Data Visualization Script: Geocoding and Routing

Matthew L. Sisk

Updated: November, 2022

Geocoding and routing

In this demonstration we will experiment with using the ggmap package to geocode addresses and to get travel times and routes between locations.

First, we need to load the ggmap package. ggmap is a powerful interface between R and google maps. More information on what you can do with it can be found here.

```
library("ggmap")
```

```
## Loading required package: ggplot2
```

```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

```
## Please cite ggmap if you use it! See citation("ggmap") for details.
```

Here, we will register a Google API key. The is one I set up for this class and it will not last beyond the end of the semester, so I would recommend setting up your own if you need geocoding or routing tools. There are a certain number of free api calls per month and a generous trial period

```
register_google(key = "YOUR_API_KEY_HERE")
```

You are welcome to use these two addresses here in South Bend, or to use your own.

```
house <- "712 West LaSalle Ave, South Bend IN 46601"
work <- "Hesburgh Library, University of Notre Dame"
```

Interactive Geocoding

Next, lets try using the geocode function to get the latitude and longitude for each of these points.

```
geocode(house)
```

```
## Source : https://maps.googleapis.com/maps/api/geocode/json?address=712+West+LaSalle+Ave,+South+Bend+
```

```
## # A tibble: 1 x 2
##     lon   lat
##   <dbl> <dbl>
## 1 -86.3  41.7
```

You can see that this works by calling the Google Maps API and parsing the results into something R can use.

With the geocode function, we have a lot of variables that we can set, most relevantly, we can either use Google's geocoder or the Data Science Kit with the source variable. In most cases it is better to use google (which is the default)

Probably most useful for you is the "output" variable. The default is to just return the coordinates, but google can also can return a measure of how good the estimate is. This is particularly useful in cases where someone entered the address and there might be errors. So lets try doing it with a bad address

```
geocode("100 Totally Fake Street, South Bend IN 46617")
```

```
## Source : https://maps.googleapis.com/maps/api/geocode/json?address=100+Totally+Fake+Street,+South+Be
```

```
## # A tibble: 1 x 2
##     lon   lat
##   <dbl> <dbl>
## 1 -86.2  41.7
```

With the default, this (obviously fake) address yields a normal lat lon pair. So it seems like it worked without a problem

```
geocode("100 Totally Fake Street, South Bend IN 46617", output = "latlona")
```

```
## Source : https://maps.googleapis.com/maps/api/geocode/json?address=100+Totally+Fake+Street,+South+Be
```

```
## # A tibble: 1 x 3
##     lon   lat address
##   <dbl> <dbl> <chr>
## 1 -86.2  41.7 south bend, in 46617, usa
```

Using output = "latlona" we can see what address the geocoder found. In this case, just the name of the city

```
geocode("100 Totally Fake Street, South Bend IN 46617", output = "more")
```

```
## Source : https://maps.googleapis.com/maps/api/geocode/json?address=100+Totally+Fake+Street,+South+Be
```

```
## # A tibble: 1 x 9
##     lon   lat type        loctype     address             north south  east  west
##   <dbl> <dbl> <chr>       <chr>       <chr>               <dbl> <dbl> <dbl> <dbl>
## 1 -86.2  41.7 postal_code approximate south bend, in 46~  41.7  41.7 -86.2 -86.3
```

With output = "more" or output = "all" we get a measure of how accurate the geocoding was (in this case type = postal_code and loctype = approximate) indicating that it was not very accurate. It would now be easy to filter these results.

A very common problem with batch geocoding is data scientists not taking into account the accuracy of the geocoding. This can result in all of the failed geocodes ending up in the same area (like the geometric center of the city) and can very strongly affect the distribution.

Batch Geocoding

The geocode function can also take a vector of addresses with little trouble, but even more useful is the mutate_geocode version, which works like a dplyr mutate.. To experiment with this, we will create a dataframe with addresses. For part of your assignment this week, you will need to load a csv. Next, we can run the mutate_geocode command over the whole data.frame. Note that sometimes you will have errors in a large dataframe of addresses and one weekness of mutate_geocode is that it will fail for all if it encounters a single error. In these cases, you may want to do the geocoding in a loop.

```
address.df <- data.frame(address = c("712 West LaSalle Ave, South Bend IN 46601", "Hesburgh Library, Uni

address.df.geocoded <- address.df %>%
  mutate_geocode(address, output = "more")
```

```
## Source : https://maps.googleapis.com/maps/api/geocode/json?address=712+West+LaSalle+Ave,+South+Bend+

## Source : https://maps.googleapis.com/maps/api/geocode/json?address=Hesburgh+Library,+University+of+N

## "Hesburgh Library,..." not uniquely geocoded, using "284 hesburgh library, notre dame, in 46556, usa

## Source : https://maps.googleapis.com/maps/api/geocode/json?address=Notre+Dame+Stadium&key=xxx

## Source : https://maps.googleapis.com/maps/api/geocode/json?address=1047+Lincoln+Way+West+South+Bend,

## Source : https://maps.googleapis.com/maps/api/geocode/json?address=227+W+Jefferson+Blvd+South+Bend,+

## New names:
## * ‘address‘ -> ‘address...1‘
## * ‘address‘ -> ‘address...7‘
```

Sometimes you will run into problems with the google geocoding api returning “geocode failed with status OVER_QUERY_LIMIT” even when you running the `geocodeQueryCheck()` command shows you have not exhausted your available queries. This is often because you are sending too many queries (or someone else on your domain) per second to the Google API. The workaround is the same as above, you can iterate through manually and geocode each address in the data frame, pausing for a few seconds if it fails.

```
for(i in 1:nrow(address.df)){
  result <- geocode(as.character(address.df$address[i]), output = "latlona", source = "google")
  while(is.na(result[1])){ #checks if the latitude is NA and reruns if it is
    Sys.sleep(2) #Pauses for a minute to let the API Catch up
    result <- geocode(as.character(address.df$address[i]), output = "latlona", source = "google")
  }
  address.df$lon[i] <- as.numeric(result[1])
  address.df$lat[i] <- as.numeric(result[2])
  address.df$geoAddress[i] <- as.character(result[3])
}
head(address.df)
```

```
##                                        address id       lon      lat
## 1  712 West LaSalle Ave, South Bend IN 46601  1 -86.26012 41.67877
## 2 Hesburgh Library, University of Notre Dame  2 -86.23415 41.70239
## 3                         Notre Dame Stadium  3 -86.23392 41.69840
## 4    1047 Lincoln Way E South Bend, IN 46601  4 -86.23598 41.66439
## 5  227 W Jefferson Blvd South Bend, IN 46601  5 -86.25308 41.67541
##                                                               geoAddress
## 1                          712 w lasalle ave, south bend, in 46601, usa
## 2                       284 hesburgh library, notre dame, in 46556, usa
## 3                      2010 moose krause cir, notre dame, in 46556, usa
## 4                        1047 e lincolnway e, south bend, in 46601, usa
## 5 county-city building, 227 w jefferson blvd, south bend, in 46601, usa
```

This is a brute force way of doing it. As the code currently is, what do you think would happen if it encountered an address it could not find?

If you this approach in your assignment, you may want to ensure that it does not run indefinitely.

Routing

We can also get a distance (and travel time) between two points using the mapdist function

```
library(ggmap)
mapdist(from = house, to = work)
```

```
## Source : https://maps.googleapis.com/maps/api/distancematrix/json?origins=712+West+LaSalle+Ave,+Sout
```

```
## # A tibble: 1 x 9
##   from                      to        m    km miles seconds minutes hours mode
##   <chr>                     <chr> <int> <dbl> <dbl>   <int>   <dbl> <dbl> <chr>
## 1 712 West LaSalle Ave, Sou~ Hesb~  6093  6.09  3.79     719    12.0 0.200 driv~
```

The default is with driving, but we can change it to public transportation, walking, or riding a bicycle using “mode” set to one of (“driving”, “walking”, “bicycling”, “transit”)

```
mapdist(from = house, to = work, mode = "bicycling")
```

```
## Source : https://maps.googleapis.com/maps/api/distancematrix/json?origins=712+West+LaSalle+Ave,+Sout
```

```
## # A tibble: 1 x 9
##   from                      to        m    km miles seconds minutes hours mode
##   <chr>                     <chr> <int> <dbl> <dbl>   <int>   <dbl> <dbl> <chr>
## 1 712 West LaSalle Ave, Sou~ Hesb~  4858  4.86  3.02    1013    16.9 0.281 bicy~
```

We can also create a path that shows the steps between two locations using the route command

```
towork <- route(from = house, to = work, mode = "bicycling", alternatives = T, structure = "legs")
```

```
## Source : https://maps.googleapis.com/maps/api/directions/json?origin=712+West+LaSalle+Ave,+South+Benc
```

```
head(towork)
```

```
## # A tibble: 6 x 11
##       m    km  miles seconds minutes   hours start_lon start_lat end_lon end_lat
##   <int> <dbl>  <dbl>   <int>   <dbl>   <dbl>     <dbl>     <dbl>   <dbl>   <dbl>
## 1   962 0.962 0.598      213   3.55  0.0592      -86.3      41.7   -86.2    41.7
## 2   317 0.317 0.197       64   1.07  0.0178      -86.2      41.7   -86.2    41.7
## 3    18 0.018 0.0112       6   0.1   0.00167     -86.2      41.7   -86.2    41.7
## 4   535 0.535 0.332      107   1.78  0.0297      -86.2      41.7   -86.2    41.7
## 5    62 0.062 0.0385      14   0.233 0.00389     -86.2      41.7   -86.2    41.7
## 6   273 0.273 0.170       83   1.38  0.0231      -86.2      41.7   -86.2    41.7
## # ... with 1 more variable: route <chr>
```

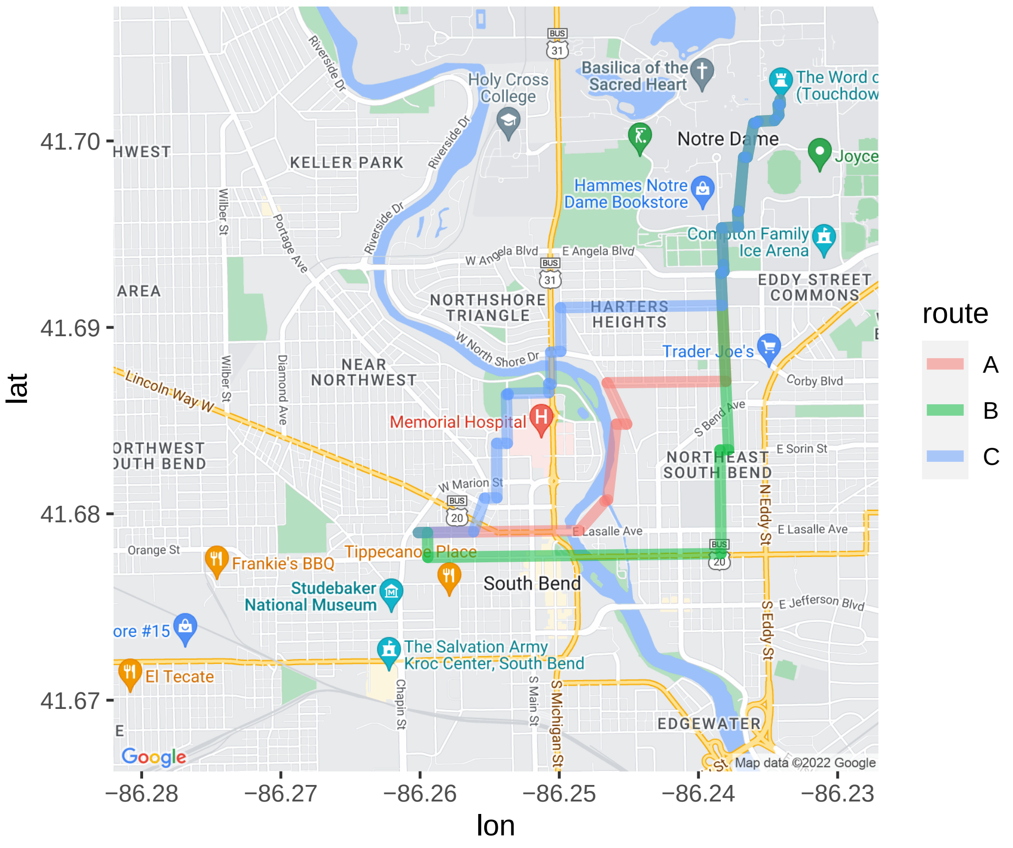
We can then plot this route with a little bit of manipulation in ggmap. First, we will use the get_map command to cache basemap tiles from google. Then we will generate lines from the start and end each leg and color them based on the route

```
southbend <- get_map("leeper park, South Bend, IN", zoom = 14, maptype = "roadmap")
```

```
## Source : https://maps.googleapis.com/maps/api/staticmap?center=leeper%20park,%20South%20Bend,%20IN&zo
```

```
## Source : https://maps.googleapis.com/maps/api/geocode/json?address=leeper+park,+South+Bend,+IN&key=x
```

```
ggmap(southbend)+
  geom_leg(aes(x = start_lon, y = start_lat, xend = end_lon, yend = end_lat,colour = route),alpha = .5,
```



Alternate Packages

The *tidygeocoder* package is also a very good option for doing geocoding. It can use the google geocoder, but also makes heavy use of the US census geocoder, which does not require an API Key! geo will call a single address, and geocode (be careful if you have both ggmap and tidygeocoder loaded since the two functions are ambigious)

```
library(tidygeocoder)
```

```
##
## Attaching package: 'tidygeocoder'

## The following object is masked from 'package:ggmap':
##
##     geocode
```

```
geo("University of Notre Dame, Notre Dame IN 46556")
```

```
## Passing 1 address to the Nominatim single address geocoder

## Query completed in: 1 seconds

## # A tibble: 1 x 3
##   address                                         lat  long
##   <chr>                                         <dbl> <dbl>
## 1 University of Notre Dame, Notre Dame IN 46556  41.7 -86.2
```

You can also do batch geocoding with the tidygeocoder. However, you will notice that it is both slower and also less flexible (it failed for Hesburgh Library) than the google geocoder. For well structured data, it may have similar results. So if you do not have the ability to use the paid service (google) it is a viable option for your homework

```
tidy.geocoded<- geocode(address.df, address = address)
```

```
## Passing 5 addresses to the Nominatim single address geocoder

## Query completed in: 5.1 seconds

## New names:
## * 'lat' -> 'lat...4'
## * 'lat' -> 'lat...6'
```

```
tidy.geocoded
```

```
## # A tibble: 5 x 7
##   address                             id   lon lat...4 geoAddress lat...6  long
##   <chr>                            <int> <dbl>   <dbl> <chr>        <dbl> <dbl>
## 1 712 West LaSalle Ave, South Bend~     1 -86.3    41.7 712 w las~    41.7 -86.3
## 2 Hesburgh Library, University of ~     2 -86.2    41.7 284 hesbu~    NA    NA
## 3 Notre Dame Stadium                    3 -86.2    41.7 2010 moos~    41.7 -86.2
## 4 1047 Lincoln Way E South Bend, I~     4 -86.2    41.7 1047 e li~    NA    NA
## 5 227 W Jefferson Blvd South Bend,~     5 -86.3    41.7 county-ci~    41.7 -86.3
```