

# Manipulação de dados em R

*Leonardo Sangali Barone*

*March 27, 2017*

## Mapas no R

Data frames foram, até este momento do curso, o protagonista de nossa análise de dados. Vimos como manipular dados que estão sempre no mesmo formato

### Informação espacial armazenada em data frames - pontos no google maps

Vamos começar a trabalhar com mapas a partir de um exemplo que, veremos, utilizará as ferramentas que aprendemos até então para produzir nossos primeiros mapas. Para tanto, vamos utilizar o cadastro de escolas que a Prefeitura Municipal de São Paulo disponibiliza aqui.

Nossa primeira tarefa é baixar os dados e faremos isso de forma inteligente e sem “cliques”. A partir do url do arquivo do cadastro, que guardaremos no objeto “url\_cadatros\_escolas”, faremos o download do arquivo e guardaremos o arquivo .csv baixado como o nome “temp.csv”:

```
url_cadatros_escolas <- "http://dados.prefeitura.sp.gov.br/dataset/8da55b0e-b385-4b54-9296-d0000014ddd5"
download.file(url_cadatros_escolas, "temp.csv")
```

Veja que baixar o arquivo diretamente no R é preferível ao processo manual, pois podemos rapidamente reproduzir o processo, além de documentá-lo. Vamos abrir o arquivo:

```
library(readr)
escolas <- read_delim("temp.csv", delim = ";")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   CODINEP = col_integer(),
##   EH = col_double(),
##   LATITUDE = col_integer(),
##   LONGITUDE = col_integer()
## )
## See spec(...) for full column specifications.
```

Explore o arquivo com o comando *glimpse*:

```
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
## 
##   filter, lag

## The following objects are masked from 'package:base':
## 
##   intersect, setdiff, setequal, union
```

```
glimpse(escolas)
```

```
## Observations: 5,324
## Variables: 51
## $ DRE <chr> "G", "FO", "MP", "BT", "PJ", "BT", "FO", "JT", "JT...
## $ CODESC <chr> "000086", "000094", "000108", "000191", "000205", ...
## $ TIPOESC <chr> "EMEI", "EMEI", "EMEF", "EMEF", "EMEBS", "EMEI", "...
## $ NOMESC <chr> "PAULO CAMILHIER FLORENCANO", "VICENTE PAULO DA SI...
## $ DIRETORIA <chr> "GUAIANASES", "FREGUESIA/BRASILANDIA", "SAO MIGUEL...
## $ SUBPREF <chr> "GUAIANASES", "CASA VERDE/CACHOEIRINHA", "SAO MIGU...
## $ CEU <chr> NA, NA...
## $ ENDERECO <chr> "RUA FELICIANO DE MENDON\ xc7A", "RUA DOUTOR FLEURY...
## $ NUMERO <chr> "502", "295", "159", "140", "206", "90", "51", "34...
## $ BAIRRO <chr> "JARDIM S\ xc30 PAULO(ZONA LESTE)", "VILA SANTA MAR...
## $ CEP <chr> "08460365", "02563010", "08090290", "05742100", "0...
## $ TEL1 <chr> "25578348", "39813227", "25865294", "58450121", "3...
## $ TEL2 <chr> "25571947", NA, "25811484", NA, "39067229", NA, NA...
## $ FAX <chr> "25571947", NA, NA, "58450121", NA, "58425113", "3...
## $ SITUACAO <chr> "Ativa", "Ativa", "Ativa", "Ativa", "Ativa", "Ativ...
## $ CODDIST <chr> "31", "50", "44", "94", "63", "94", "29", "89", "8...
## $ DISTRITO <chr> "GUAIANASES", "LIMAO", "JARDIM HELENA", "VILA SONI...
## $ SETOR <chr> "3103", "5002", "4402", "9404", "6301", "9404", "2...
## $ CODINEP <int> 35098711, 35098361, 35098760, 35098462, 35079029, ...
## $ CODCIE <chr> "098711", "098361", "098760", "098462", "079029", ...
## $ EH <dbl> 1.610781e+14, 1.610702e+14, 1.610771e+14, 1.610792...
## $ DT_CRIACAO <chr> "13/06/1988", "04/07/1988", "05/07/1988", "27/05/1...
## $ ATO_CRIACAO <chr> "26.134", "26.314", "26.312", "26.003", "26.229", ...
## $ DOM_CRIACAO <chr> "13/06/1988", "04/07/1988", "05/07/1988", "27/05/1...
## $ DT_INI_FUNC <chr> "22/09/1988", "01/08/1988", "01/09/1988", "02/10/1...
## $ DT_AUTORIZA <chr> "16/03/1991", "16/03/1991", "13/03/2001", "16/03/1...
## $ NOME_ANT <chr> NA, NA, "VILA NITRO OPERARIA", NA, NA, NA, "INSTAL...
## $ T2D3D <chr> "2D", "2D", "2D", "2D", "2D", "2D", "2D", "2...
## $ T2D3D15 <chr> "2D", "2D", "2D", "2D", "2D", "2D", "2D", "2...
## $ T2D3D14 <chr> "2D", "2D", "2D", "2D", "2D", "2D", "2D", "2...
## $ T2D3D13 <chr> "2D", "2D", "2D", "2D", "2D", "2D", "2D", "2...
## $ T2D3D12 <chr> "2D", "2D", "2D", "2D", "2D", "2D", "2D", "2...
## $ T2D3D11 <chr> "2D", "2D", "2D", "2D", "2D", "2D", "2D", "2...
## $ T2D3D10 <chr> "2D", "2D", "2D", "2D", "2D", "2D", "2D", "2...
## $ T2D3D09 <chr> "2D", "2D", "2D", "2D", "2D", "2D", "2D", "2...
## $ T2D3D08 <chr> "3D", "3D", "2D", "2D", "2D", "3D", "2D", "2...
## $ T2D3D07 <chr> "3D", "3D", "3D", "2D", "2D", "3D", "2D", "3...
## $ DTURNOS <chr> "2D", "3D", "3D", "2D", "2D", "3D", "3D", "3...
## $ DTURNOS15 <chr> "MT", "MT", "MT", "MT", "MT", "MT", "MT", "MT...
## $ DTURNOS14 <chr> "MT", "MT", "MT", "MT", "MT", "MT", "MT", "MT...
## $ DTURNOS13 <chr> "MT", "MT", "MT", "MT", "MT", "MT", "MT", "MT...
## $ DTURNOS12 <chr> "MT", "MT", "MT", "MT", "MTN", "MT", "MT", "MT...
## $ DTURNOS11 <chr> "MT", "MT", "MT", "MT", "MTN", "MT", "MT", "MT...
## $ DTURNOS10 <chr> "MT", "MT", "MTN", "MT", "MTN", "MT", "MT", "MT...
## $ DTURNOS09 <chr> "MT", "MT", "MTN", "MT", "MTN", "MT", "MT", "MT...
## $ DTURNOS08 <chr> "MT", "MT", "MTN", "MT", "MTN", "MT", "MT", "MT...
## $ DTURNOS07 <chr> "MIV", "MIV", "MTN", "MT", "MTN", "MIV", "MT", "MT...
## $ LATITUDE <int> -23553905, -23489728, -23478312, -23612237, -23486...
## $ LONGITUDE <int> -46398452, -46670198, -46427344, -46749888, -46733...
## $ REDE <chr> "DIR", "DIR", "DIR", "DIR", "DIR", "DIR", "DIR", "...
```

```
## $ DATABASE      <chr> "28/02/2017", "28/02/2017", "28/02/2017", "28/02/2017",
```

Não há nada de extraordinário no arquivo, que se assemelha aos que vimos até então. Há, porém, uma dupla de variáveis que nos permite trabalhar “geograficamente” com o dado: LATITUDE e LONGITUDE. “Lat e Long” são a informação fundamental de um dos sistemas de coordenadas (coordinate reference system, CRS) mais utilizados para localização de objetos na superfície da terra.

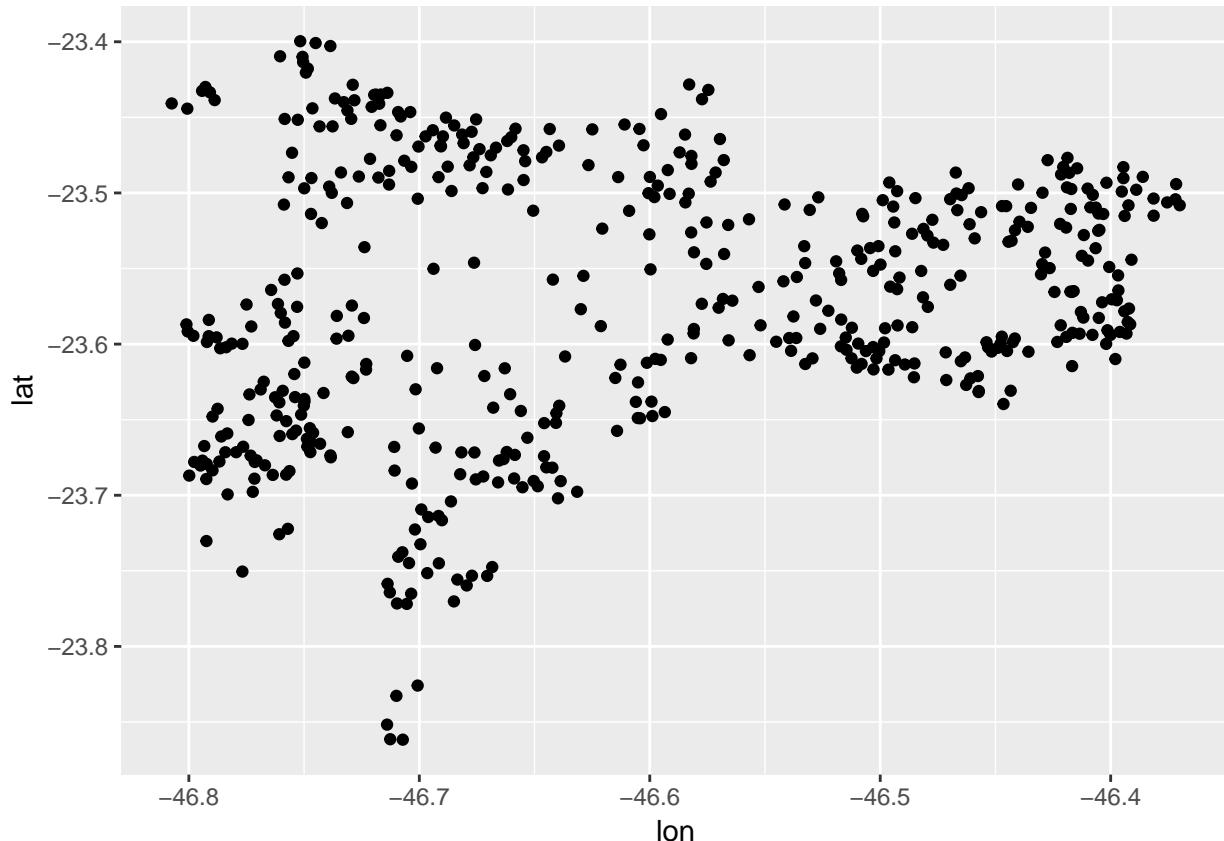
Por uma razão desconhecida, a informação fornecida pela PMSP está em formato diferente do convencional. Latitudes são representadas por números entre -90 e 90, com 8 casas decimais, e Longitudes por números entre -180 e 180, também com 8 casas decimais. Em nosso par de variáveis, o separador de decimal está omitido e por esta razão faremos um pequena modificação na variável. Aproveitaremos também para renomear algumas variáveis de nosso interesse – como tipo da escola (CEI, EMEI, EMEF, CEU, etc) e o ano de início do funcionamento – e selecionaremos apenas as linhas referentes a EMEF (Escolas Municipal de Ensino Fundamental):

```
emef <- escolas  %>%
  rename(lat = LATITUDE, lon = LONGITUDE, tipo = TIPOESC) %>%
  mutate(lat = lat / 1000000,
        lon = lon / 1000000,
        ano = as.numeric(substr(DT_INI_FUNC, 7, 10))) %>%
  filter(tipo == "EMEF")
```

Pronto! Temos agora uma informação geográfica das EMEFs e uma variável de interesse – ano – que utilizaremos para investigar a expansão da rede.

Vamos construir um primeiro mapa, usando a função que conhecemos – ggplot – a partir das informações de latitude e longitude das escolas:

```
library(ggplot2)
ggplot(aes(lon, lat), data = emef) +
  geom_point()
```



Veja que podemos “imaginar” o formato da cidade de São Paulo com os pontos, mas o mapa não é propriamente um mapa. Falta uma “camada” básica, sobre a qual os pontos serão desenhados.

Vamos utilizar o pacote *ggmap*, que é um pacote para visualização de dados espaciais com o pacote *ggplot2* para obter tal “camada”. Com a função *get\_map*, faremos o download de um mapa que servirá de base para os pontos das EMEFs.

A função *get\_map* requer como argumento principal um par de coordenadas a partir do qual o mapa será centralizado. No nosso exemplo vamos utilizar as coordenadas da Praça da Sé, que serão armazenadas em um vetor. É fácil obter coordenadas de um local a partir de serviços de localização na internet.

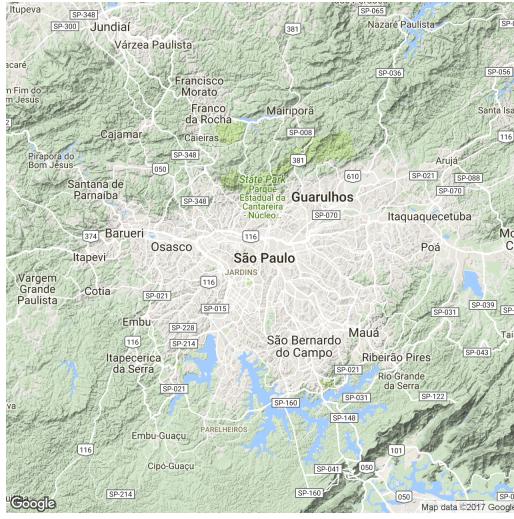
```
library(ggmap)
se <- c(lon = -46.6362714, lat = -23.5500806)
```

Com as coordenadas da Praça de Sé, vamos obter um mapa de São Paulo:

```
map_sp <- get_map(se)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=10&size=600x300
```

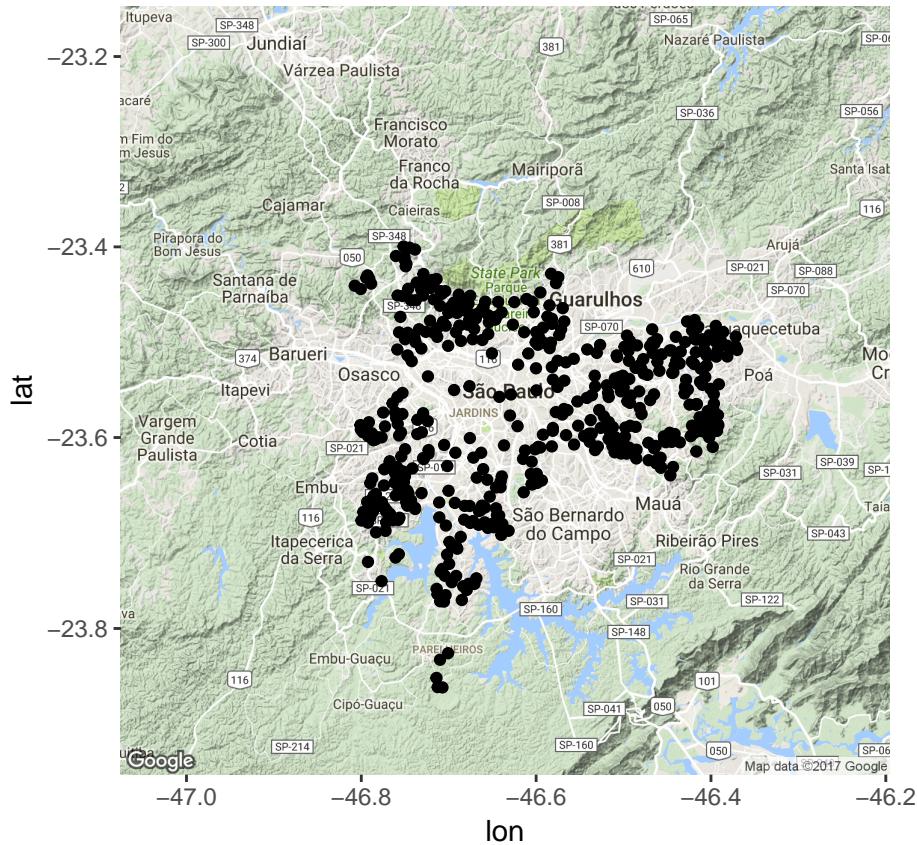
```
plot(map_sp)
```



Por padrão, `get_map` retorna um mapa de “terreno” e utiliza a API da Google, com zoom e escala automáticos.

Antes de alterar estes argumentos, vamos utilizar uma função “irmã” à `ggplot`, `ggmap`, para juntar o mapa de São Paulo com os pontos das escolas:

```
ggmap(map_sp) +
  geom_point(aes(lon, lat), data = emef)
```



Feio ainda, porém bastante mais informativo. Veja que combinamos duas fontes de dados: o cadastro da PMSP e um mapa obtido na API da Google.

Vamos obter mapas mais interessantes com `get_map`. Em primeiro lugar, podemos variar os tipos de mapas

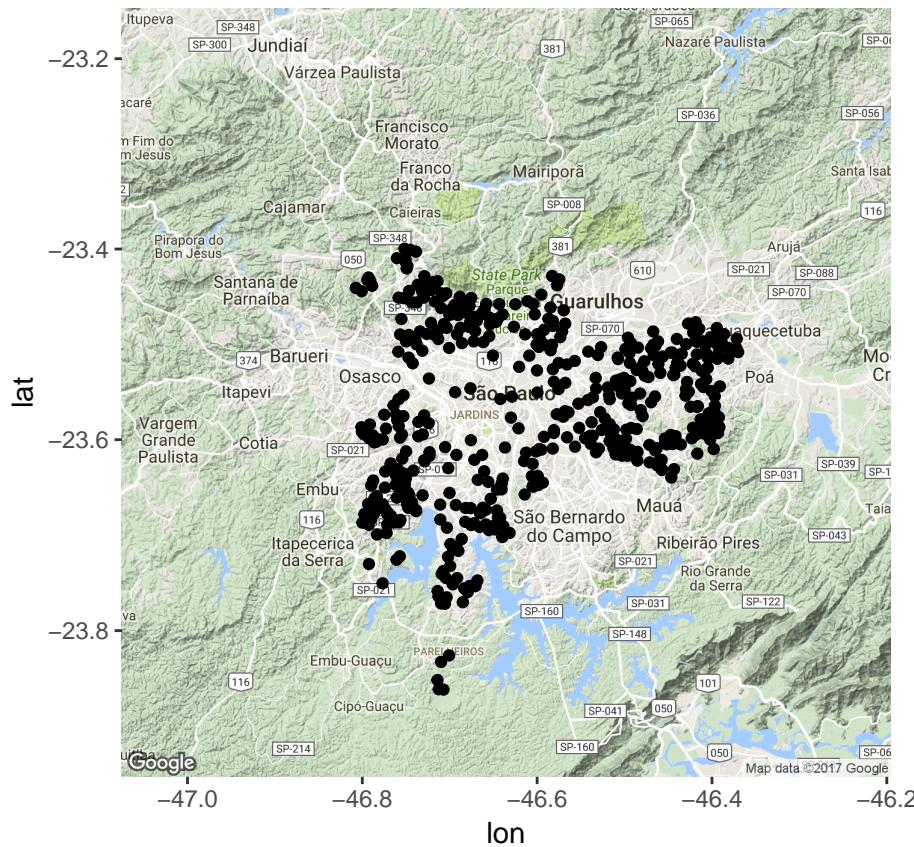
(argumento “`maptyle`”) e obter qualquer um dentre as seguintes opções: “`terrain`”, “`terrain-background`”, “`satellite`”, “`roadmap`”, “`hybrid`”, “`toner`”, “`watercolor`”, “`terrain-labels`”, “`terrain-lines`”, “`toner-2010`”, “`toner-2011`”, “`toner-background`”, “`toner-hybrid`”, “`toner-labels`”, “`toner-lines`” e “`toner-lite`”.

Também podemos variar a API – argumento “`source`” – na qual o mapa será obtido: Google (“`google`”), Open Street Maps (“`osm`”) ou Stamen (“`stamen`”). Nem todas as “`source`” e “`maptypes`” podem ser combinados e os “`toner`” provêm da API Stamen.

Vejamos alguns exemplos (note o uso do operador “`pipe`”) abaixo:

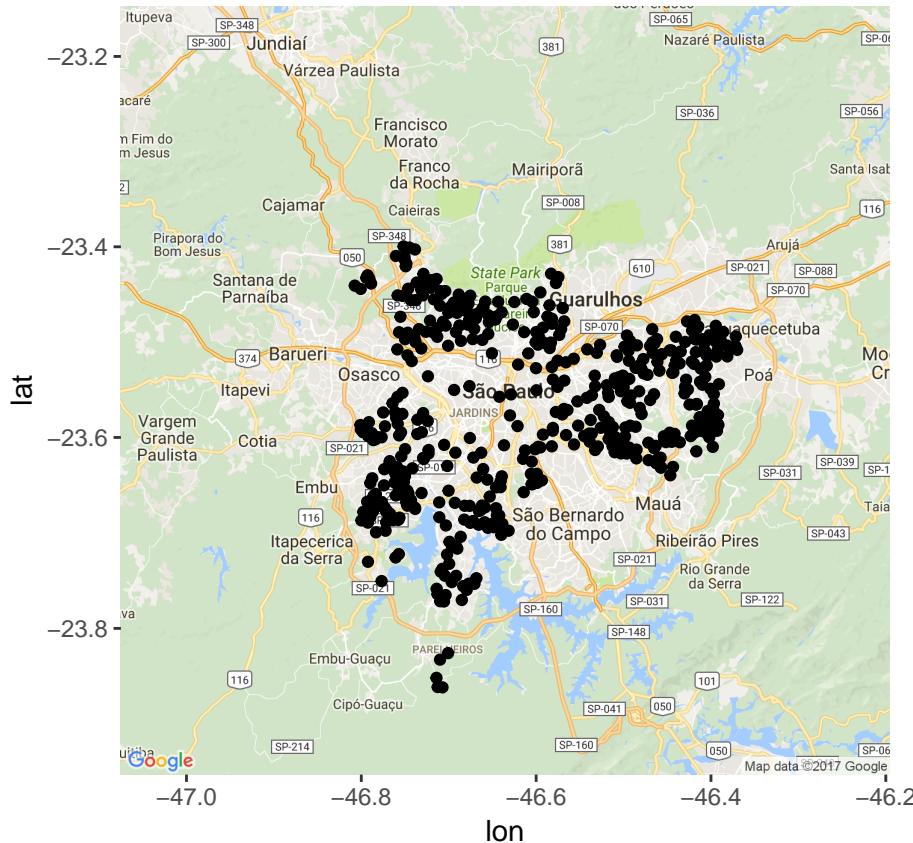
```
# Terrain Google
get_map(se, source = "google", maptype = "terrain") %>%
  ggmap() +
  geom_point(aes(lon, lat), data = emef)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=10&size=640x480&maptype=terrain
```



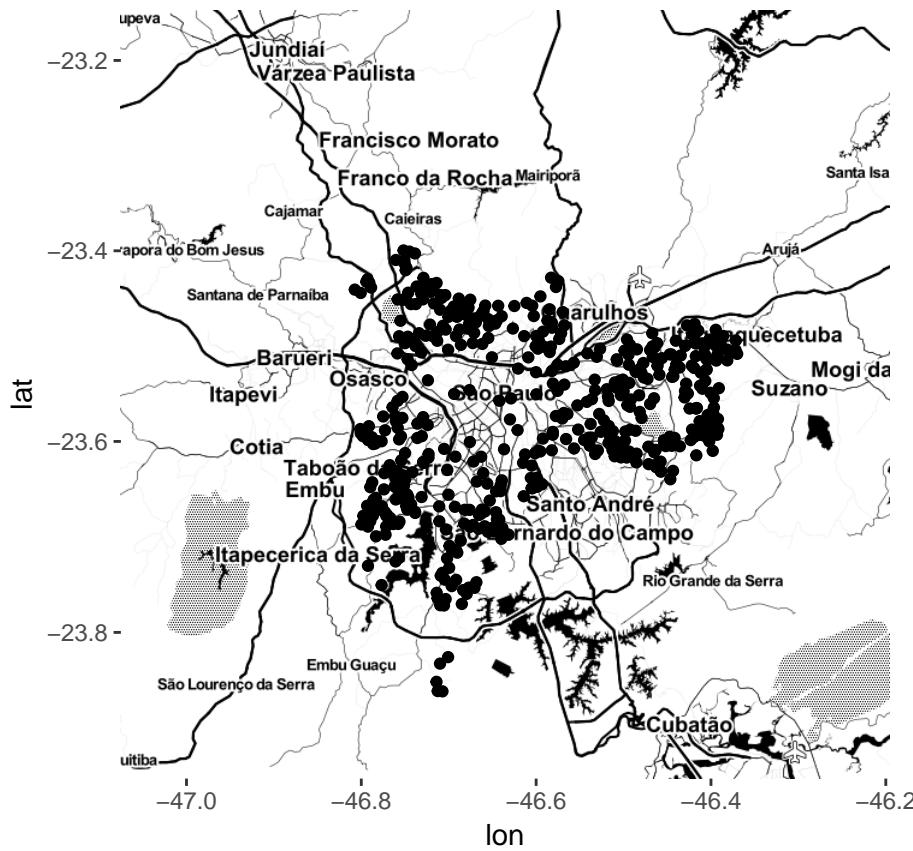
```
# Roadmap Google
get_map(se, source = "google", maptype = "roadmap") %>%
  ggmap() +
  geom_point(aes(lon, lat), data = emef)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=10&size=640x480&maptype=roadmap
```



```
# Toner Stamen
get_map(se, source = "stamen", maptype = "toner") %>%
  ggmap() +
  geom_point(aes(lon, lat), data = emef)
```

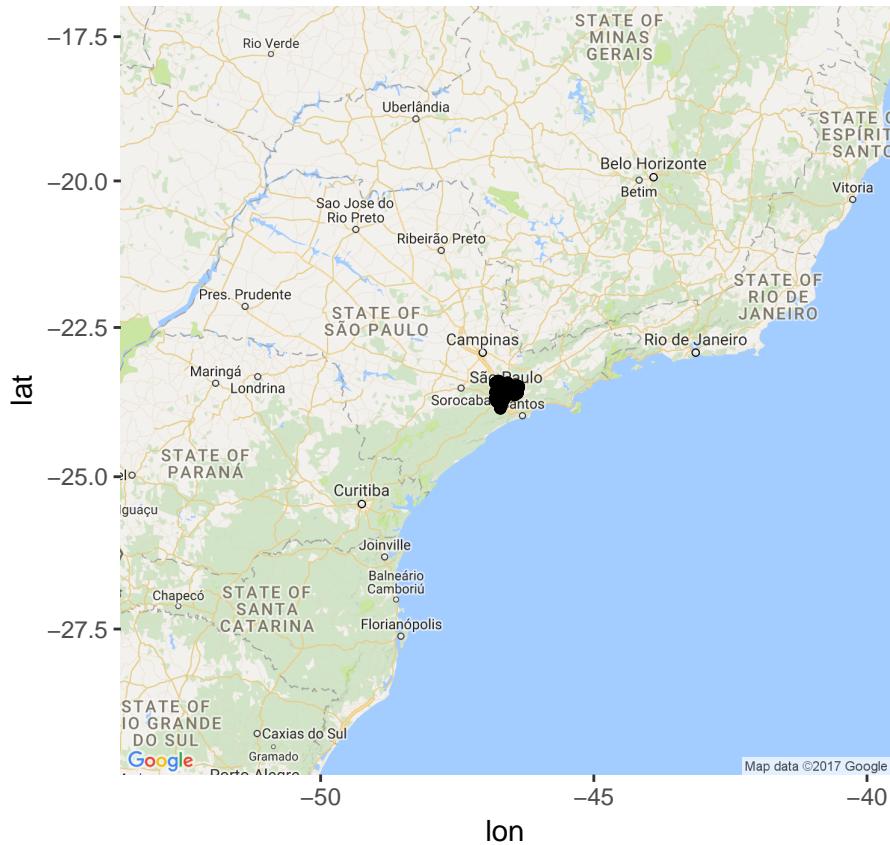
```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=10&size=640x480&maptype=toner
## Map from URL : http://tile.stamen.com/toner/10/378/579.png
## Map from URL : http://tile.stamen.com/toner/10/379/579.png
## Map from URL : http://tile.stamen.com/toner/10/380/579.png
## Map from URL : http://tile.stamen.com/toner/10/378/580.png
## Map from URL : http://tile.stamen.com/toner/10/379/580.png
## Map from URL : http://tile.stamen.com/toner/10/380/580.png
## Map from URL : http://tile.stamen.com/toner/10/378/581.png
## Map from URL : http://tile.stamen.com/toner/10/379/581.png
## Map from URL : http://tile.stamen.com/toner/10/380/581.png
## Map from URL : http://tile.stamen.com/toner/10/378/582.png
## Map from URL : http://tile.stamen.com/toner/10/379/582.png
## Map from URL : http://tile.stamen.com/toner/10/380/582.png
```



Vamos alterar o zoom dos mapas e obter apenas mapas e ver o efeito:

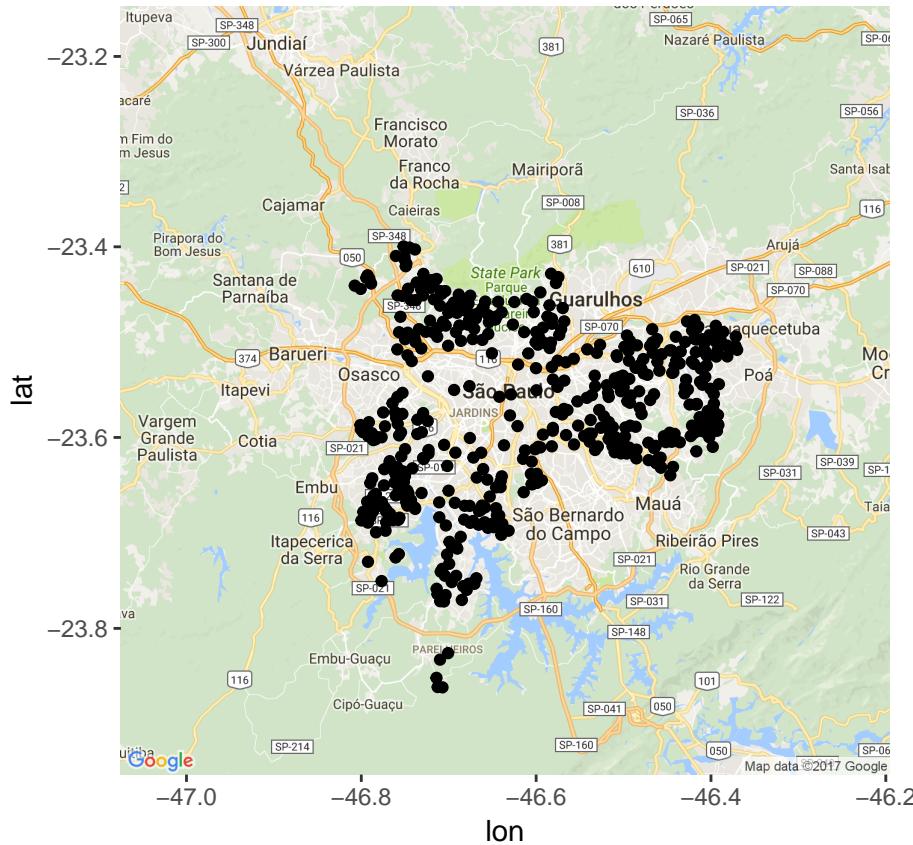
```
get_map(se, source = "google", maptype = "roadmap", zoom = 6) %>%
  ggmap() +
  geom_point(aes(lon, lat), data = emef)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=6&siz
```



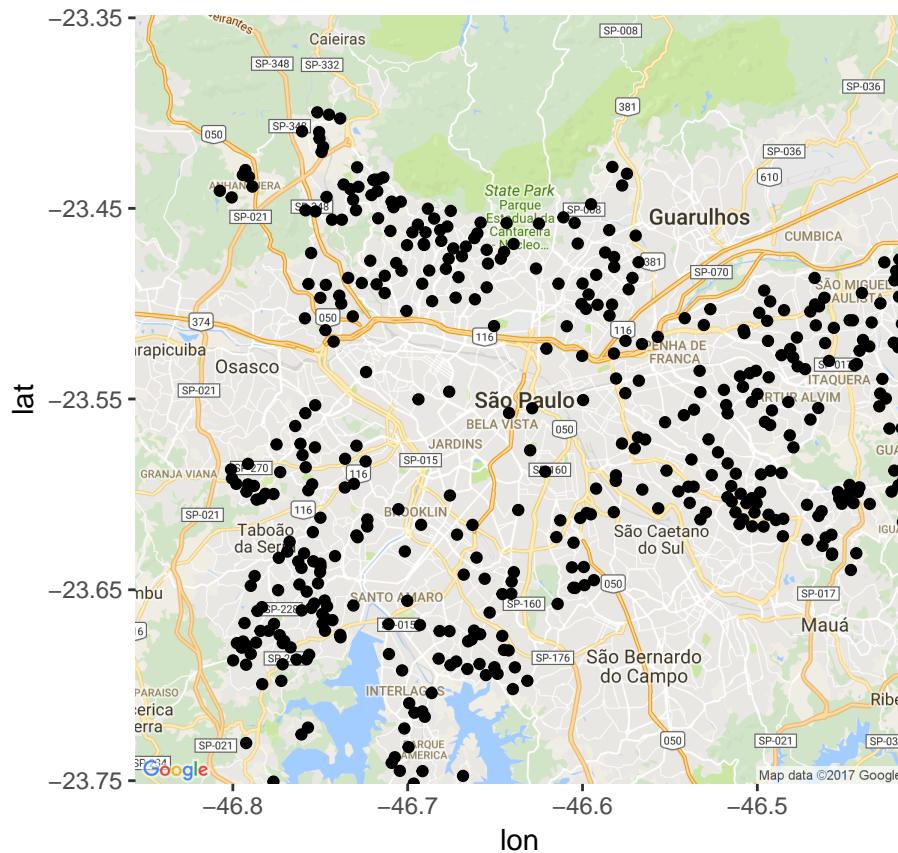
```
get_map(se, source = "google", maptype = "roadmap", zoom = 10) %>%
  ggmap() +
  geom_point(aes(lon, lat), data = emef)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=10&size=600x400
```



```
get_map(se, source = "google", maptype = "roadmap", zoom = 11) %>%
  ggmap() +
  geom_point(aes(lon, lat), data = emef)

## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=11&size=600x400
## Warning: Removed 65 rows containing missing values (geom_point).
```

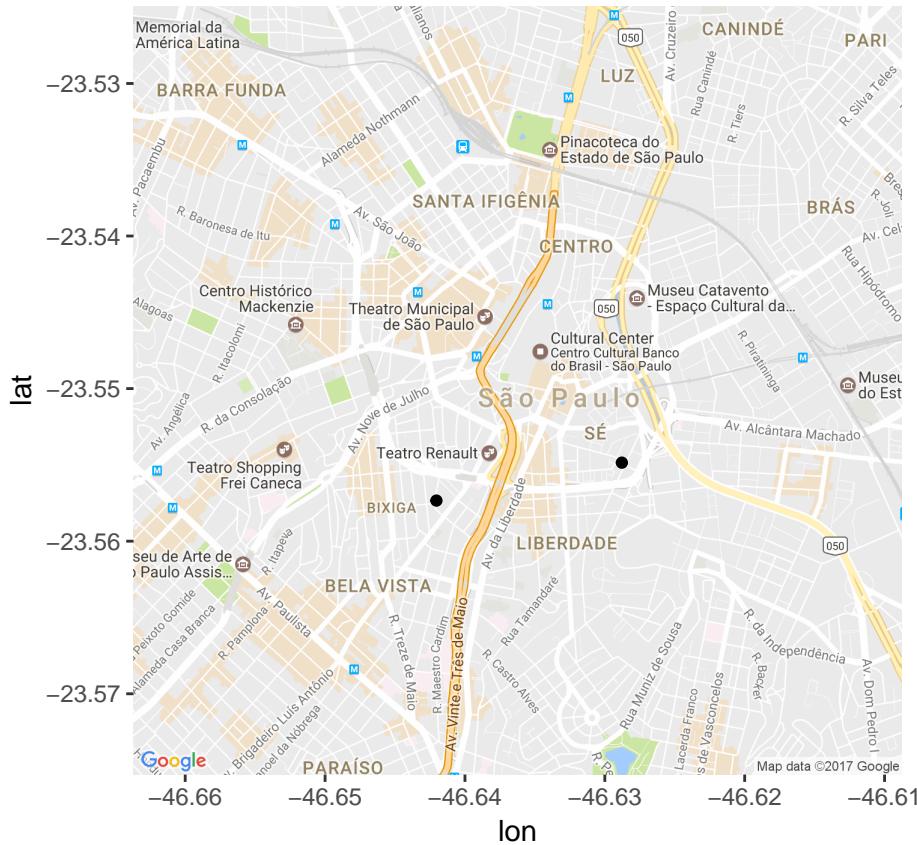


```

get_map(se, source = "google", maptype = "roadmap", zoom = 14) %>%
  ggmap() +
  geom_point(aes(lon, lat), data = emef)

## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=14&size=640x480
## Warning: Removed 499 rows containing missing values (geom_point).

```

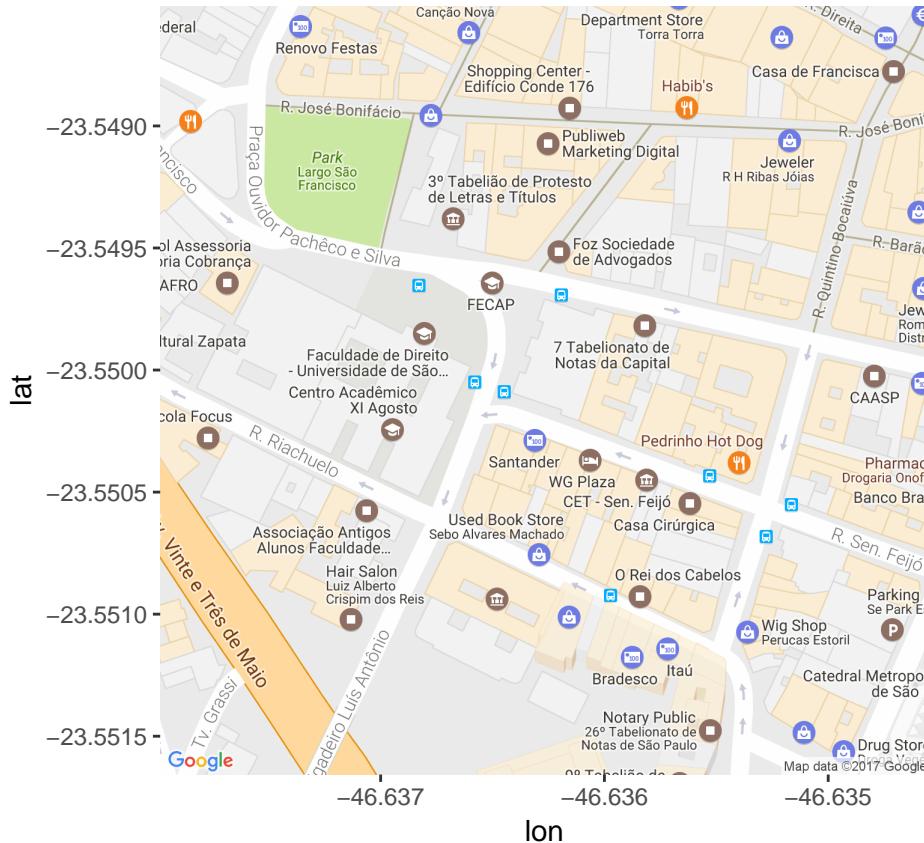


```

get_map(se, source = "google", maptype = "roadmap", zoom = 18) %>%
  ggmap() +
  geom_point(aes(lon, lat), data = emef)

## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=18&size=600x400
## Warning: Removed 501 rows containing missing values (geom_point).

```



Note que, quando estão fora da base, os pontos são descartados da visualização e obtemos um “warning”.

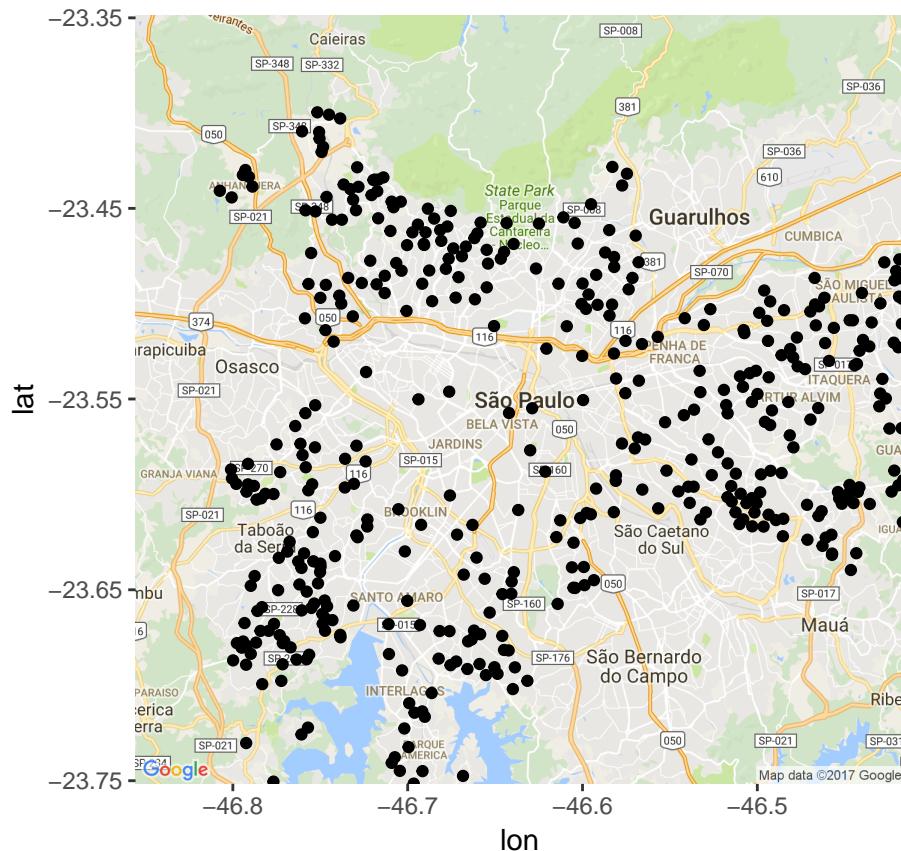
Convém, por conta da gramática da família de funções *ggplot*, definir uma camada básica na função *ggmap*, que, no nosso caso, são os pontos que estão em análise (e não a camada de mapa que (sic) dá base à visualização):

```
map_sp <- get_map(se, source = "google", maptype = "roadmap", zoom = 11)

## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=11&size=640x480&maptype=roadmap&key=AIzaSyDwzXGKUyfjJYQZMgkqfIwvzJLcJLJLJU

ggmap(map_sp,
      base_layer = ggplot(aes(lon, lat), data = emef)) +
      geom_point()

## Warning: Removed 65 rows containing missing values (geom_point).
```



Finalmente, vamos usar uma informação sobre as escolas para diferenciar os pontos. Os dados do cadastro não trazem informações muito interessantes sobre a escola e por isso utilizaremos o ano de criação. Em nosso desafio seguinte trabalharemos com informações sobre a escola que vêm de outra base do mesmo portal da prefeitura.

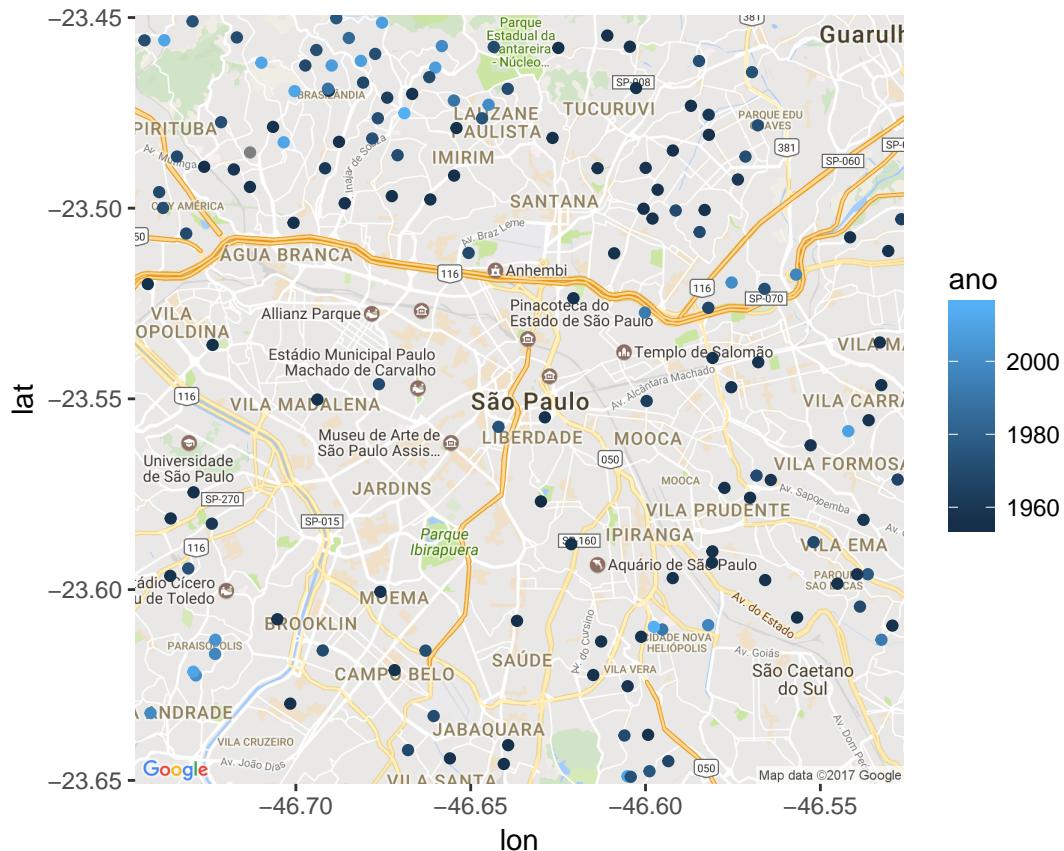
Introduziremos em nosso mapa uma escala de cores para diferenciar as EMEFs por ano de criação:

Nota: se você estiver com dificuldade em relação à gramática do `ggplot`, é sinal que pulou as últimas duas semanas e não leu os capítulos recomendados. Volte às aulas 6 e 7.

```
map_sp <- get_map(se, source = "google", maptype = "roadmap", zoom = 12)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=-23.550081,-46.636271&zoom=12&size=600x300&maptype=roadmap&key=AIzaSyCwDyfJLWVQHgkXGKjPQZBzvIYUOOGdIw
ggmap(map_sp,
      base_layer = ggplot(aes(lon, lat, color = ano), data = emef)) +
      geom_point()
```

`## Warning: Removed 348 rows containing missing values (geom_point).`



Como era de se esperar, os pontos mais claros na cidade estão nos extremos e as EMEFs mais antigas, e geral, nos bairros que consolidaram mais cedo no processo de urbanização.

## Procurando Lat e Long com o pacote ggmap

E quando temos apenas o endereço dos locais que queremos encontrar? Bem, há alternativas oferecidas pelo próprio pacote *ggmap*.

Vamos agora criar um novo data frame a partir dos dados do cadastro da PMSP que contém apenas os CEUs. Vamos juntar todas as informações de endereço e agregar a elas o texto “, Sao Paulo, Brazil”

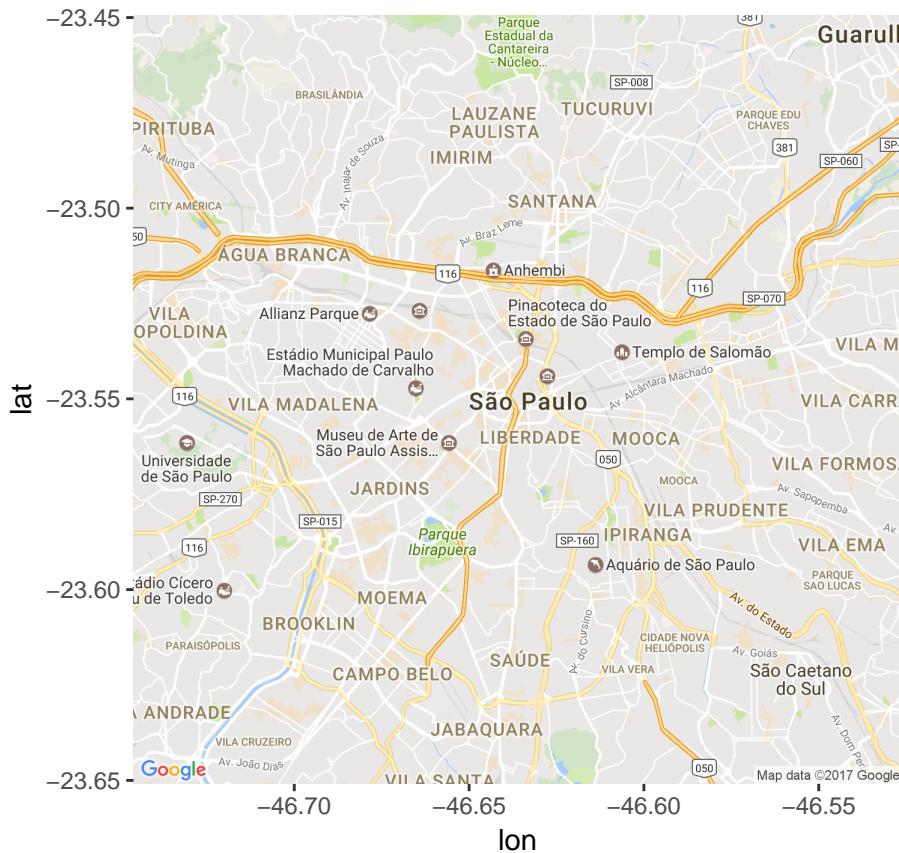
```
ceu <- escolas %>%
  filter(TIPOESC == "CEU") %>%
  mutate(endereco = paste(ENDERECHO, NUMERO, BAIRRO, CEP, ", Sao Paulo, Brazil"))
```

Com a função *geocode*, procuraremos a latitude e longitude dos 46 CEUs. Vamos ver o exemplo do primeiro CEU:

```
ceu1 <- geocode(ceu$endereco[1])
```

```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=ESTRADA%20D0%20ALVAR
ggmap(map_sp,
      base_layer = ggplot(aes(lon, lat), data = ceu1)) +
      geom_point()

## Warning: Removed 1 rows containing missing values (geom_point).
```



Usando o que já havíamos visto, obtivemos um mapa com um único ponto. De fato, se procurarmos a informação deste CEU (CEU Alvarenga), veremos que obtivemos a latitude e longitude corretas.

Com um *for loop*, procuraremos a latitude e longitude dos 46 endereços:

```
# Para remover problemas com encoding, removi algumas observações usando
# as linhas marcadas como comentário abaixo. No Windows pode ser desnecessário
ceu$endereco <- iconv(ceu$endereco, to = "ASCII//TRANSLIT")
ceu <- ceu[!is.na(ceu$endereco),]
latlong <- data.frame()
for (i in 1:nrow(ceu)){
  latlong <- bind_rows(latlong, geocode(ceu$endereco[i]))
}

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=ESTRADA%20DO%20ALVAR
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%200LGA%20FADEL%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20ENGENHEIRO%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20INTERLAGOS%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20BARAO%20BARROS%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20BARBINOS%20111%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20PERA-MARMELO%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20BERNARDO%20JOS%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20CINIRA%20POLON%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=ESTRADA%20DO%20BARRO%
```

```

## .
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20MARECHAL%20
## .
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20CLARA%20PETREL
## .
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20CARLOS%20L
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20DOMINGOS%20TAR
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20ANTONIO%20
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=ESTRADA%20DA%20BARON
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20CANTOS%20D
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20APARECIDA%20DO
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20MANUEL%20DA%20
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20MANUEL%20QUIRIL
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20ENGENHEIRO
## .
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20DANIEL%20GRAN%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20PROFESSOR%20ART
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=RUA%20NAZIR%20MIGUEL
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20CONDESSA%20
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20KENKITI%20
## .
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=AVENIDA%20BENTO%20GU
ceu <- bind_cols(ceu, latlong)

```

Simples, não! O principal problema da função `geocode` é que há um limite de consultas por conta dos termos da Google Maps API. A alternativa é usar como argumento “source = ‘dsk’” Data Science Toolkit, que reune uma série de fontes de dados e utiliza outra API para a consulta de latitude e longitude de logradouros.

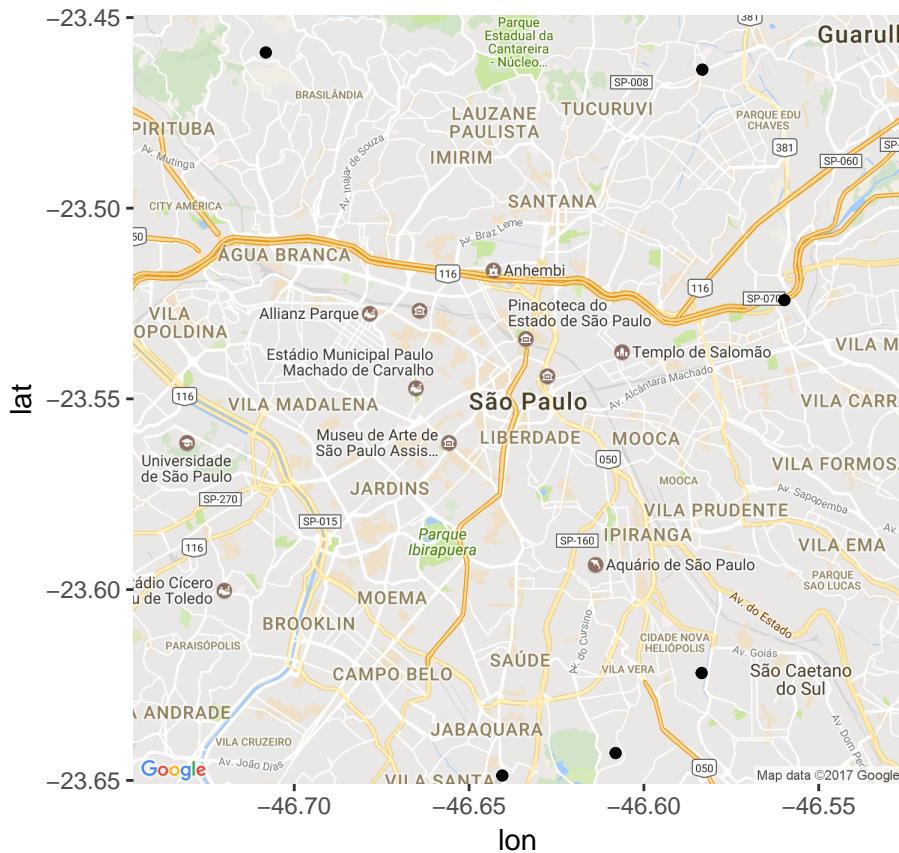
Vamos agora elaborar um mapa com os pontos obtidos para cada CEU:

```

ggmap(map_sp,
      base_layer = ggplot(aes(lon, lat), data = ceu)) +
      geom_point()

## Warning: Removed 21 rows containing missing values (geom_point).

```



## Pontos, linhas, polígonos e classes sp

Em nossos exemplos utilizamos *data frames* para armazenar as informações espaciais. Isso foi possível por que, ao trabalharmos com latitude e longitude de pontos, ficamos limitados a um único sistema de coordenadas e tivemos pouca precisão de mais de uma linha do data frame para armazenar o “desenho” de um objeto (ponto) em um mapa.

Há vários sistemas de coordenadas utilizados em diferentes bases de dados cartográficas. Assim, por um lado, precisamos de métodos que nos permitam transformar um sistema em outro para combinar diferentes fontes de dados. Dados especiais armazenados em data frames não contêm referência alguma ao sistema de coordenadas ao qual pertencem.

Por outro lado, se precisarmos de mapas que incluem outras representações geométricas, como linhas e pontos, somos obrigados a multiplicar o número de linhas em um data frame pelo número de vértices de tais figuras (2 vezes para uma linha e  $n$  vezes para um polígono, onde  $n$  é o número de vértices do polígono). Esse método, ainda que possível, é muito ineficiente.

A solução é trabalharmos com classes de objetos mais complexas (e completas) do que data frames. Vamos falar sobre isso ao explorarmos as principais classes do pacote *sp*. Em nosso tutorial vamos trabalhar com as classes de objetos do pacote *sp* e com as funções disponíveis para sua manipulação.

Antes disso, porém, vamos conhecer o formato mais comum para armazenar dados espaciais – *shapefile* – e como abri-los em R.

## Shapefiles

```
library(sp)
```

Ademais de precisarmos de outras classes de objetos em R para armazenarmos informação espacial, precisamos também de formatos de arquivos diferentes para compartilhar dados espaciais. *Shapefiles* são os mais populares e diversos repositórios de dados espaciais, como a Prefeitura de São Paulo e o Centro de Estudos da Metrópole (CEM).

Nosso primeiro exemplo foi retirado do site do CEM e ligeiramente modificado. Sugiro que você dê uma olhada nas bases cartográficas do CEM, pois há bastante coisa interessante lá.

Vamos começar fazendo o download do arquivo .zip onde estão os dados:

```
url_shp_eleicoes_sp <- "https://github.com/leobarone/FLS6397/raw/master/data/rmsp.zip"
download.file(url_shp_eleicoes_sp, "temp.zip")
unzip("temp.zip")
```

Veja que o arquivo descompactado é uma pasta (“rmsp”) com arquivos de diversas extensões. Não vamos entrar nos detalhes dos arquivos. Quando falamos de “shapefile”, estamos nos referindo a arquivos .shp. Mas raramente ele vem desacompanhado. A informação que nos importa é: todos eles, exceto, obviamente, o arquivo em .pdf, farão parte do objeto que criaremos.

```
list.files("rmsp")
```

```
## [1] "MunRM07.dbf" "MunRM07.pdf" "MunRM07.shp" "MunRM07.shx"
```

É fundamental que os arquivos de diferentes extensões tenham o mesmo nome, inclusive maiúsculas e minúsculas, e que os nomes das extensões estejam em letras minúsculas. Por esta razão, tome cuidado ao usar os arquivos do CEM e da PMSP, pois, em geral, algum arquivo vem com o nome e extensão em letras maiúscula.

A biblioteca para abertura de dados espaciais em R é *rgdal*. *readORG*, função de *rgdal* que utilizaremos para abrir os dados, recebe dois argumentos: a pasta em que estão os arquivos – no nosso caso “rmsp”, e o nome dos arquivos sem extensão (e por esta razão é fundamental que tenham o mesmo nome):

```
library(rgdal)
```

```
## rgdal: version: 1.2-7, (SVN revision 660)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 1.11.3, released 2015/09/16
## Path to GDAL shared files: /usr/share/gdal/1.11
## Loaded PROJ.4 runtime: Rel. 4.9.2, 08 September 2015, [PJ_VERSION: 492]
## Path to PROJ.4 shared files: (autodetected)
## Linking to sp version: 1.2-4
rmsp <- readOGR("rmsp", "MunRM07")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "rmsp", layer: "MunRM07"
## with 39 features
## It has 8 fields
```

Vamos observar a classe do objeto importado:

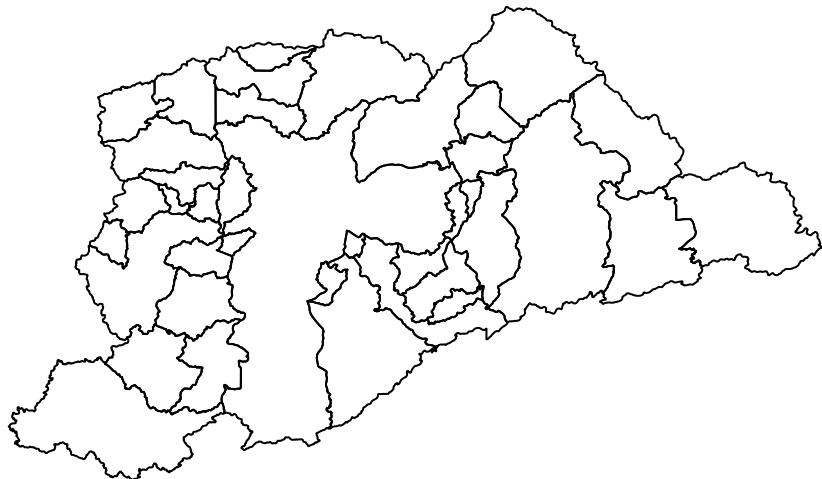
```
class(rmsp)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

“SpatialPolygonsDataFrame” é, como é fácil deduzir, um objeto espacial que contém polígonos (municípios da Região Metropolitana de São Paulo) e que acompanha um data frame. Falaremos sobre essa classe de objetos a seguir.

Tal classe de objetos pode ser rapidamente visualizada utilizando o comando *plot*:

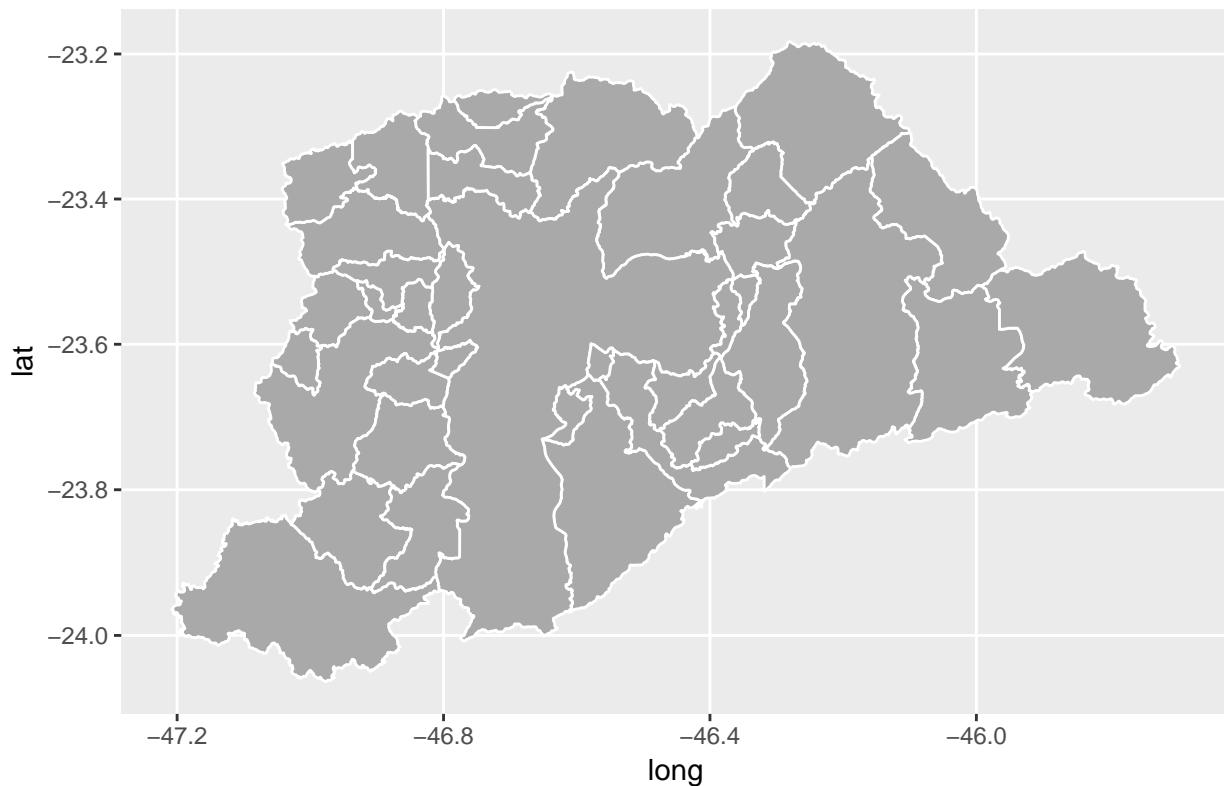
```
plot(rmsp)
```



Podemos utilizar o pacote *ggplot2* e sua gramática para plotar objetos da classe “SpatialPolygonsDataFrame”:

```
ggplot(data = rmsp,
       aes(x = long, y = lat, group = group)) +
  geom_polygon(color = 'white', fill = 'darkgrey') +
  coord_map()

## Regions defined for each Polygons
```



### Exercício:

Vá a uma das duas fontes de mapas indicadas – Prefeitura de São Paulo ou Centro de Estudos da Metrópole (CEM) – importe os dados e produza um mapa. Dependendo do tema que você escolher, você produzirá mapas com polígonos (por exemplo, mapas de divisões políticas ou administrativas), linhas (ruas, ciclovias, etc) ou pontos (unidades de saúde, escolas, etc).

### Classes do pacote sp e data frames

Se nosso objetivo fosse apenas importar os dados e produzir um mapa, poderíamos para na sessão anterior. Contudo, a regra é tentarmos combinar bases de dados espaciais entre si ou com data frames que contém informações não presentes no arquivos que acompanham o “shapefile”. No Desafio 4, adicionaremos informações geradas no Desafio 3 ao mapa que acabamos de produzir com municípios da região metropolitana, por exemplo.

Como combinar dois “SpatialPolygonsDataFrame” com sistemas coordenadas diferentes? Como relacionar um data frame a um objeto da classe “SpatialPolygonsDataFrame”?

Vamos conhecer o que há dentro de um “SpatialPolygonsDataFrame” com a função `str`. Para não poluir o console, vamos adicionar o argumento “`max.level = 2`”:

```
str(rmsp, max.level = 2)
```

```
## Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
##   ..@ data      :'data.frame': 39 obs. of  8 variables:
##   ..@ polygons  :List of 39
##   ..@ plotOrder : int [1:39] 12 37 10 5 11 1 16 6 3 4 ...
##   ..@ bbox      : num [1:2, 1:2] -47.2 -24.1 -45.7 -23.2
##   ... .- attr(*, "dimnames")=List of 2
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
```

Um “SpatialPolygonsDataFrame” contém vários elementos. O principal deles são, obviamente, os polígonos. Para o objeto “rmsp”, são 39 polígonos. Já entraremos no detalhe de cada um. “plotOrder” e “bbox” são elementos do objeto que definem, respectivamente, a ordem de “plotagem” e as dimensões de um retângulo que contém todos os polígonos e raramente nos importaremos com ambos.

“proj4string” armazena a informação sobre qual é sistema de coordenadas ao qual as coordenadas dos polígonos pertencem.

Um objeto da classe “SpatialPolygons” contém estes quatro elementos. Os objetos “SpatialPolygonsDataFrame” são iguais aos “SpatialPolygons” com a adição do elemento @data.

“SpatialLines”, “SpatialLinesDataFrame”, “SpatialMultiPoints” e “SpatialMultiPointsDataFrame” são as classes correspondentes para linhas e pontos e tem estrutura semelhante.

Podemos selecionar um elemento da estrutura de um objeto das classes do pacote *sp* usando o símbolo @. Vamos separar o objeto “rmsp” em 3 novos objetos:

```
rmsp_data <- rmsp@data
rmsp_poligonos <- rmsp@polygons
rmsp_projecao <- rmsp@proj4string
```

“rmsp\_data” é um data frame com os dados dos municípios:

```
head(rmsp_data)
```

	ID	COD_IBGE	SIGLA	NOME	NOMECAPS	POP_2000	DENS_DEMO
## 0	74	3546801	SIS	Santa Isabel	SANTA ISABEL	43740	120
## 1	76	3518305	GMA	Guararema	GUARAREMA	21904	81
## 2	75	3506607	BBM	Biritiba-Mirim	BIRITIBA MIRIM	24653	77
## 3	211	3518800	GRU	Guarulhos	GUARULHOS	1072717	3361
## 4	238	3545001	SPS	Sales\xf3polis	SALESOPOLIS	14357	34
## 5	246	3528502	MRP	Mairipor\xe3	MAIRIPORA	60111	187
##			AREA_KM2				
## 0			364.1687				
## 1			271.8909				
## 2			318.7029				
## 3			319.0683				
## 4			424.5088				
## 5			321.4058				

“rmsp\_projecao” é um

Conhecendo a estrutura de tais classes, já temos uma pista do que precisamos para (1) colocar duas projeções no mesmo sistema de coordenadas (alterando a informação sobre projeção) e (2) adicionar dados provenientes de um data frame (combinando-o com o que estiverem em @data). Faremos este último a seguir e não cobriremos o primeiro. Antes disso, vamos entender a complexa estrutura dos elementos em @polygons.

Vamos criar o objeto “poligono1” com o primeiro elemento de “rmsp\_poligonos” e examiná-lo.

```
poligono1 <- rmsp_poligonos[[1]]
str(poligono1)
```

```
## Formal class 'Polygons' [package "sp"] with 5 slots
##   ..@ Polygons :List of 1
##     ... .$. :Formal class 'Polygon' [package "sp"] with 5 slots
##       ... . . .@ labpt  : num [1:2] -46.2 -23.3
##       ... . . .@ area   : num 0.0321
##       ... . . .@ hole   : logi FALSE
##       ... . . .@ ringDir: int 1
##       ... . . .@ coords : num [1:436, 1:2] -46.2 -46.2 -46.2 -46.2 -46.2 ...
```

```

## ..@ plotOrder: int 1
## ..@ labpt    : num [1:2] -46.2 -23.3
## ..@ ID       : chr "0"
## ..@ area     : num 0.0321

```

Note que ele é da classe “Polygons”, também do pacote *sp*. Um polígono contém várias informações: os vértices do polígono; as conexões entre os vértices; a existências de “buracos” no polígono (que o buraco é também um polígono e polígonos com buracos lembram rosquinhas); etc. Não vamos examinar o que há em cada polígono, mas você já deve ter percebido que “SpatialPolygons” e “SpatialPolygonsDataFrame” são conjuntos de “Polygons” com algumas informações adicionais (dentre as quais um data frame com características de cada polígono no caso do último). “SpatialPoints” e “Lines” são classes análogas.

A não se que você deseje criar novos (ou redesenhar) polígonos, não convém examinar essa classe de objetos.

## Particionando um objeto de dados espaciais

Eventualmente, não são todos os polígonos importados de um “shapefile” que interessam. As classes do pacote *sp* aceitam seleção de polígonos da mesma forma que os data frames. No objeto “rmsp”, São Paulo é o 12º município, como vemos pelo vetor de nomes dos municípios em “rmsp\_data”:

```
rmsp_data$NOME
```

```

## [1] Santa Isabel           Guararema
## [3] Biritiba-Mirim        Guarulhos
## [5] Sales\xf3polis        Mairipor\xe3
## [7] Franco da Rocha       Santana de Parna\xedba
## [9] Pirapora do Bom Jesus Juquitiba
## [11] S\xe3o Bernardo do Campo S\xe3o Paulo
## [13] Cajamar              Itapevi
## [15] Santo Andr\xe9         Cotia
## [17] S\xe3o Louren\xe7o da Serra Osasco
## [19] Carapicu\xedba        Jandira
## [21] Francisco Morato     Rio Grande da Serra
## [23] Itaquaquecetuba      Suzano
## [25] Mau\xe1               Itapecerica da Serra
## [27] Embu                  Tabo\xe3o da Serra
## [29] Diadema              S\xe3o Caetano do Sul
## [31] Aruj\xe1              Po\xe1
## [33] Embu-Gua\xe7u          Barueri
## [35] Caieiras             Ferraz de Vasconcelos
## [37] Moji das Cruzes       Ribeir\xe3o Pires
## [39] Vargem Grande Paulista
## 39 Levels: Aruj\xe1 Barueri Biritiba-Mirim Caieiras ... Vargem Grande Paulista

```

Podemos separar o polígono de São Paulo e criar seu mapa da seguinte forma:

```

saopaulo <- rmsp[12,]
plot(saopaulo)

```



Repetindo o exemplo, mas para os 3 municípios do ABC:

```
posicoes_abc <- c(11, 15, 30)
abc <- rmsp[posicoes_abc,]
plot(abc)
```



Examinando a estrutura de “abc” notamos que o conjunto de polígonos e o data frame contém agora apenas 3 observações, como esperávamos.

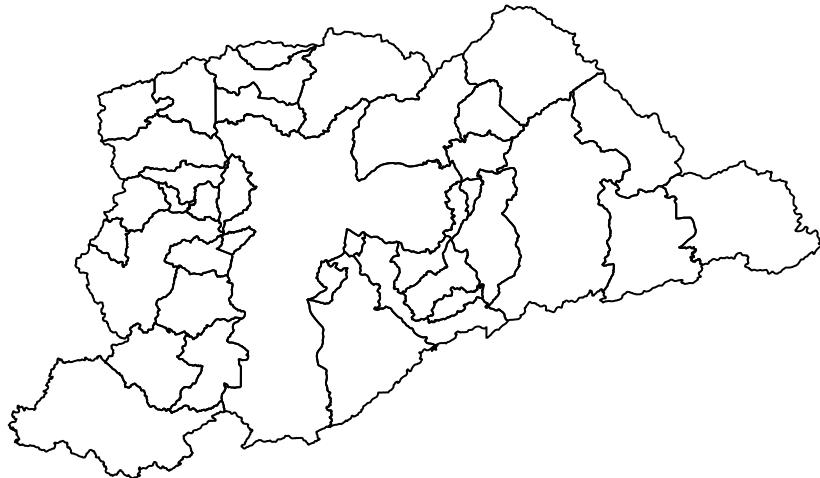
```
str(abc, max.level = 2)

## Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
##   ..@ data      : 'data.frame': 3 obs. of 8 variables:
##   ..@ polygons  : List of 3
##   ..@ plotOrder : int [1:3] 1 2 3
##   ..@ bbox      : num [1:2, 1:2] -46.7 -24 -46.3 -23.6
##   ... ..- attr(*, "dimnames")=List of 2
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
```

### Agregando e visualizando características de objetos espaciais

Vamos voltar ao nosso mapa da Região Metropolitana de São Paulo:

```
plot(rmsp)
```



Bastante sem graça, certo? Não estamos apresentando nenhuma característica dos municípios no mapa, apenas os polígonos que os definem.

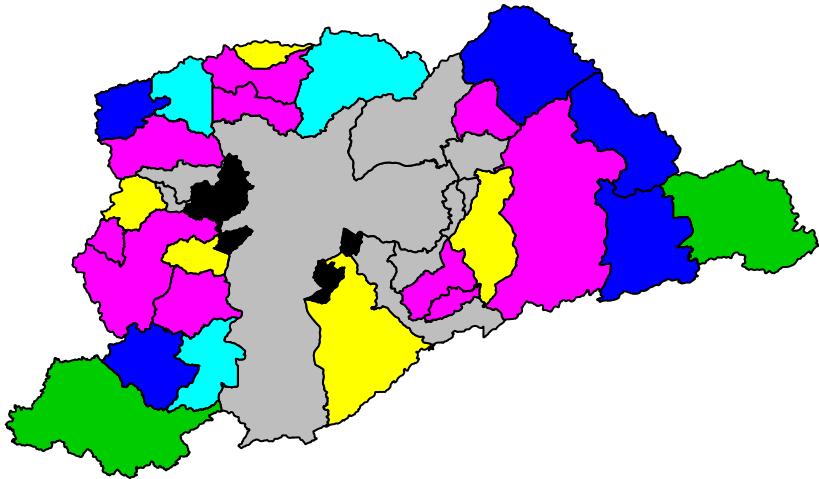
Os dados que acompanham o “shapefile”, porém, nos apresentam uma característica interessante dos municípios: a densidade demográfica (“DENS\_DEMO”)

```
head(rmsp@data)
```

```
##      ID COD_IBGE SIGLA          NOME      NOMECAPS POP_2000 DENS_DEMO
## 0    74  3546801   SIS Santa Isabel     SANTA ISABEL    43740      120
## 1    76  3518305   GMA Guararema      GUARAREMA    21904       81
## 2    75  3506607   BBM Biritiba-Mirim  BIRITIBA MIRIM    24653       77
## 3   211  3518800   GRU Guarulhos     GUARULHOS  1072717     3361
## 4   238  3545001   SPS Sales\xf3polis SALESOPOLIS    14357       34
## 5   246  3528502   MRP Mairipor\xe3    MAIRIPORA    60111      187
##      AREA_KM2
## 0 364.1687
## 1 271.8909
## 2 318.7029
## 3 319.0683
## 4 424.5088
## 5 321.4058
```

Vamos repetir o mapa, colorindo os municípios de acordo com a densidade demográfica:

```
plot(rmsp, col = log(rmsp@data$DENS_DEMO))
```



Não muito bonito, mas bastante mais interessante. Vamos trazer uma variável externa aos dados para tornar nosso exemplo mais atraente. Utilizaremos os dados de planejamento urbano da MUNIC-IBGE 2015. Note que, por parcimônia, não estamos buscando os dados diretamente na fonte, mas em uma cópia dos dados no repositório do curso.

No nosso exemplo buscaremos o ano de elaboração do plano diretor de cada município da RMSP nas informações sobre planejamento urbano. Separaremos esta variável de interesse e o código do município, ambas variáveis juntas renomeadas:

```
url_munic_15 <- "https://raw.githubusercontent.com/leobarone/FLS6397/master/data/planejamento_munic_2015.csv"
munic_15 <- read.table(url_munic_15, header = T, sep = ";") %>%
  rename(COD_IBGE = A1, ano_pd = A18) %>%
  select(COD_IBGE, ano_pd) %>%
  mutate(ano_pd = as.numeric(as.character(ano_pd)))

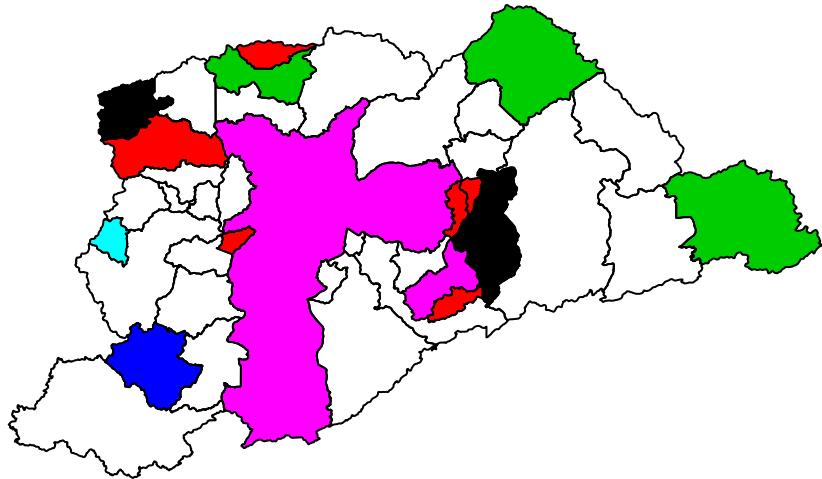
## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion
```

Para adicionarmos essa informação ao objeto de dados espaciais, basta fazermos um *left\_join* com os dados em @data:

```
rmrsp@data <- rmrsp@data %>%
  left_join(munic_15, by = "COD_IBGE")
```

Nada de novo, exceto o fato de que o data frame está dentro de um objeto mais complexo. Vamos agora plotar o ano de elaboração dos Planos Diretores de cada município como cores nos mapas:

```
plot(rmsp, col = factor(rmsp@data$ano_pd))
```



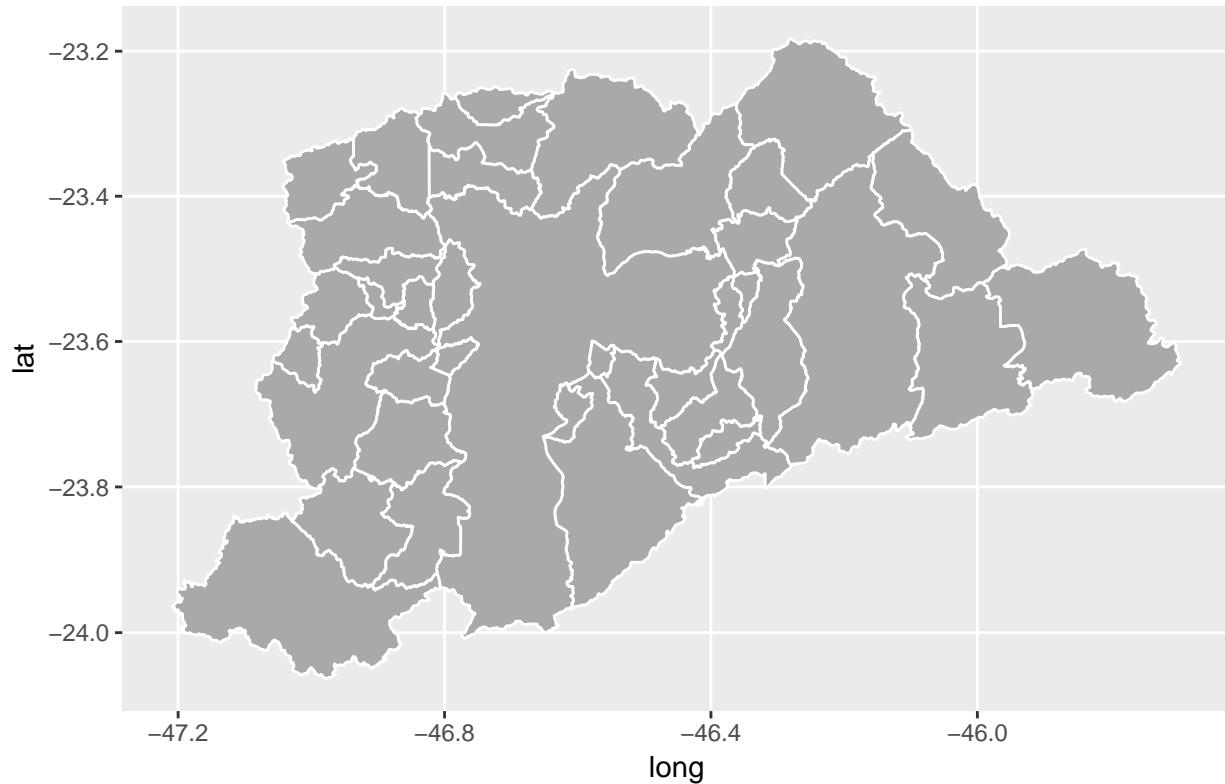
De novo, bastante feio, mas conseguimos trazer rapidamente dados de uma fonte externa para nosso mapa.

### Deixando o mapa bonito com *ggplot2*

Vimos anteriormente que podemos plotar mapas usando o pacote *ggplot2*. Vejamos o mapa básico:

```
ggplot(data = rmsp,
       aes(x = long, y = lat, group = group)) +
  geom_polygon(color = 'white', fill = 'darkgrey') +
  coord_map()
```

## Regions defined for each Polygons



Para adicionarmos novas variáveis utilizando o *ggplot2*, porém, precisamos reorganizar nossos dados. Infelizmente, se simplesmente adicionarmos uma variável que esteja em @data ao argumento “fill”, por exemplo, obteremos um mensagem de erro.

A solução será reorganizar os dados de polígonos em um data frame novo com a função *fortify*, perdendo justamente a eficiência que as classes de objetos do pacote *sp* forneceram.

Na sequência abaixo, criaremos “rmssp\_df” com a função *fortify*. “rmssp\_df” terá apenas variáveis “espaciais”. Para adicionar as demais variáveis que estão em @data, teremos de criar um “id” comum a ambos os data frames e realizar um *left\_join*:

```
rmssp_df <- fortify(rmsp)
```

```
## Regions defined for each Polygons
rmssp_df$id <- as.numeric(rmsp_df$id)
rmssp@data$id <- 0:(nrow(rmsp@data)-1)
rmssp_df <- left_join(rmsp_df, rmssp@data, by = "id")
```

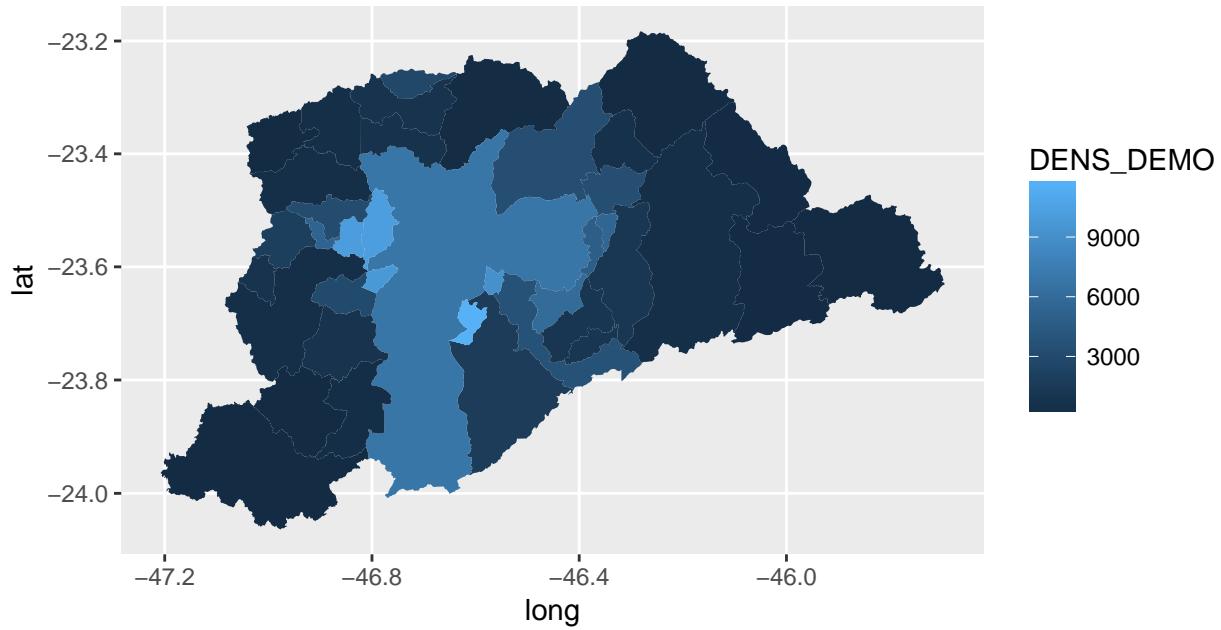
Observemos o objeto “rmssp\_df”:

```
glimpse(rmsp_df)
```

```
## Observations: 19,673
## Variables: 16
## $ long      <dbl> -46.15734, -46.16225, -46.16271, -46.16266, -46.1622...
## $ lat       <dbl> -23.34258, -23.34526, -23.34622, -23.34684, -23.3475...
## $ order     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1...
## $ hole      <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FAL...
## $ piece     <fctr> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ id        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ group    <fctr> 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0...
## $ ID        <int> 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, ...
## $ COD_IBGE <dbl> 3546801, 3546801, 3546801, 3546801, 3546801, 3546801...
## $ SIGLA    <fctr> SIS, SIS, SIS, SIS, SIS, SIS, SIS, SIS, S...
## $ NOME     <fctr> Santa Isabel, Santa Isabel, Santa Isabel, Santa Isa...
## $ NOMECAPS <fctr> SANTA ISABEL, SANTA ISABEL, SANTA ISABEL, SANTA ISA...
## $ POP_2000 <dbl> 43740, 43740, 43740, 43740, 43740, 43740, 43740, 437...
## $ DENS_DEMO <dbl> 120, 120, 120, 120, 120, 120, 120, 120, 120, 12...
## $ AREA_KM2 <dbl> 364.1687, 364.1687, 364.1687, 364.1687, 364.1687, 36...
## $ ano_pd   <dbl> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007...
```

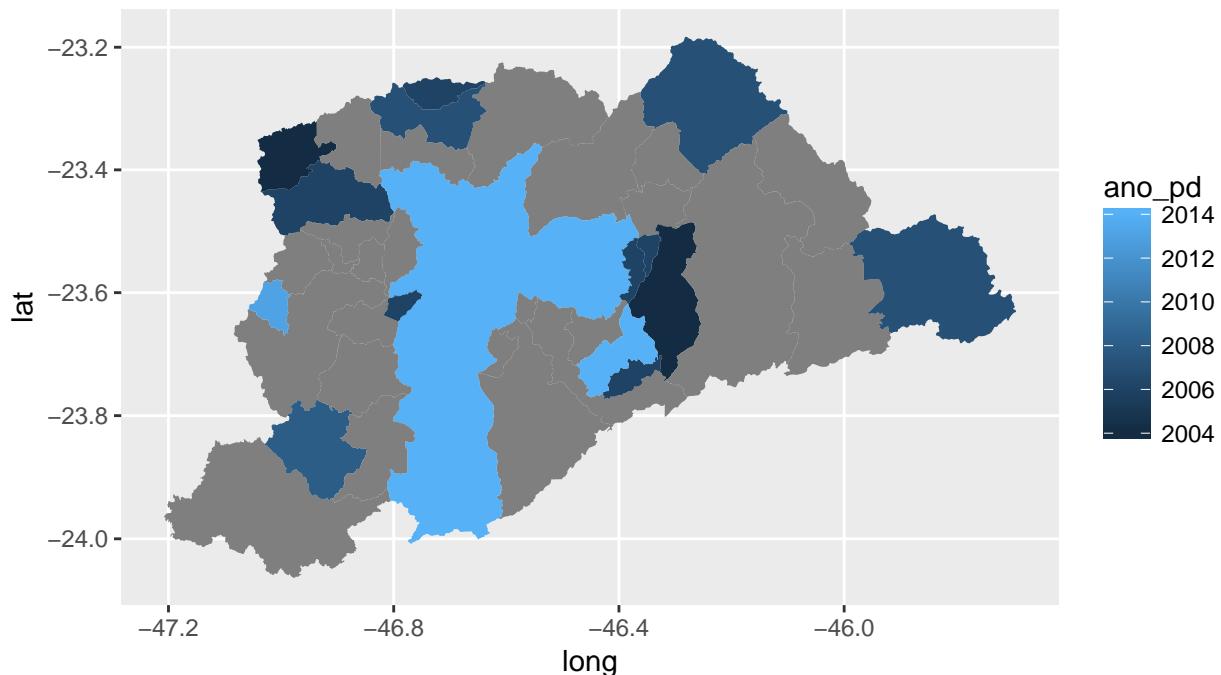
Note que todas as variáveis que precisamos estão aí. Reorganizando os argumentos do nosso mapa anterior, produzido com *ggplot*, podemos gerar mapas bastante mais elegantes. O primeiro, usando a informação sobre densidade demográfica:

```
ggplot(data = rmssp_df,
       aes(x = long, y = lat, group = group, fill = DENS_DEMO)) +
  geom_polygon() +
  coord_map()
```



E agora o ano dos planos diretores, variável que contém um número grande de missing values para os municípios da RMSP:

```
ggplot(data = rmsp_df,
       aes(x = long, y = lat, group = group, fill = ano_pd)) +
  geom_polygon() +
  coord_map()
```



A grande vantagem de trabalharmos com a gramática do *ggplot2* é que podemos editar os mapas da mesma forma que editamos gráficos e as possibilidades são inúmeras.

**Fim**