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 1)
Webscraping using R

April 12, 2021

Aims of this lab

- Learn how to merge various datasets to analyze a rich urban problem or phenomenon
- Use APIs to scrape data from a website
- Use Google location data to understand mobility patterns
- Consider the value and limitations of using data not created for the express purpose of research

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.

Scraping Data

Often, information is available online, but not in publicly accessible datasets or in easily analyzable formats. Web scraping allows you to extract information from websites and store it in a useable way. You can scrape data from websites like Zillow, Google Places, Tripadvisor. In some cases, website owners provide application programming interfaces (APIs) that establish a protocol for requesting batches of information from the website.

In the first part of the lab, we will use Yelp APIs to identify points of interest within the Cambridge squares that we will then re-visit in Part 2 during virtual field work trips using Google!

There are a myriad of programs you can use to access APIs such as Yelp's, but we'll use a specific R library in this lab.

R is a powerful and popular open-source suite of statistical tools that supports many freely available software 'libraries' to assist in web scraping, data parsing, and outputting of data in standard formats that can be read by GIS and data analysis software.

To facilitate your use of R, we will use an IDE called RStudio which allows you to write, edit and execute your R script. We will also provide you with an R script that is set up to extract data from Yelp using their application programming interface (API) protocols.

Download R and R Studio

1. Go to https://cran.r-project.org/mirrors.html
2. Click on any CRAN location (a mirror site)
3. Click on the respective "Download R for…" link which corresponds to your OS
4. Click on 'base' and follow the installation instructions.

Now that R is installed, you need to download and install RStudio.

You can download the free version of R Studio Desktop here:
https://rstudio.com/products/rstudio/download/

Initializing R Document

We start our R script by installing a 'library' as a 'package'. These are basically a list of pre-programmed functions written by the R community (even you can write one!) to easily help us undertake computation we otherwise would have had to create code for from scratch

For the first part of the lab we'll install two libraries: 'devtools' and 'yelpr'.

To start the document we'll need to set up our working directory space. Start Rstudio and follow these instructions: [Note: lines starting with '#' are comments.]

Since you have write-access to R's default workspace folder in your C drive, you can go ahead and install the following packages if you haven't done so already:

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

#install devtools
install.packages("devtools")

#run command to download/install yelpr library directly from its github repos
itory
devtools::install_github("OmaymaS/yelpr")
```

We can now initialize our R session by telling R to 'attach' certain libraries to this session, since we will be making use of functions in them.

Downloading and installing the libraries needs to be done only once on one machine. During subsequent sessions using R on the same machine, you can begin with this next portion of the script to utilize the already-installed libraries.

```
#initiate library devtools
library("devtools")

## Warning: package 'devtools' was built under R version 3.3.3

#initiate library yelpr
library(yelpr)
```

Set up API Key

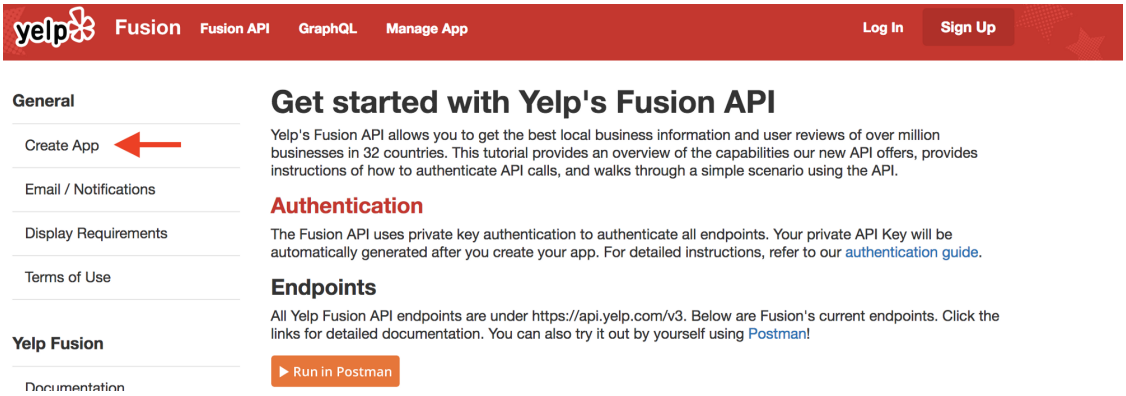
To scrape data from public websites we often make use of something called an API. APIs are interfaces that allow different software to talk to each other. In this case, an API will allow R to talk to Yelp's data server and make a request for specific data based on the parameters that we provide. In response Yelp's server provides an XML tagged list, which we then 'translate' into a format suitable for manipulation in R so that, eventually, we can write the data into a text file that can be viewed in a spreadsheet, ArcMap, and other applications.

To make this call we need something called an API key. This is a form of identification provided by the websites we are accessing like Yelp, so that they can keep track of the calls we make to their server. It also keeps the provider (Yelp in our case) in control of who is allowed to access their services and is often used to limit the daily or weekly number of requests that will be allowed for free.

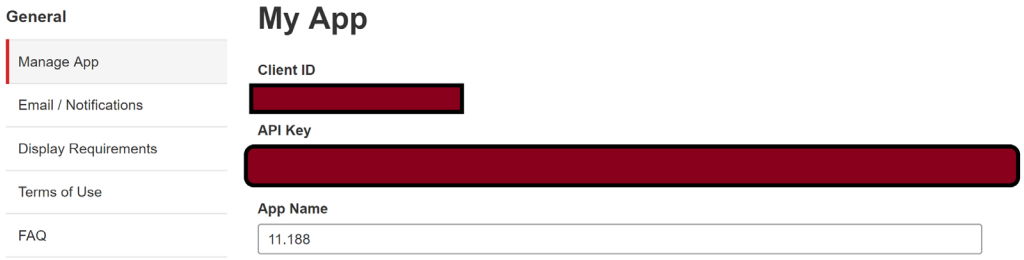
To obtain our unique API key, we will go to Yelp's developer section: (https://www.yelp.com/developers/documentation/v3/authentication), and request an API Key for Yelp Fusion API. This API key is a long string of characters that is unique for you and must be inserted into each of your requests for data from Yelp.

The first step is to create an 'app' for yourself. You may be asked to log in or create a Yelp account before you are allowed access to the App creation page shown below.

The App creation Yelp page may look something like this:



When you first sign up, you will receive an email link that must be clicked to verify your account. Once activated, you must return to this page to create your own 'app' and, then, your API key and Client ID will be displayed of the top of the Yelp page.



This API Key is your own personal identification. Do not share this key with anyone outside your group, because each key has a limited number of 'calls' to the server it can make. You don't want to blow through your calls (though for this lab that is unlikely to happen)

Now that you have your key we are ready to get started.

Making calls to the Yelp server for data

We start our script by defining some parameter variables. This basically means we are telling R that there are some parameters like location, search radius etc. we would like to define our search with just like putting keywords in a Google Search.

It is handy to define these at the start as 'variables' which you can then refer to throughout your document. This way if you need to change something you can do so in one place.

In this case we will search within 400m of Kendall Square T stop's location for the term 'food'. An easy way to get the lat/lon of the T stop is to use maps.google.com to view Kendall Square, then right-click on the station stop, and select 'What's here'.

There are other ways to find lat/lon values and we can discuss them later.

Remember anything within single quotes (‘ ’) or double quotes (" “) will be treated as character strings, while objects without quotes will be treated as variables or integers. You can comment your code by using # to have the rest of that line ignored by the script, or you can enclose multiple lines of text within double quotes (” “), as long as the same quote character is not included in the intervening lines.

```
key<- 'Insert your API Key within these quotation marks' #insert API key

keyword<- 'food' #term to search for
lat<- 42.362491 #latitude of your location
long<- -71.085577 #longitude of your location
rad<- 400 #radius of search in meters
limit<-50 #the number of results on one page you would like to see. The max f
or one page is 50
```

Now that we have defined the variables we will use, we are ready to discuss our basic script to search Yelp API for all businesses that are tagged with the word of your choice within 400m of Kendall Square. Up to 50 entries will be returned from a single request. You can search for **food** or **restaurants** to see the difference.

The function called business_search() uses the ‘yelpr’ library to request information about all businesses close to the lat/lon that you provide. Since we need to store our results for later use, we save the XML-tagged data returned by Yelp into a temporary dataframe and call it ‘results’. The ‘<-’ notation in R is used to save the data to the right of the symbol in a dataframe with the name on the left. In our case, whatever is returned from Yelp will be saved into an R dataframe to which we assign the name ‘results’.

```
results<-business_search(api_key = key,
                         term = keyword,
                         latitude = lat,
                         longitude = long,
                         radius=rad,
                         limit=limit)

## No encoding supplied: defaulting to UTF-8.
```

The line beginning with ‘##’ is a message to you from R. It indicates that R will use the default UTF-8 specification for character encoding when interpreting the data in your new dataframe. You may ignore this warning.

The portion of the data returned from Yelp that we want to keep is tagged as ‘businesses’. We can reference that portion of our ‘results’ dataframe by using the notation:

```
results$businesses
```

We can now store this component as a new dataframe that will be easier for us to manipulate.

```
res_store<-results$businesses
```

Lets look at the first 5 rows of the res_store dataframe we just created to get a better sense of what data we're dealing with. Notice the notation. When you add brackets, [ ], next to the name of a dataframe that means you would like to point towards specific rows or columns in the dataframe.

```
res_store[1,] #first row of res_store
```

```
##                        id                         alias            name
## 1 wrMG0WYZKIZX9Zp30kCFGA clover-food-lab-cambridge-8 Clover Food Lab
##                                                                image_url
## 1 https://s3-media4.fl.yelpcdn.com/bphoto/1FZMdYFh3t12_6RzEKnusQ/o.jpg
##   is_closed
## 1     FALSE
##
url
## 1 https://www.yelp.com/biz/clover-food-lab-cambridge-8?adjust_creative=Iu_
wBl8JK0u_dhtjknjLOg&utm_campaign=yelp_api_v3&utm_medium=api_v3_business_searc
h&utm_source=Iu_wBl8JK0u_dhtjknjLOg
##   review_count
## 1          261
##                                                        categories rating
## 1 sandwiches, cafes, newamerican, Sandwiches, Cafes, American (New)      4
##   coordinates.latitude coordinates.longitude transactions price
## 1             42.36026             -71.08655                  $$
##   location.address1 location.address2 location.address3 location.city
## 1   5 Cambridge Ctr                                        Cambridge
##   location.zip_code location.country location.state
## 1             02139              US             MA
##                location.display_address phone display_phone distance
## 1 5 Cambridge Ctr, Cambridge, MA 02139                      147.2364
```

'res_store' dataframe is basically a spreadsheet table where each row has the same structure and the columns contain numbers or strings with various meanings. Here are the commands to examine just the first column, the first row, the first five rows, etc.

```
res_store[,1]#first column of res_store
res_store[1,1]#first row, first column
res_store[1:5,] #first 5 rows of  res_store
```

However notice how many unneccessary columns we have which we don't need for our analysis. So we can tell R to recreate our dataframe with just the columns we need.

```
res_neat<-res_store[,c("name","review_count","rating","price")]
```

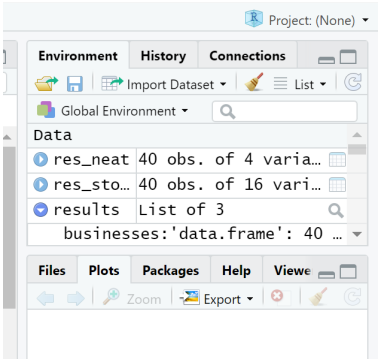
Lets look at the first 5 rows of this new dataframe now

```
res_neat[1:5,]
```

```
##                  name review_count rating price
## 1     Clover Food Lab          261    4.0    $$
## 2                CAVA           17    4.5  <NA>
```

```
## 3 Tatte Bakery & Cafe          508    4.0    $$
## 4                Vester           34    4.0  <NA>
## 5   Dumpling Daughter             60    3.0  <NA>
```

You can also examine the newly created data frame by going to the 'Environment' tab and click on the dataframe you would like to explore



name	review_count	rating	price
Clover Food Lab	283	4.0	$$
CAVA	46	4.5	NA
Shy Bird	64	3.5	$$
Vester	48	4.0	$
Dumpling Daughter - Cambridge	92	3.0	NA
Aceituna Grill	323	4.0	$$
Chipotle Mexican Grill	99	2.0	$
Za	267	3.5	$$
Legal Sea Foods	475	3.0	$$
Tatte Bakery & Cafe	546	4.0	$$

That looks better, though we do need important location information which we didn't include! If you look at the original dataframe res_store we can see location and coordinate information stored, though column names have a period (.) in the middle: e.g., coordinates.longitude

coordinates.latitude	coordinates.longitude	transactions	price	location.address1
42.36282	-71.08757	character(0)	$$	5 Cambridge Ctr

This indicates that these are stored in a subsection of the original list that we can access directly.

```
coord<-results$businesses$coordinates

add<-results$businesses$location$address1 #accessing the column address1 stor
ed in table location, stored in the businesses component of 'results'
```

We can now merge all these columns together in one final dataset through the command cbind which adds columns to datasets. **cbind** appends columns together or binds columns hence cbind.

```
#combine columns in res_neat , coord , add together
```

```
yelp_data_combine<-cbind(res_neat,coord,add)

yelp_data_combine[1:5,]
```

```
##                   name review_count rating price latitude longitude
## 1     Clover Food Lab          261    4.0    $$ 42.36026 -71.08655
## 2                CAVA           17    4.5  <NA> 42.36321 -71.08796
## 3 Tatte Bakery & Cafe          508    4.0    $$ 42.36492 -71.08267
## 4              Vester           34    4.0  <NA> 42.36186 -71.08831
## 5   Dumpling Daughter           60    3.0  <NA> 42.36186 -71.08831
##                add
## 1 5 Cambridge Ctr
## 2      82 Ames St
## 3    318 Third St
## 4      73 Ames St
## 5      73 Ames St
```

You'll notice though that our results were limited to ''50' rows. This did not pose a problem with this particular exercise since we only seemed to have 42 results but could be a problem with other places which are more dense.

So for that, we would need to insert something called offset which is a way to tell the Yelp API to basically turn the page. If our limit (i.e. page size) is 50, we would add an offset of the same size, until we reach the end of our search, or return 1000 results (which is the maximum allowed by Yelp for one search of their API).

However, for larger searches constantly adding offset then saving it in a dataset, then doing it all over again and again would be cumbersome. So we can write a loop which would ask R to extract data from Yelp in batches of 50, and store them in a dataset till we get an empty call or hit a 1000 results.

So lets try that.

We are writing a 'for' loop which assigns 'offset' a value every time it runs. The seq function defines (start value, end value, increment) so offset would be 0, 50, 100, 150 etc. till we hit an empty call or 1000 searches. In our case we will hit an empty call right after the first call has been executed.

Notice that within the for-loop we have nested an if-statement, which will continue to add more data to our dataset until temp$businesses has a length of 0

```
loop_yelp<-data.frame() #creating an empty dataframe


for (offset in seq(0,1000,50)) {

      #temporary storage container for our call to the server
      temp <- business_search(api_key = key,
                              term = keyword,
                              latitude = lat,
```

```
                                longitude = long ,
                                radius=rad,
                                limit=limit,
                                offset=offset)

      #use if statement to execute the next part if temp$businesses is not em
pty

      if (length(temp$businesses)!=0) {

                #store  results of call in a dataframe called temp1
                temp1<-temp$businesses

                #select columns we want
                temp1<-temp1[,c("name","review_count","rating","price")]

                #retrieve coordianates and address
                geom<-temp$businesses$coordinates
                add<-temp$businesses$location$address1

                #bind columns together
                merge<-cbind(temp1,geom,add)

                #append rows generated by the loop to the loop_yelp dataframe
                loop_yelp<-rbind(loop_yelp,merge)

                }

      # loop to input data into loop_yelp is completed  when you get this far



}  # end of the outer 'for' loop
```

We can now compare both datasets: the one we got with the loop and without the loop. One way to do this is check their 'dimensions'. In this case since we had less than 50 results to scrape, the datasets both would be the same:

```
dim(yelp_data_combine)

dim(loop_yelp)
```

You can now save your final dataframe in a comma separated value (CSV) format.

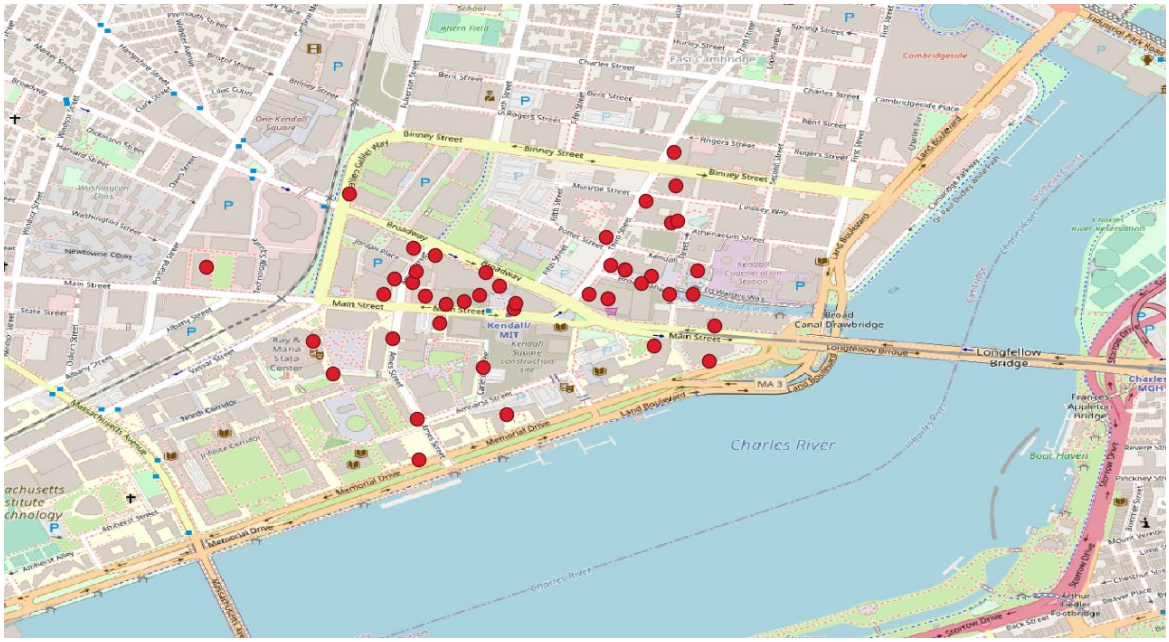
```
write.csv(loop_yelp, file = "Yelp_kendall.csv")
```

Your file will be saved in your working directory. If you are not sure where this directory is located, type the command getwd() to display the path. If you want to change the working directory location, use the setwd() command.

Displaying results in QGIS

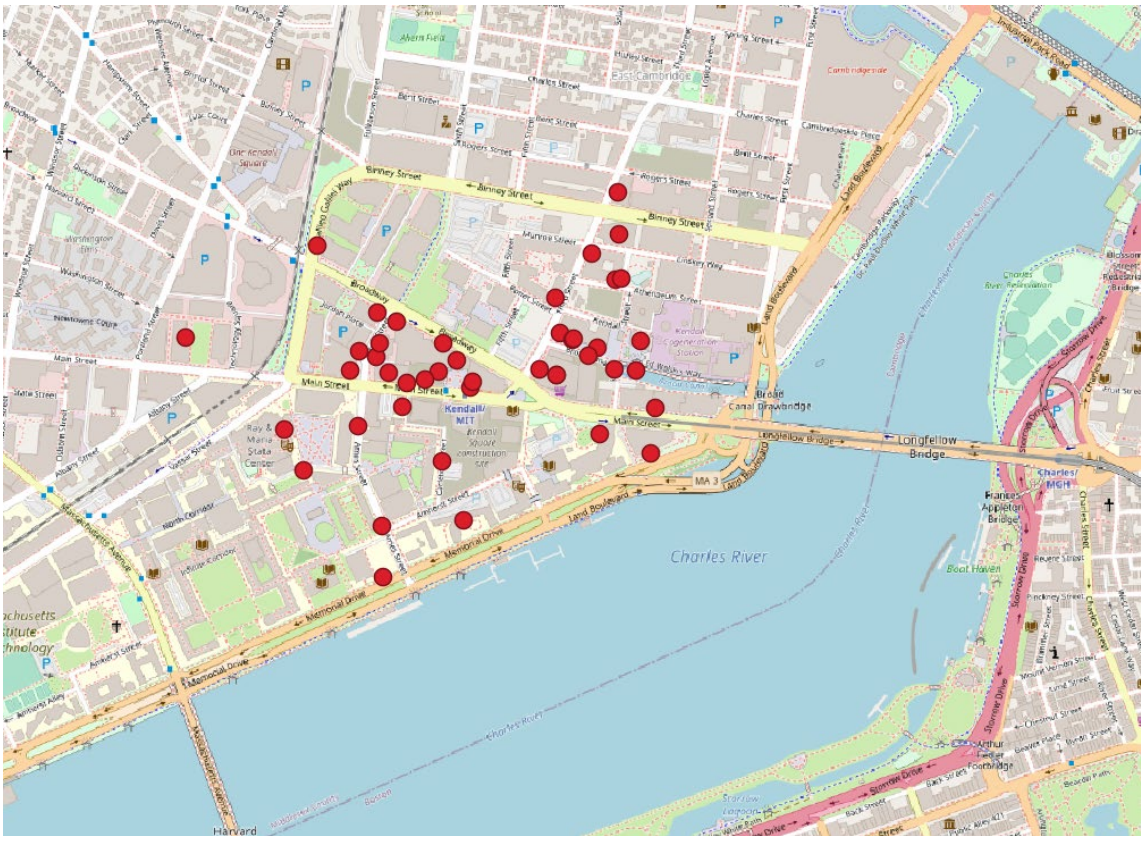
So far in class we have only used pre-made shapefiles to display point data. However, you can create your own shapefile as long as you have lat/lon data available.

- Click on the "Open Data Source Manager" icon in QGIS (shortcut: Ctrl + L)
- Select the "Delimited Text" option on the left of the pop-up window
- Browse to the "Yelp_kendall.csv" file on your local machine for the "File name"
- The default encoding is UTF-8 and the default file format is CSV. Both of these are fine for comma separated value (CSV) files, such as "Yelp_kendall.csv".
- Expand the "Geometry Definition" option and select the "longitude" column as your "X field" and the "latitude" column as your "Y field".
    - ***Why is longitude ~ X and latitude ~ Y?***
- Set the "Geometry CRS" to WGS 84 (EPSG 4326)
- On clicking the "Add" button, a new point shapefile "Yelp_kendall" should be visible on your QGIS project window.
- For context, add a background layer. Click on "Web" on the top ribbon, select "QuickMapServices" and browse to "OSM". Expand the options and select "OSM Standard". You can choose other background layers as well.
- You should see the following:



- Remember that these points are restaurants located within a 400-meter buffer of the Kendall Square T station. Do you notice anything odd?

- Change the project CRS to NAD83 / Massachusetts Mainland (EPSG 26986). You should see the following:



- Much better! The farthest restaurant to the west looks to be within 400 meters now. It's all about visual perception!

- Don't forget to save this shapefile! Right click on the point layer in the QGIS Layers window, select "Export", then "Save Features as". In the pop-up window, be sure to change the "Format" to "ESRI shapefile". Set the "File name" (and location) and the "CRS" based on your preferences.

Lab Assignment

We have given you the script for scraping restaurant data in Kendall Square. You should now change the script to search for restaurants in/around ***Central Square*** and analyze the data. Think about how you would do this.

Part 1: Analyzing the data (Due Wednesday, April 21)

1. Create two CSV files, one for restaurants within a 400m and one for restaurants within a 1000m of the Central Square T station.
2. For both datasets note down:
    1. Number of restaurants
    2. The average user rating for all restaurants in the radius

3. The average price level for all restaurants in the radius
4. Also create a frequency table of :
    - Number of restaurants in each price category for each radius
    - Number of restaurants for each 'rating' level for each radius

3. Use the 400m csv created through the exercise to create a map of the restaurants:
    - Show one map with restaurants with symbols classified by price level
    - Show one map with restaurants with symbols classified by user ratings
    - Make sure you take care of the map presentation basics we have focused on during the first half of the semester

Submit your answers and map in ****a single PDF or Word document**** to Stellar by **April 21 2021.**