

Drawing beautiful maps in R

Mar Espadafor

17/11/2020

Before we start (I)

- ▶ Make sure you have the last R version (4.03), you can update it by downloading this version here:

<https://cran.r-project.org>

- ▶ Before we can start geocoding, we need to obtain an API key from Google. Go to the registration page, and follow the instructions (select all mapping options). The geocoding API is a free service, but up to a limit. That's why you nevertheless need to associate a card with the account (sorry about that!).

<https://cloud.google.com/maps-platform/#get-started>

Before we start (II)

- ▶ Now, set your working directory to the folder where all the materials, use your google key and the packages that we are going to use. You can also follow lines 1-18 in “exercises_datapoints.R”
- ▶ Read about different coordinate systems, map projections and transformations:

<http://resources.esri.com/help/9.3/arcgisengine/dotnet/89b720a5-7339-44b0-8b58-0f5bf2843393.htm>

Intro: Types of spatial data in a nutshell:

Using coordinates (longitude and latitude) we can create the following types of data:

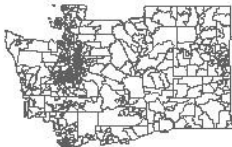
Spatial data types



cities • *Points*



railroads • *Lines*



zipcodes • *Polygons*

Intro: Feature types and their representations

Feature Type

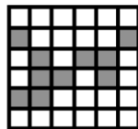
Vector Model

Raster Model

Point Feature



Line Feature



Area Feature

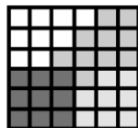
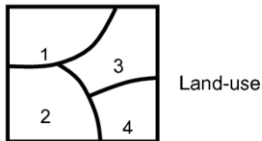
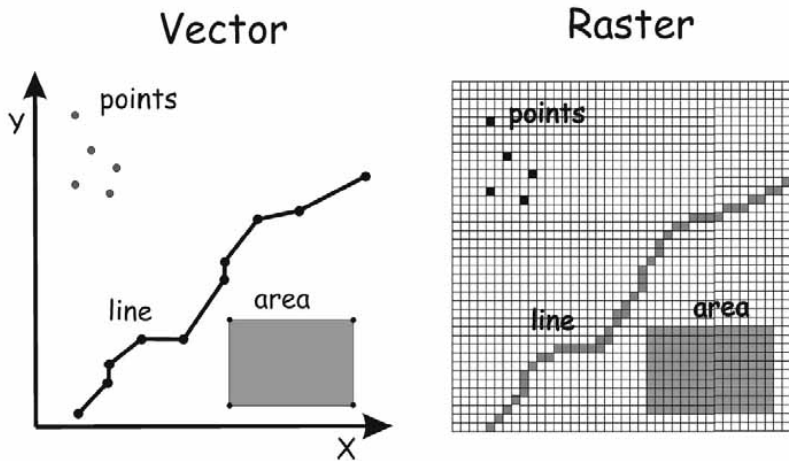


Figure 2: s

Intro: Differences between vector and raster model:



Vector and raster data models.

Grabbing a background map

There are two steps to adding a map to a ggplot2 plot with ggmap:

- ▶ Download a map using `get_map()`
- ▶ Display the map using `ggmap()`

Download the map with `Get_map()`

`Get_map()` has a number of arguments that control what kind of map we can get:

- ▶ **Location:** provides a longitude and latitude pair of coordinates where you want the map to be centered.
- ▶ **Zoom:** controls how far the mapped is zoomed in.
- ▶ **Scale:** controls resolution of the downloaded maps (we are setting to 1 so it takes fewer time to download)
- ▶ **Maptype:** the type of map we are getting

Example I: Getting NYC map

STEP 1: Geolocating NYC

#Two ways of geolocating NYC:

```
nyc <- c(lon = -74.0059, lat = 40.7128) #google lat and lon  
nyc<-geocode("New York City") #naming the place  
head(nyc)
```

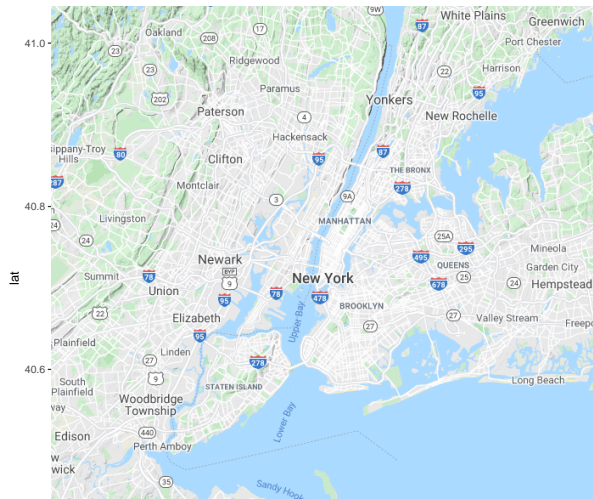
```
## # A tibble: 1 x 2  
##   lon   lat  
##   <dbl> <dbl>  
## 1 -74.0  40.7
```

```
nyc_map <- get_map(location = nyc, zoom = 10, scale = 1)
```

Example I: Getting NYC map

STEP 2: Displaying the map Displaying the map is then as simple as calling `ggmap()` with your downloaded map as the only argument

```
ggmap(nyc_map)
```



Exercise I: Map of Madrid

You can find exercise I is from lines 21 to 33 in “exercises_datapoints.R”. The tasks are the following:

- ▶ Geolocate Madrid
- ▶ Get map of Madrid at zoom 5 and then 12
- ▶ BONUS: Can you think of a quicker way around this?
- ▶ Plot map calling ggmap

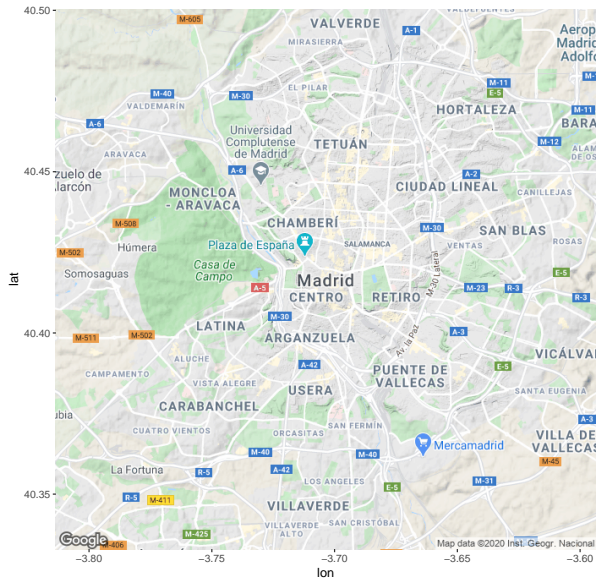
Exercise I: Solution

```
#EXERCISE I: Geolocate Madrid
madrid<-geocode("Madrid")
# Get map at zoom level 5: fmap
mmap<- get_map(madrid, zoom=5 , scale = 1)
#ggmap(mmap)
#get a map at zoom level 12 and plot it
fmap12<-get_map(madrid, zoom=12, scale=1)
#ggmap(fmap12)

#shorter way around it:
madrid_map<-get_map(geocode("Madrid"),zoom= 12, scale=1)
```

We did our first map!!

```
ggmap(madrid_map)
```



Geolocating I

We did geolocate Madrid doing:

```
madrid<-geocode("Madrid")
```

But we could geolocate anything we want, as long as we have an address for it...let's try a street:

```
myhouse<-geocode("Via Cino da Pistoia, 26, Firenze")  
head(myhouse)
```

```
## # A tibble: 1 x 2  
##   lon   lat  
##   <dbl> <dbl>  
## 1  11.3  43.8
```

Geolocating II:

In Exercise II (lines 35-60) we are going to be geocoding schools in Madrid. The data “schools10.dta” has information of secondary schools in Madrid. The variable “address2” identifies the location of the schools.

For simplicity we are going to be using a sample of 10 (so it goes fast)

Let's try out!

Geolocating III:

- ▶ First load “schools10.dta”
- ▶ Run the two first ways of getting the locations
- ▶ Can you think of an easier way of doing it?

HINT: How do you create a variable within the tidyverse package?

Geolocating IV: Solutions

```
## Geolocating schools in Madrid  
#wrong way  
schools10_f<- geocode(schools10$address2)  
schools10$fake<-geocode(schools10$address2)  
# the way  
schools10_locations<- mutate_geocode(schools10, address2)
```

Layering data on top (I)

This means that we might want to add a layer of something other than the map, for this we need to specify both the mapping and data arguments to the geom.

- ▶ How might this look like in R? Think of a ggplot. . .

```
#   ggplot(data, aes(x,y)) +  
#   geom_point()
```

Layering data on top (II): Swapping ggplot for ggmap

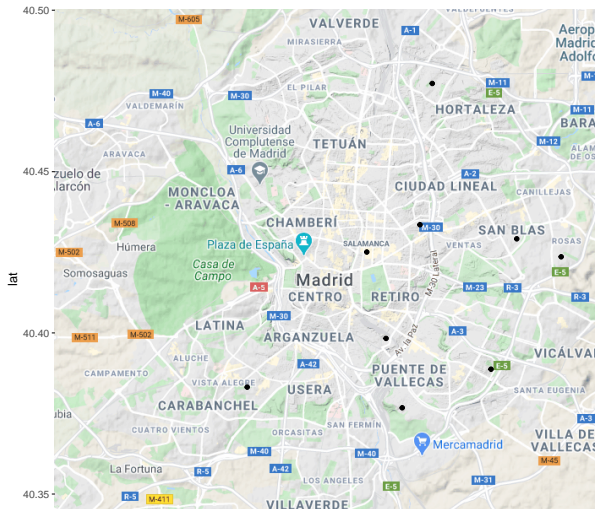
EXERCISE IIIa (lines 48-61)

- ▶ Instead of `x` and `y` we have now longitude and latitude points.
- ▶ First we swap from `ggplot` to `ggmap` to have as our first layer our map
- ▶ Then we keep using `geom_point`, specifying `lon` and `lat` in the `aes()` argument and where is the data coming from in the `data` argument.
- ▶ Look at the warning message: what do you think it happened?

Layering data on top (III): Exercise IIIa solution

#EXERCISE IIIa: Layer schools into Madrid's map

```
ggmap(madrid_map)+  
  geom_point(aes(lon,lat), data=schools10_locations)
```



Layering data on top (IV): what about aesthetics?

The default Google map downloaded by `get_map()` is useful when you need major roads, basic terrain, and places of interest, but visually it can be a little busy.

You want your map to add to your data, not distract from it, so it can be useful to have other “quieter” options.

Take a look at:

<http://maps.stamen.com/#toner/12/37.7706/-122.3782>

Layering data on top (V): changing aesthetics

- ▶ Go and try to do Exercise IIIb (lines 61-72)
- ▶ Change the type of map using the "maptype" argument

Let's go!

Layering data on top (V): changing aesthetics

```
madrid_map<-get_map(geocode("Madrid"),zoom= 12,  
                    scale=1, maptype = "toner-lite")  
ggmap(madrid_map)+  
  geom_point(aes(lon,lat), data=schools10_locations)
```



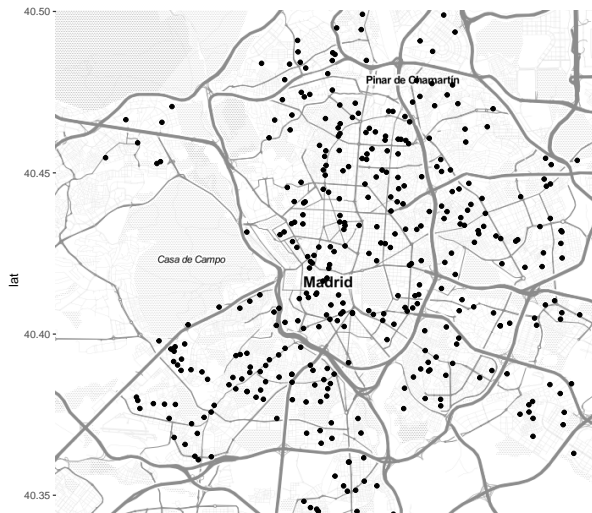
Layering data on top (VI): layering multiple things

In Exercise IV (lines 74-96) we are going to be layering both schools and betting houses in Madrid city. This data comes from a paper Sergi Marinez and I are working on, so please, don't share it!

- ▶ Load “schools.dta” and “bets.dta”
- ▶ These datasets contain coordinates (lon,lat) for both schools and betting houses
- ▶ Make a plot using both of them
- ▶ Use the argument color to differentiate them

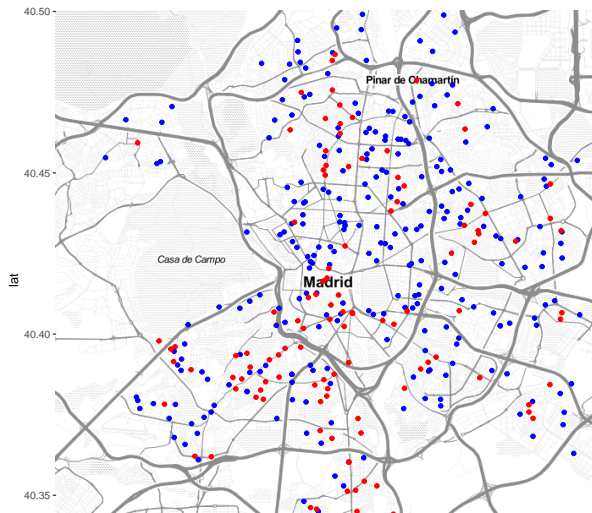
Exercise IVa solution

```
ggmap(madrid_map)+  
  geom_point(aes(lon,lat), data=schools)+  
  geom_point(aes(lon,lat), data=bets)
```



Exercise IVb solution

```
ggmap(madrid_map)+  
  geom_point(aes(lon,lat), data=schools, color="blue")+  
  geom_point(aes(lon,lat), data=bets, color="red")
```



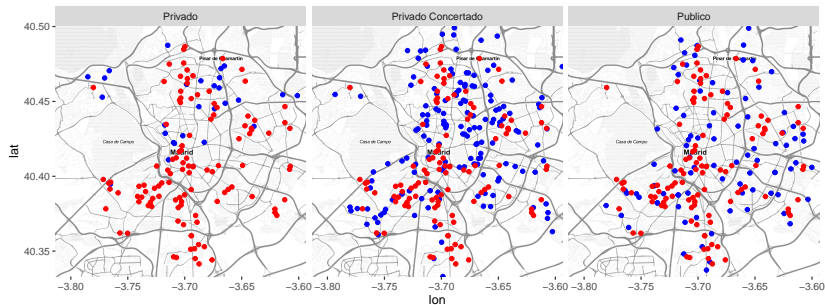
How could we do multiple maps?

Maybe we are interested in looking at differences by types of schools. To do so, we could use `face_wrap`.

- ▶ Type 2 is a variable that indicates the type of school (private, public or charter)
- ▶ Using `face_wrap`, create a differentiated map for each school type

Example using different types of schools:

```
ggmap(madrid_map)+  
  geom_point(aes(lon,lat), data=schools, color="blue")+  
  facet_wrap(~type2)+  
  geom_point(aes(lon,lat), data=bets, color="red")
```



BONUS: argument size and tmaptools

- ▶ With the argument size we can make our data points bigger or smaller, according to another specified variable.
- ▶ tmaptools is similar to geocode, but for free

```
#EXTRA BONUS: alternative way of geolocating using tmaptools  
#install.packages("tmaptools")  
#library(tmaptools)  
streets <- "Adramuttiou, Thessaloniki, Greece"  
#inspect streets  
streets_tmaptools <- geocode_OSM(paste("street", streets),  
                                details = TRUE, as.data.frame = FALSE)  
#head(street_tmaptools)
```