Fundamentals of Spatial Analysis in R

Marc Weber

2020-02-28

Contents

Chapter 1

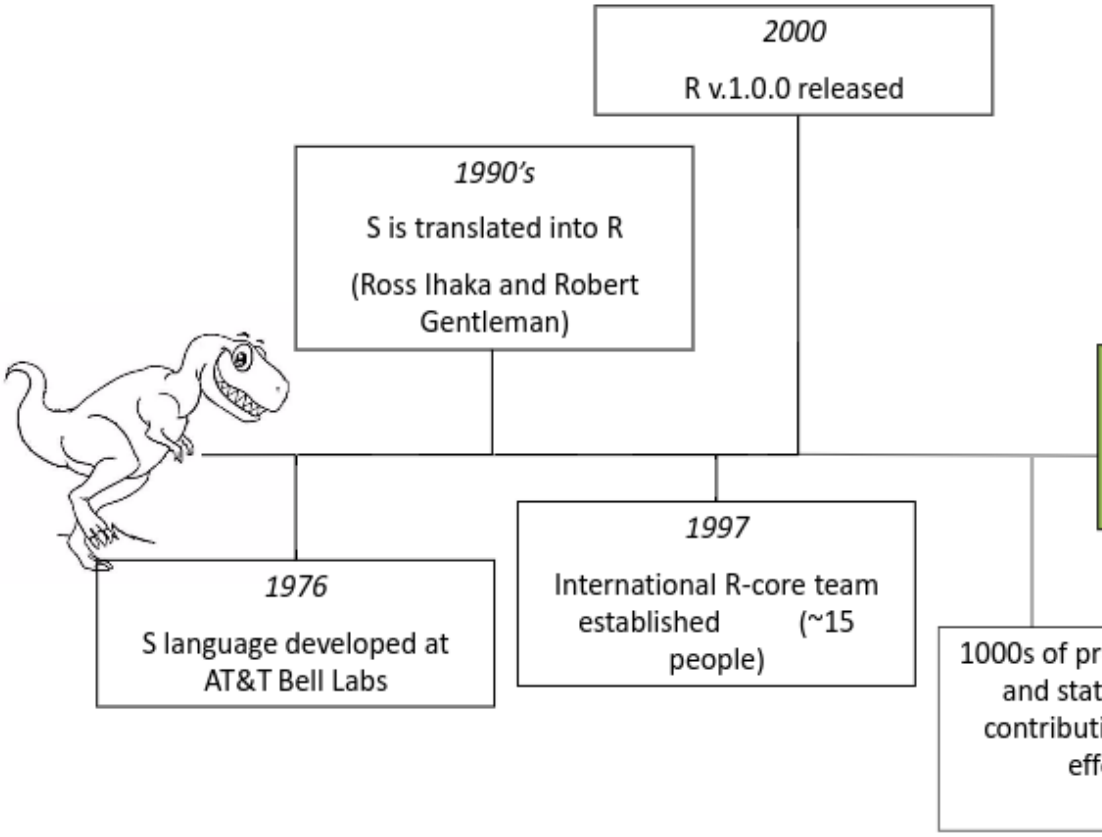
Introduction

- A bit about me
- Intros
- What is everyone's experience level?
- What are you expected to know?
  - Basic R objects and methods
  - `tidyverse` packages and syntax
    - i.e. `ggplot2`, `dplyr`, `readr`, `tidyr`, the pipe operator %>%
  - Very basic familiarity with RMarkdown and with git
  - If these are new, don't sweat it - google them, we'll talk through them as we go if needed
- This portion of workshop there no expectation of experience with spatial in R - if you already have some, you are sure to pick up new tricks - if you don't, we'll cover all the basics

1.0.1 Overview

What is R and why should we use R for spatial analysis? Let's break that into two questions - first, what is R and why should we use it?

- A language and environment for statistical computing and graphics
- R is lightweight, free, open-source and cross-platform
- Works with contributed packages - currently 15,362 - extensibility
- Automation and recording of workflow (reproducibility)
- Optimized work flow - data manipulation, analysis and visualization all in one place
- R does not alter underlying data - manipulation and visualization in memory



of R.bb

Figure 1.1: History of R

- R is great for repetetive graphics

Second, why use R for spatial, or GIS, work?

- Spatial and statistical analysis in one environment
- Leverage statistical power of R (i.e. modeling spatial data, data visualization, statistical exploration)
- Can handle vector and raster data, as well as work with spatial databases and pretty much any data format spatial data comes in
- R's GIS capabilities growing rapidly right now - new packages added monthly - currently about 200 spatial packages (depending on how you categorize)

Some drawbacks to using R for GIS work

- R not as good for interactive use as desktop GIS applications like ArcGIS or QGIS (i.e. editing features, panning, zooming, and analysis on selected subsets of features)
- Explicit coordinate system handling by the user, no on-the-fly projection support
- In memory analysis does not scale well with large GIS vector and tabular data
- Steep learning curve
- Up to you to find packages to do what you need - help not always great

1.1 Quick R basics review

```
getwd()
```

```
## [1] "F:/GitProjects/AWRA_2020_R_Spatial"
```

Which should return something like:

```
[1] "/home/marc/GitProjects/AWRA_GIS_R_Workshop"
```

To see what is in the directory:

```
dir()
```

```
##  [1] "_after_body.html"           "_book"
##  [3] "_bookdown.yml"              "_bookdown_files"
##  [5] "_output.yml"                "02-vector.Rmd"
##  [7] "03-raster.Rmd"              "04-application.Rmd"
##  [9] "05-summary.Rmd"             "06-references.Rmd"
## [11] "AWRA_2020_R_Spatial.Rmd"    "AWRA_2020_R_Spatial.Rproj"
## [13] "AWRA_2020_R_Spatial_files" "book.bib"
## [15] "css"                        "docs"
## [17] "images"                     "index.Rmd"
## [19] "js"                         "packages.bib"
```

```
## [21] "preamble.tex"              "README.md"
```

To establish a different directory:

```
setwd("/home/marc/GitProjects")
```

1.1.0.1 Terminology: data structures

R is an interpreted language (access through a command-line interpreter) with a number of data structures (vectors, matrices, arrays, data frames, lists) and extensible objects (regression models, time-series, geospatial coordinates) and supports procedural programming with functions.

To learn about objects, become friends with the built-in `class` and `str` functions. Let's explore the built-in iris data set to start:

```
class(iris)
```

```
## [1] "data.frame"
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 .
```

As we can see, `iris` is a data frame and is used extensively for beginning tutorials on learning R. Data frames consist of rows of observations on columns of values for variables of interest - they are one of the fundamental and most important data structures in R.

But as we see in the result of str(iris) above, following the information that iris is a data frame with 150 observations of 5 variables, we get information on each of the variables, in this case that 4 are numeric and one is a factor with three levels.

First off, R has several main data types:

- logical
- integer
- double
- complex
- character
- raw
- list
- NULL
- closure (function)
- special

- builtin (basic functions and operators)
- environment
- S4 (some S4 objects)
- others you won't run into at user level

We can ask what data type something is using `typeof`:

```
typeof(iris)
```

```
[1] "list"
```

```
typeof(iris$Sepal.Length)
```

```
[1] "double"
```

```
typeof(iris$Specis)
```

```
[1] "integer"
```

We see a couple interesting things here - `iris`, which we just said is a data frame, is a data type of `list`. `Sepal.Length` is data type `double`, and in `str(iris)` we saw it was numeric - that makes sense - but we see that `Species` is data type `integer`, and in `str(iris)` we were told this variable was a factor with three levels. What's going on here?

First off, `class` refers to the abstract type of an object in R, whereas `typeof` or `mode` refer to how an object is stored in memory. So iris is an object of class `data.frame`, but it is stored in memory as a list (i.e. each column is an item in a list). Note that this allows data frames to have columns of different classes, whereas a matrix needs to be all of the same mode.

For our `Species` column, We see it's `mode` is numeric, it's `typeof` is `integer`, and it's class is `factor`. Nominal variables in R are treated as a vector of integers 1:k, where k is the number of unique values of that nominal variable and a mapping of the character strings to these integer values.

This allows us to quickly see see all the unique values of a particular nominal variable or quickly re-asign a level of a nominal variable to a new value - remember, everything in R is in memory, so don't worry about tweaking the data!

```
levels(iris$Species)
levels(iris$Species)[1] <- 'sibirica'
```

See if you can explain how that re-asignment we just did worked.

To access particular columns in a data frame, as we saw above, we use the `$` operator - we can see the value for `Species` for each observation in 'iris by doing:

```
iris$Species
```

To access particular columns or rows of a data frame, we use indexing:

```
iris[1,3] # the 1st row and the 3rd column
```

```
[1] 1.4
```

```
iris[4,5] # the 4th row and the 5th column
```

```
[1] sibirica
Levels: sibirica versicolor virginica
```

A handy function is `names`, which you can use to get or to set data frame variable names:

```
names(iris)
names(iris)[1] <- 'Length of Sepal'
```

Explain what this last line did

1.1.0.2 Overview of Classes and Methods

- Class: object types
  - `class()`: gives the class type
  - `typeof()`: information on how the object is stored
  - `str()`: how the object is structured
- Method: generic functions
  - `print()`
  - `plot()`
  - 'summary()

1.2 Workshop Data and Logistics

1.3 Spatial Data in R

1.4 Code along

Just a sampling of things we'll cover. Run code, examine output, ask any questions - we'll explore it all in more detail through the morning.

1.4.1 Geocoding example with tmaptools using open street map

```
# uses OSM
library(tmap)
library(tmaptools)
library(dplyr)
tex_cap <-tmaptools::geocode_OSM("Texas Capital",
```

```
        as.sf = TRUE) %>%
  glimpse()
```

```
## Observations: 1
## Variables: 8
## $ query    <chr> "Texas Capital"
## $ lat      <dbl> -31.46748
## $ lon      <dbl> -64.22844
## $ lat_min  <dbl> -31.46748
## $ lat_max  <dbl> -31.46748
## $ lon_min  <dbl> -64.22995
## $ lon_max  <dbl> -64.22723
## $ geometry <POINT [°]> POINT (-64.22844 -31.46748)
```

1.4.2 Interactive mapping

```
library(mapview)
mapview(tex_cap)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed,
```

1.4.3 Choropleth map

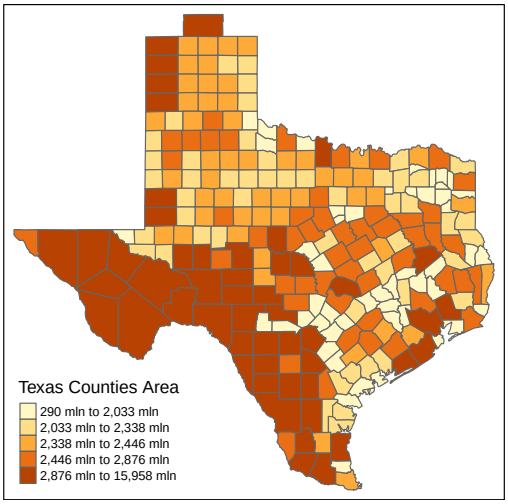
The package maps (automatically installed and loaded with ggplot2) provides maps of the USA, with state and county borders, that can be retrieved and converted as sf objects:

```
library(sf)
library(maps)
counties <- st_as_sf(map("county", plot = FALSE, fill = TRUE))
counties <- subset(counties, grepl("texas", counties$ID) & !grepl('missouri,texas',counties$ID))
counties$area <- as.numeric(st_area(counties))
head(counties)
```

```
## Simple feature collection with 6 features and 2 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -103.0751 ymin: 28.14942 xmax: -94.13123 ymax: 37.00161
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
##                            geometry             ID       area
## 2168 MULTIPOLYGON (((-101.6255 3... oklahoma,texas 5434507068
## 2491 MULTIPOLYGON (((-95.75271 3... texas,anderson 2817584981
## 2492 MULTIPOLYGON (((-102.2042 3...  texas,andrews 3962852909
## 2493 MULTIPOLYGON (((-94.13123 3... texas,angelina 2200352194
## 2494 MULTIPOLYGON (((-96.80122 2...  texas,aransas  290370313
```

```
## 2495 MULTIPOLYGON (((-98.42269 3...   texas,archer 2422607253
```

```
tm_shape(counties) +
  tm_polygons("area",
              style="quantile",
              title="Texas Counties Area")
```



example-1.bb

1.5 Challenge: Does this work?

Did my .css styling adjustment work?

1.5.1 Answer

1. Yes
2. No

Chapter 2

Vector data with sf

Load `tidycensus` - you'll need to set your Census API key. A key can be obtained from here.

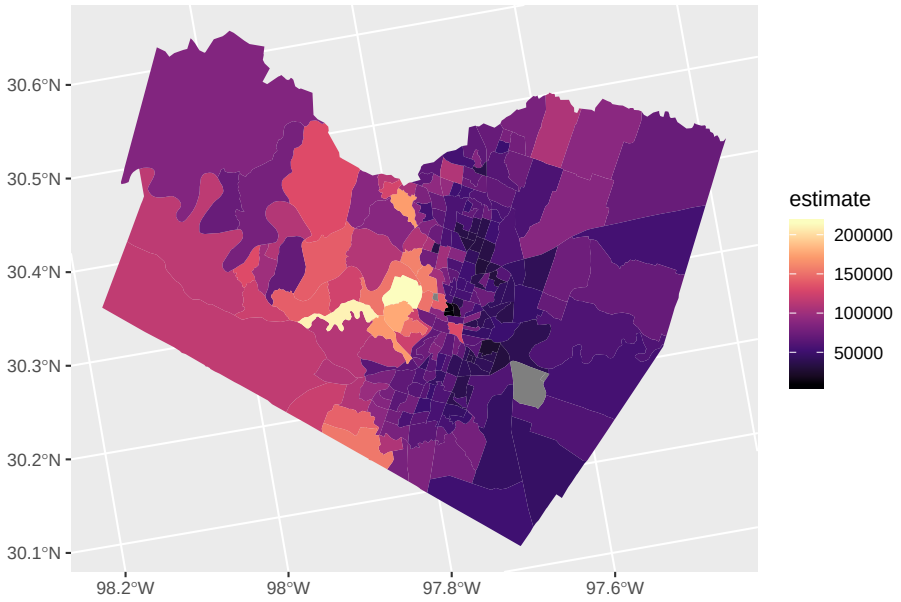
```
library(tidycensus)
library(tidyverse)

census_api_key("YOUR API KEY GOES HERE")
```

```
library(tidycensus)
library(ggplot2)
options(tigris_use_cache = TRUE)
austin_tracts <- get_acs(state = 'TX', county = 'Travis', geography = "tract",
                         variables = "B19013_001", geometry = TRUE)
```

```
## Getting data from the 2013-2017 5-year ACS
```

```
austin_tracts %>%
  ggplot(aes(fill = estimate)) +
  geom_sf(color = NA) +
  coord_sf(crs = 26911) +
  scale_fill_viridis_c(option = "magma")
```


30.6°N
30.5°N
30.4°N
30.3°N
30.2°N
30.1°N
98.2°W
98°W
97.8°W
97.6°W
estimate
200000
150000
100000
50000

Chapter 3

Raster data

Chapter 4

Applications

Some *significant* applications are demonstrated in this chapter.

4.1 Example one

4.2 Example two

Chapter 5

Final Words

We have finished a nice book.

References

5.0.1 R Spatial Resources

- R Spatial - **Spatial Data Science with R**
- **Geocomputation with R**
- **R Spatial Task View**
- **Modern Geospatial Data Analysis with R by Zev Ross**
- **Spatial Data Science - Pebesma and Bivand**
- **Spatial Data Science Course- Prof. Adam Wilson**
- **Introduction to Mapping and Spatial Analysis with R**
- **Google R Style Guide**
- **Advanced R by Hadley Wickham**
- **Intro to GIS and Spatial Analysis by Manuel Gimond**
- **FOSS4G2019 R for Geospatial Processing**
- **An Introduction to Spatial Analysis and Mapping in R**

5.0.2 R Vector Processing / Simple Features Resources

- **Simple Features for R**
- **Spatial Data in R: New Directions**
- **sp-sf Migration**
- **An Exploration of Simple Features for R**
- **Simple Features: Building Spatial Data Pipelines in R**
- **Tidy spatial data in R: using dplyr, tidyr, and ggplot2 with sf**

5.0.3 R Raster Resources

- Wageningen University **Intro to Raster**
- Wageningen University **Advanced Raster Analysis**
- **The Visual Raster Cheat Sheet GitHub Repo**
- **Rastervis**
- **stars - spatiotemporal arrays**

5.0.4 R Mapping Resources

- **mapview**
- **Leaflet for R**
- **tmap**
- Zev Ross **Creating beautiful demographic maps in R with the `tidycensus` and `tmap` packages**
- Geocomputation with R: **Making maps with R**
- Ryan Peek: **Mapping in R**