data like below:

```
leafmap.febpoint<- leaflet(locfeb2020) %>%
  addProviderTiles(providers$CartoDB.Positron,
                   options = providerTileOptions(opacity = 0.5)) %>%

  fitBounds(  ~min(-71.144427), ~min(42.346422), ~max(-71.048083), ~max(42.39
8743))%>%
  addCircleMarkers( #adding google tracks
    stroke = FALSE, fillOpacity = .3,
    radius= r.goog,
    color= col.goog,
    lng = ~locfeb2020$lonGPS, lat = ~locfeb2020$latGPS
  ) %>%
  addCircleMarkers(   #adding yelp data

                     lng = ~yelp$longitude, lat = ~yelp$latitude,

                     stroke = FALSE, fillOpacity = 1,
                     radius= r.yelp,
                          color = col.yelp
  )

leafmap.febpoint
```



or we can ask leaflet to cluster our restaurants, which 'uncluster' as we zoom in.

```
#making leaflet map for February 2020 with clusters
leafmap.feb.cluster<- leaflet(locfeb2020) %>%
  addProviderTiles(providers$CartoDB.VoyagerLabelsUnder,
                   options = providerTileOptions(opacity = 0.5)) %>%

  fitBounds(  ~min(-71.144427), ~min(42.346422), ~max(-71.048083), ~max(42.39
8743))%>%
  addCircleMarkers( #adding google tracks
    stroke = FALSE, fillOpacity = .3,
    radius= radius,
    color= col.goog,
    lng = ~locfeb2020$lonGPS, lat = ~locfeb2020$latGPS
  ) %>%
  addCircleMarkers(   #adding yelp data

    lng = ~yelp$longitude, lat = ~yelp$latitude,
    clusterOptions = markerClusterOptions()
  )


leafmap.feb.cluster
```



As mentioned earlier, our aim is to figure out the access to restaurants the quarantined person has. For that we would need to know her home location. Looking at the mobility patterns do you have any idea where this person lives?

Note down any ideas you may have about how to find this information from the tracks.

We can tell you that this individual lives at **42.363065, -71.101517**. Now that you have this information, was your guess correct?

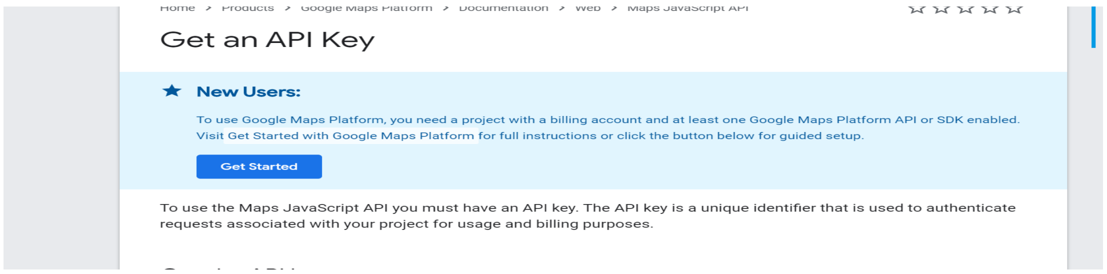
Calculating walking distance to restaurants

Setting up Google API

Now that we have the home location, we can use the googleway library to calculate walking distance from the home to every restaurant in the 1000m CSV we added to R.

To access the Google API we will first need a key, just like the API key we acquired for the Yelp API:

1. Go to, https://developers.google.com/maps/gmp-get-started and click on Get Started



Home > Products > Google Maps Platform > Documentation > Web > Maps JavaScript API

Get an API Key

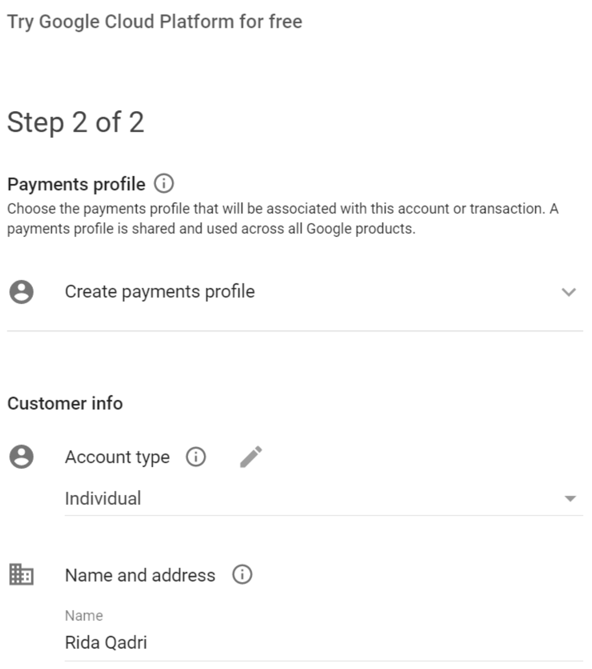
★ **New Users:**

To use Google Maps Platform, you need a project with a billing account and at least one Google Maps Platform API or SDK enabled. Visit Get Started with Google Maps Platform for full instructions or click the button below for guided setup.

Get Started

To use the Maps JavaScript API you must have an API key. The API key is a unique identifier that is used to authenticate requests associated with your project for usage and billing purposes.

2. Follow instructions to sign in to your Gmail account and set up a Billing Profile.


Try Google Cloud Platform for free

Step 2 of 2

Payments profile
Choose the payments profile that will be associated with this account or transaction. A payments profile is shared and used across all Google products.

Create payments profile

Customer info

Account type
Individual

Name and address
Name
Rida Qadri

3. Google gives you a year long free trial to access its platform so you wont be billed for this, you will just need to input a Debit or Credit card ***[if you do not have a debit or credit card or are uncomfortable using yours for this, please send us a note]***
4. After setting up your billing profile, visit https://console.cloud.google.com/google/maps-apis/overview and follow instructions to set up an API Key.
5. Your 'project' will now be associated with an API Key, and you can enable however many APIs you want for that particular project. Each API lets you access a different service, e.g. Google Places, Google Maps etc.
6. For this we will need the Distance Matrix API
7. To enable that go to : https://console.cloud.google.com/marketplace and search for Distance Matrix API in the search bar
8. Click on the resulting link and enable the API by clicking Enable

Using R and Googleway to calculate street network distance

We are now set up to find distances and walking duration using Google Map's built-in network analysis algorithm that gives you the estimate for travelling from one destination to another.

Below is a sample code which uses origin and destination lat/long to find walking distance and duration along the street network. Notice the inputs the google_distance function takes:

- 1 pair of origin lat/long
- 1 pair of destination lat/long

- mode (in this case walking)
- API Key

```
 # API Key for Google Distance
keyDist = 'insert key here'

# Example of distance between 1 restaurant and the house
distance.test <- google_distance(origins=c(42.363065,-71.101517),
                           destinations=c(42.3654,-71.10458) ,
                           mode =  "walking",
                           key=keyDist, simplify = TRUE)

print(distance.test$rows$elements)
```

Do you get a table that looks like this:

```
  distance.text distance.value duration.text duration.value status
1        0.4 km            413        6 mins            350     OK
```

Now that this works, we can set up a loop which allows us to calculate the distance from the quarantined individual's home to each restaurant. This loop first calls the Google server to find the walking distance, which returns a list with various elements. We extract the duration (in seconds) and distance (in meters) from this result and then add them to a temp dataframe which we can merge at the end with our Yelp dataset.

We set up a print statement in the loop so that we can track how our 'i' and lat long changes

The logic of the loop is:

- iterate across yelp dataset, printing i and lat/long for each iteration
- for each iteration, take the lat long pair and input that as destination
- for that pair, calculate distance to origin
- extract duration as seconds and distance as meters into variables
- Append each duration/distance pair to the temp dataframe

```
#setup an empty data container with rows equal to number of rows in our Yelp
dataset and 2 columns
temp<- matrix(nrow=nrow(yelp),ncol=2)

#set names of columns
colnames(temp) <- c('distance.meter', 'duration.minutes')

# set origin point as lat long of home
```

```r
origin = c(42.363061, -71.101564)
mode="walking"

for (i in (1:nrow(yelp))) #loop will run as many times as there are rows in y
elp dataset
  {
  #for each iteration of the loop, print updated i, and lat long
  print(paste0("i= ",i," Lat/Long: ",yelp$latitude[i],",", yelp$longitude[i])
)

  #pause loop call to server for .1s to not have Google block us for too many
calls
  Sys.sleep(.1)

   dist <- google_distance(origins=origin,
                          destinations= c(yelp$latitude[i], yelp$longitude[i]
),
                          key=keyDist,  mode =  mode, simplify = TRUE)

 #create variables duration and distance

  duration<-  as.numeric(unlist(dist$rows$elements[1])["duration.value"])
  distance<- as.numeric(unlist(dist$rows$elements[1])["distance.value"])

  temp[i,1]= distance
  temp[i,2]= duration/60.0 #to convert into minutes

}
#test if loop appended correctly by testing first result and last result of l
oop manually

distance.test <- google_distance(origins=origin,
                                 destinations=c(yelp$latitude[1], yelp$longit
ude[1]) ,
                                 mode = mode,
                                 key=keyDist, simplify = TRUE)
print(distance.test$rows$elements)

temp[1,]
```

That look's fine. Let's test the last result. In my case **its 182nd** result but you can use nrow(yelp) and nrow(temp) to see what your last result is. They both should be equal.

```r
distance.test <- google_distance(origins=origin,
                                 destinations=c(yelp$latitude[182], yelp$long
itude[182]) ,
                                 mode =  mode,
                                 key=keyDist, simplify = TRUE)
print(distance.test$rows$elements)

temp[182,]
```

Now we can append our results to the yelp dataset. Since the order of temp is the same as the order of yelp data we can just go ahead and do R's version of copy paste

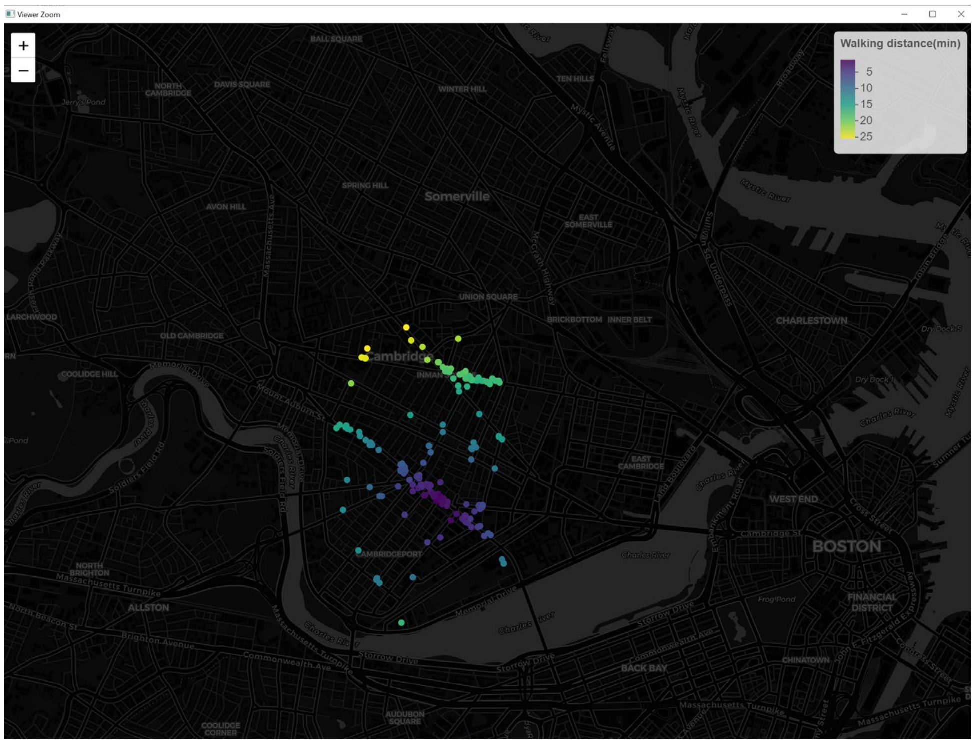
```
yelp.final<-cbind(yelp,temp)

#Create a final dataset which only includes restaurants which have a value fo
r duration. minutes
yelp.final<-subset(yelp.final, !is.na(duration.minutes))
```

Save this file as a csv using the write.csv command we used in the last lab.

Now we can map the restaurants, symbolized by the minutes it takes to walk to them from the point of origin.

```
pal <- colorNumeric("viridis", NULL) #COLOR scale for the distance

leaflet(yelp.final) %>%
  addProviderTiles(providers$CartoDB.DarkMatter,
                   options = providerTileOptions(opacity = 1)) %>%

    fitBounds(  ~min(-71.144427), ~min(42.346422), ~max(-71.048083), ~max(42.
398743))   %>%

  addCircleMarkers(stroke = FALSE, fillOpacity = 1, fillColor = ~pal(yelp.fin
al$duration.minutes),
                   radius = 4, lng = ~yelp.final$longitude, lat = ~yelp.final
$latitude ) %>%
  addLegend(pal = pal,   title = "Walking distance(min)", values = ~yelp.fina
l$duration.minutes, opacity = 0.8)
```

Your map will look something like this:



And that's it.

Lab Assignment (Due Monday, April 26)

We have given you a sample R script above which you can edit to answer the following questions:

1. From the loc2020 dataset, extract datapoints from starting March 12, 2020 (including March 12). We have included a hint in the lab above to help you with how to do this.
   - How many observations are in the February 2020 dataset and how many in your partial-March dataset?
3. Create an interactive leaflet map using the newly extracted March dataset and add a screenshot of the map to your answers. Make sure you include in the map :
   - Google tracks for the partial-March dataset
   - Restaurant locations from the 1000m csv
4. Compare mobility patterns from February and March. In one or two sentences, what do you notice about the changing patterns if any?
5. Examine the March mobility patterns. If we had not given you the location of this person's home would you have been able to guess the location using the google track data? In a few sentences, note down 2 ways in which you could use the google dataset (using any or all the variables, location, time, day, etc.) to find out where this person lives?
6. Include a screenshot of the first 5 rows of your **final_yelp dataset** that has the restaurant data and duration/distance information.
7. Include a screenshot of your map with restaurants symbolized by distance/duration.

8. Using the final_yelp dataset, write a brief message to this person telling her about:
   - How many restaurants she can visit in under a 10 minute walk
   - Your recommendation for a few restaurants that are close (under 15 mins walking distance) are well-rated and moderately priced (mention your interpretation of these variables).

Submit all screenshots, maps and answers as ***a single PDF or Word document*** to Stellar.

Appendix

This is how we accessed and cleaned the JSON file of all location history in R:

```
#set working directory
setwd('C:/Users/ridaq/Dropbox (MIT)/PHD-Work/Dissertation_Files/Computational
Data/Driver Spatial Tracks/RidaGoogleApr')

#get json file.
loc.json= fromJSON(txt = "rida.json")

#extract locations element from list and make dataframe

google.locations <- loc.json$locations

#set system time to be able to create date from milisecond count in the datas
et
Sys.setlocale("LC_TIME", "C")

#create date and timestamp by counting ms since 1970
google.locations$time <- as.POSIXct(as.numeric(google.locations$timestampMs)/
1000, origin="1970-01-01")


#create day

google.locations$day <- factor(format(google.locations$time, "%a"),
                                    levels = c("Mon", "Tue", "Wed", "Thu", "F
ri", "Sat", "Sun"))

#add year and month and date columns by extracting relevant string form timek
eeping column

google.locations<-transform(google.locations, year = substr(timekeeping, 1, 4
))
google.locations<-transform(google.locations, month = substr(timekeeping, 6,
7))
google.locations<-transform(google.locations, date = as.numeric(substr(timeke
eping, 9, 10)))



#converting e7 to gps
google.locations$latGPS <- as.numeric(google.locations$latitudeE7/1E7)
google.locations$lonGPS <- as.numeric(google.locations$longitudeE7/1E7)
```

```r
#creating subset dataset which only has locations
loc.only<-subset(google.locations, select=c("time", "velocity", "day", "year"
, "date",
                                            "month", "latGPS","lonGPS","accur
acy"))

#now subset larger dataset into February and March 2020  dataset
loc2020<-loc.only[which(loc.only$year=='2020' & loc.only$month  %in% c("02","
03")), ]

write.csv(loc2020,'loc2020.CSV')
```