Regional Research Institute
West Virginia University

Technical Document Series

Using R and Google-API tools to estimate geographic features

Juan Tomas Sayago Gomez

RRI TechDoc 2017-01

Date submitted: October 31 of 2016
Date revised: March 28, 2017
Key words/Codes: Geocode, Elevation, distance matrix

Using R and Google-API tools to estimate geographic features

Juan Tomas Sayago Gomez

March 28, 2017

Abstract

This technical document is a guide for using Google APIs to find information for research purposes. First, I apply the code to find the elevation at a given set of coordinates for specific locations. Second, I apply the code to find the street distances between two or more sets of coordinates. All the codes and sample files are available in the zip file attached to this guide.

1 Introduction

The package *ggmap* created to use R and Google - APIs to visualize spatial data and models on static Google Maps includes tools for geocoding (or geolocation) and distance estimations Kahle and Wickham (2013). However, the function to compute map distance has a shortcoming that is important to note when using it for developing ; i.e., the function *mapdist* uses addresses as arguments, but some of the developing countries have issues with the addresses[1]. In this document, I provide a guide to use the Google - API to find elevation[2] and to compute the distance matrix using coordinates. This guide explains how use Google - APIs to compute elevation and map distances using coordinates as input. The code creates an interaction between R and Google Maps.

[1]The solution was to use coordinatesthat give an accurate, usuable location; however, *mapdist* converts the coordinates to addresses using the *geocode* function and then the problem recurs.

[2]To date, I could not find an R function that could provide this information.

2 Elevation

This section shows how to calculate elevation for locations or points in our data set. It uses an interaction between Google and R; the estimates come from an API created in Google and sets and estimate for each location's elevation.

The code must have the coordinates to measure the elevation and then it creates a command that inputs the coordinates to the API and exports the result, which consists of elevation and coordinates. The elevation is received as meters, which can be converted to Km, miles or feet. The data uses the coordinates of every point to find the elevation on each location; for example, the first location point has the following coordinates: "39.6400125,-80.0332251" and the elevation for such a location is 378.7 meters above sea level.

```
R> dest=read.csv("destinations.csv",header=T,sep=",")
R> names(dest)

[1] "Address" "Lat"     "Lng"

R> j=1
R>   googleelevataddress <- paste(dest$Lat[j],",",dest$Lng[j],collapse=" ", sep="")
R>    googleelevataddress

[1] "39.6400125,-80.0332251"

R>   dest$elevat <- vector(mode="character", length=length(dest$Lat))
R>  for (j in 1:length(dest$Lat)) {
+
+     googleelevataddress <- paste(dest$Lat[j],",",dest$Lng[j],collapse=" ", sep="")
+     googleelevataddress
+  googleurl <- url(paste("http://maps.googleapis.com/maps/api/elevation/json?
+locations=",googleelevataddress,"&sensor=false",sep=""))
+     googlell <- readLines(googleurl, warn=FALSE)
+  googlell
+  ot <- grep("elevation", googlell)
+  if(length(ot)>0) dest$elevat[j] <- googlell[ot] else  dest$elevat[i] <-"NA"
+  close(googleurl)
+
+
+  }
R> dest$elevation=gsub("[^.0-9]", "", dest$elevat)
R> dest$elevation
```

```
[1] "378.7772216796875" "259.3993225097656" "388.6174011230469"
```

3 Distance and Duration

This section demonstrates how to use the distance calculator with the interaction between Google Maps and R; the estimator has a code that reads origin and destinations and uses Google to find the distance between two locations. The document shows how to use addresses and coordinates as inputs for R and exports the results as data for R to use.

The commands in R create a link that connects to Google Maps API to do a request for the distance or distances and obtain the results in JSON output. The output has the following organization: the origin and destination places and then the distance(s). The requests enable you to use multiple destinations and origins, also to define different transportation methods such as driving, public transportation, walking and bycicling. The requests also include information about tolls.

The code first reads the origin and destination locations, which can be addresses or coordinates. The data is in the files *origins.csv* and *destinations.csv* and includes addresses and latitude and logitude for each address.

```
R> orig <- read.csv("origins.csv", header=TRUE, sep=",")
R> names(orig)

[1] "Address" "Lat"     "Lng"

R> summary(orig$Lat)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  32.87   37.82   39.55   37.90   39.63   39.65

R> summary(orig$Address)

4840 Tanger Outlet Boulevard, Tanger Outlets, North Charleston, SC 29418, USA
                                                                           1
                                    507 Elm Street, Morgantown, WV 26501, USA
                                                                           1
                  6051 University Town Centre Drive, Morgantown, WV 26501, USA
                                                                           1
                               800-898 Virginia Avenue, Fairmont, WV 26554, USA
                                                                           1

R> dest=read.csv("destinations.csv",header=T,sep=",")
R> names(dest)

[1] "Address" "Lat"     "Lng"
```

Next, the code creates a vector to save the information collected from Google; in this example we are going to create two vectors, one for duration time and one for distance. Then the code will run a loop to assign all the different origin points to run against the first destination (*j=1*) and save the distances and duration times. The following commands find the information that needs to be stored. The information that is reported in distances will be in meters and duration will be reported in seconds. The information can be transformed to miles, kilometers, hours, or minutes and used afterwards for analysis.

```
R> # create new columns and set the significant digits to 5;
R> j=1
R>   orig$distan <- vector(mode="character", length=length(orig$Lat))
R>  orig$durat <- vector(mode="character", length=length(orig$Lat))
R>  for (i in 1:length(orig$Lat)) {
+     googleorigaddress <- paste(orig$Lat[i],",",orig$Lng[i],collapse=" ", sep="")
+     googledestaddress <- paste(dest$Lat[j],",",dest$Lng[j],collapse=" ", sep="")
+  googleurl <- url(paste("http://maps.googleapis.com/maps/api/distancematrix/json?
+ origins=",googleorigaddress,"&destinations=",googledestaddress,"&mode=driving&
+ sensor=false&",sep=""))
+     googlell <- readLines(googleurl, warn=FALSE)
+  googlell
+  ot <- grep("distance", googlell)
+  if(length(ot)>0) orig$distan[i] <- googlell[ot+2] else  orig$distan[i] <-"NA"
+  otdur<- grep("duration", googlell)
+  if(length(otdur)>0) orig$durat[i] <- googlell[otdur+2] else orig$durat[i] <- "NA"
+        close(googleurl)
+  }
R> orig$distance=gsub("[^0-9]", "", orig$distan)
R> orig$duration=gsub("[^0-9]", "", orig$durat)
R> orig$distance

[1] "5848"   "11364"  "37753"  "930199"

R> orig$duration

[1] "553"   "1004"  "1588"  "31136"
```

The same loop procedure can be applied for longitudes, making the loop calculate all destinations into distances. For more information check the following link: https://developers.google.com/maps/docu

References

Kahle, D. and Wickham, H. (2013). ggmap: Spatial visualization with ggplot2. *The R Journal*, 5(1):144–161.