

Lecture 7: Maps 1 of 3

March 19, 2018

Overview

Course Administration

Good, Bad and Ugly

Maps, Lead In

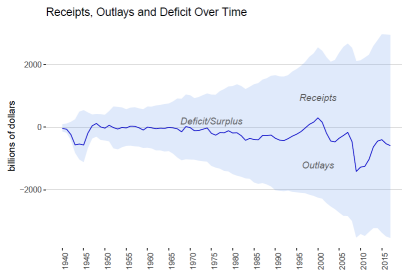
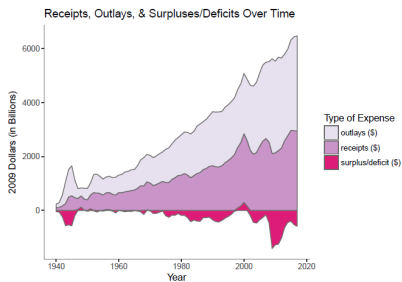
Representing Maps Digitally

Starting with Maps in R

Course Administration

1. Rosa has graded problem sets – thank you
2. Schedule consultations
 - form is online
 - meet in my office or call/hangout
 - Bring graph sketches at a minimum.
3. Will post in-class workshop instructions by the end of the week (please nag if I don't do it!)
4. Missing anything else from me?

Revenues and Outlays



Next Week's Good Bad and Ugly

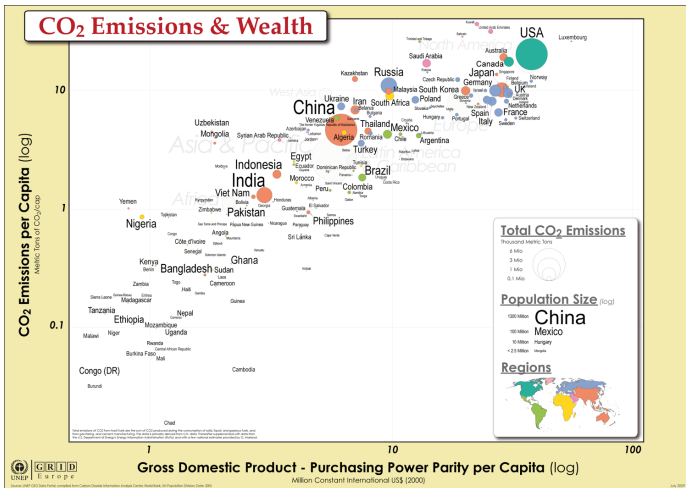
Monday by 9 am. Earlier is ok.

- Sophie Godfrey-McKee
- Colleen McBride

This Week's Good Bad and Ugly

- Raphe Breit
- Meghan Demeter

Raphe's Example

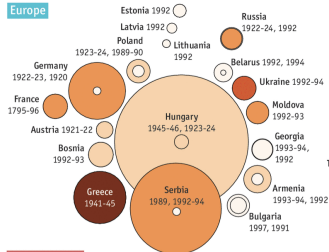


Meghan's Example

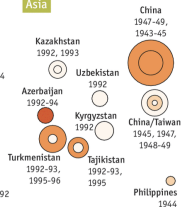
Who wants to be a zillionaire?

Historic cases of countries that have experienced hyperinflation, selected
Dates labelled in descending order of daily rate

Europe



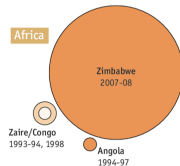
Asia



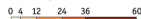
Latin America



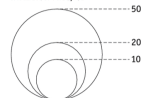
Africa



Duration, months



Maximum daily inflation rate, %



Maps: Why and How

Today

1. Why maps?
2. When do maps deceive?
3. How do we describe maps in R?
4. Save for next time: Choropleth maps

Why Maps?

- Use a map when you want to show a **spatial** relationship
- Don't use a map if you want to compare geographic units

When is Space Important?

1. To show relationship between two geographic things.
Examples?

When is Space Important?

1. To show relationship between two geographic things.
Examples?
 - metro stops relative to average home prices
 - population density relative to the equation
2. To show a geographic pattern in an outcome. Examples?

When is Space Important?

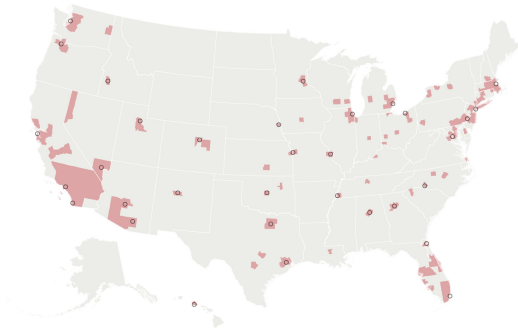
1. To show relationship between two geographic things.
Examples?
 - metro stops relative to average home prices
 - population density relative to the equation
2. To show a geographic pattern in an outcome. Examples?
 - voting outcomes correlated over space
 - geographic features that change smoothly and sharply over space

Don't use a map if you can do something simpler!

Why Avoid Maps?

- They add complexity
- Geographic unit size infrequently related to importance
 - but remember that size indicates value
 - problematic!
- Examples?

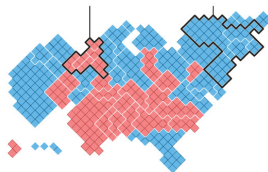
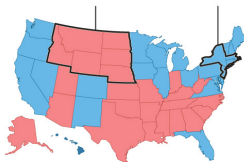
Red and Grey Areas Have About the Same Number of Votes Cast in 2012



With many thanks to the [Washington Post](#)

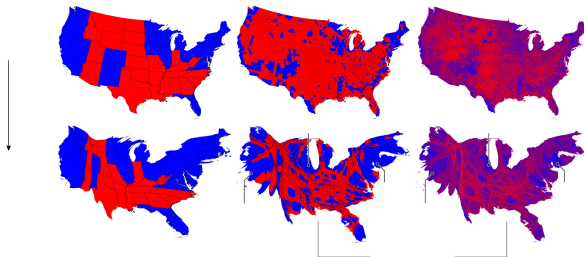
One Possible Solution

- A “cartogram” sizes locations by something: votes or people or electoral votes
- Five red midwestern states correspond to red block
- Mid-Atlantic corresponds to blue block

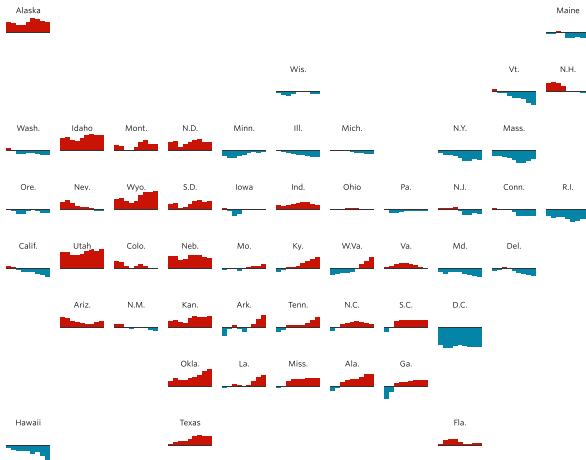


Another Possible Solution

- Thanks to U of Michigan physicist Newman
- Columns are state winner, county winner, county shaded by popular vote share
- Top is real map, bottom is cartogram
- Leftmost sized by electoral votes, others by votes cast



And a Quasi Map



Thanks to the Wall Street Journal, [here](#).

How Do Computers Make Maps?

Maps Have

- Units defined by coordinates in space
- Data for each unit

Examples of such units, please!

Three Major Types of Shapes for Maps

1. points
2. lines
3. polygons

Points in Space

- location 1: (x, y)
- location 2: (x, y)
- location 3: (x, y)

What would you represent with points?

Lines in Space

- location 1: $(x_1, y_1), (x_2, y_2)$
- location 2: $(x_1, y_1), (x_2, y_2)$
- location 3: $(x_1, y_1), (x_2, y_2)$


What would you represent with lines?

Polygons in Space

- location 1: $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_1, y_1)$
- location 2: $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_1, y_1)$
- location 3: $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_1, y_1)$

Note that last point is the same as the first point.¹

What would you represent with polygons?

¹Polygons can have holes; we can talk about this. 

But Where Do the Points Go?

- A map file needs some instructions on what the points mean
- We are not drawing on a globe, so we need some way of taking true coordinates and making them flat
- Map makers define coordinate systems so that everyone agrees on what (x_1, y_1) , (x_2, y_2) means
- Coordinate systems have a defined unit of measurement: meters, feet, decimal degrees
- There are two major types of systems
 1. geographic/global/spherical system: in latitude/longitude
 2. projected coordinate system: in terms of meters/feet/miles

Implications for Mapping

- You can't put maps with two different coordinate systems on top of each other
- Easier to calculate distances and areas with projected coordinate systems
- You can go from one projection to another, but **use the right command**
- Digital maps usually come with a projection defined

Maps in R, I

Today

A. Programming

- ▶ number of characters in a string: `nchar()`
- ▶ loops and `lapply`
- ▶ merging data

B. Getting started on maps

- ▶ packages for maps in R
- ▶ maps in R
- ▶ what a map file looks like
- ▶ s3 vs s4 classes
- ▶ loading revised `ggplot2` and `sf`

C. Making maps

- ▶ reading maps: `readOGR()` or `st_read()`
- ▶ how to find the “coordinate reference system”
- ▶ plot command

Programming

- ▶ number of characters in a string: `nchar()`
- ▶ loops and `lapply`
- ▶ merging data

Finding the length of a string

- ▶ `nchar()` reports the number of characters in a string
- ▶ we'll use it today

```
somejunk <- c("blahleahblah")  
nchar(somejunk)
```

```
## [1] 12
```

- ▶ can use this new variable length as a basis for doing other operations

Doing the Same Thing Many Times

- ▶ if you are doing the same thing many times
- ▶ don't copy and paste!
- ▶ why?
 - ▶ it's (maybe) easier the first time
 - ▶ but inevitably worse the 12th time
 - ▶ errors propagate
 - ▶ with looped code, you fix errors once

Basic idea of a loop

- ▶ Repeat some operation that is the same
- ▶ Except for an index number or string
- ▶ Here is something that could be put into a loop

```
print(paste0("hello number ", 1))
```

```
## [1] "hello number 1"
```

```
print(paste0("hello number ", 2))
```

```
## [1] "hello number 2"
```

```
print(paste0("hello number ", 3))
```

```
## [1] "hello number 3"
```

Basic idea of a loop

- ▶ Simpler code is

```
for(p in 1:3)
{
  print(paste0("hello number ", p))
}
```

```
## [1] "hello number 1"
```

```
## [1] "hello number 2"
```

```
## [1] "hello number 3"
```

A further improvement

- ▶ Loops are not “R-like”
- ▶ Use instead `lapply`, which takes even fewer lines

```
lapply(1:3, function(x){  
  print(paste0("hello number ", x))})
```

```
## [1] "hello number 1"  
## [1] "hello number 2"  
## [1] "hello number 3"
```

```
## [[1]]  
## [1] "hello number 1"  
##  
## [[2]]  
## [1] "hello number 2"  
##  
## [[3]]  
## [1] "hello number 3"
```

Merging Data

- ▶ This is a fundamental data management concept
- ▶ One of the best things statistical software does for you relative to Excel
- ▶ Basic idea
 - ▶ You have two datasets that you'd like to put together
 - ▶ Where you join them by some variable in both datasets

Merging Data Example: seta

```
seta <- data.frame( year = c(1999, 2000, 2001, 2002,  
                             2005, 2010),  
                    berries = c(6,8,7,9,10,6))  
seta
```

```
##   year berries  
## 1 1999        6  
## 2 2000        8  
## 3 2001        7  
## 4 2002        9  
## 5 2005       10  
## 6 2010        6
```

Merging Data Example: setb

```
setb <- data.frame( year = c(1999 ,2000, 2001, 2002,  
                             2003, 2005, 2010),  
                    jam = c(10,12,15,18,20,21,5))  
setb
```

```
##   year jam  
## 1 1999  10  
## 2 2000  12  
## 3 2001  15  
## 4 2002  18  
## 5 2003  20  
## 6 2005  21  
## 7 2010   5
```

Let R merge for you

- ▶ R can put these two datasets together
- ▶ How many observations should we have?

Let R merge for you

- ▶ R can put these two datasets together
- ▶ How many observations should we have?
- ▶ Here's how you do it

```
setab <- merge(seta, setb,  
               by.x = c("year"),  
               by.y = c("year"),  
               all = TRUE)
```

```
setab
```

##	year	berries	jam
## 1	1999	6	10
## 2	2000	8	12
## 3	2001	7	15
## 4	2002	9	18
## 5	2003	NA	20
## 6	2005	10	21
## 7	2010	6	5

Getting Started with Maps

- ▶ packages for maps in R
- ▶ loading revised `ggplot2` and `sf`
- ▶ maps in R
 - ▶ what a map file looks like
 - ▶ s3 vs s4 classes

Packages for Maps in R

We will focus on a few packages

- ▶ `rdgal`
 - ▶ lots of basic stuff to set up shapefiles
 - ▶ requires `sp` package
 - ▶ lets you open files with `readOGR`
 - ▶ write files with `writeOGR`
- ▶ `raster`
 - ▶ useful for gridded data
- ▶ `sf`
 - ▶ the “new” mapping package recommended by the most R guru person I know
 - ▶ simplifies many mapping tasks
 - ▶ seems like the future
 - ▶ but is so new you need to download the “development” version

Special Loading for Packages for Today

- ▶ We are using packages so new they are under development
- ▶ Install the `devtools` package
- ▶ Use this package to re-load `ggplot2` and `sf`
- ▶ As described in the tutorial
- ▶ If this doesn't end up working for you, use the older commands we review in the tutorial

Data classes in R

- ▶ We use two types of classes of files
 - ▶ dataframes are S3 class
 - ▶ old map files are s4 class: dataframe plus some complicated way of storing shapes
- ▶ New sf package
 - ▶ has a dataframe plus
 - ▶ a list, for each shape of points in polygon/line/point
 - ▶ MUCH faster for large files
- ▶ Can treat s3 class data like a dataframe
- ▶ Need to treat s4 class data differently

Making Maps

- ▶ reading maps: `readOGR()` or `st_read()`
- ▶ how to find the “coordinate reference system”
- ▶ plot command

How to read maps using rgdal

- ▶ First, you'll need to load a map
- ▶ It's not a csv, so we can't load it with `read.csv()`
- ▶ If using `rgdal` package, then

```
## Loading required package: sp
```

```
## rgdal: version: 1.2-16, (SVN revision 701)
```

```
## Geospatial Data Abstraction Library extensions to R suc
```

```
## Loaded GDAL runtime: GDAL 2.2.0, released 2017/04/28
```

```
## Path to GDAL shared files: C:/Users/lbrooks/Documents/I
```

```
## GDAL binary built with GEOS: TRUE
```

```
## Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_
```

```
## Path to PROJ.4 shared files: C:/Users/lbrooks/Documents
```

```
## Linking to sp version: 1.2-7
```

```
shape.rgdal <- readOGR(dsn = "h:/pppa_data_viz/2018/tutoriala  
layer = "gz_2010_us_050_00_500k")
```

```
## OGR data source using the OGR API
```

How to read maps using sf

- ▶ If using sf package, then

```
## Linking to GEOS 3.6.1, GDAL 2.2.3, proj.4 4.9.3
```

```
shape.sf <- st_read(dsn = "h:/pppa_data_viz/2018/tutorials/  
                      layer = "gz_2010_us_050_00_500k")
```

```
## Reading layer `gz_2010_us_050_00_500k' from data source  
## Simple feature collection with 3221 features and 6 fields  
## geometry type:  MULTIPOLYGON  
## dimension:      XY  
## bbox:           xmin: -179.1473 ymin: 17.88481 xmax: 179.1473 ymax: 42.69278  
## epsg (SRID):    4269  
## proj4string:     +proj=longlat +datum=NAD83 +no_defs
```

- ▶ dsn is the directory
- ▶ layer is the map name without any extension

Find a coordinate reference system of a file

Useful if you want to * know the projection of a map * compare the projection of two maps * know the units of the output

Find a coordinate reference system of a file

- Using sp package (s4 class)

```
library(sp)
library(raster)
projout <- crs(shape.rgdal)
projout
```

```
## CRS arguments:
```

```
## +proj=longlat +datum=NAD83 +no_defs +ellps=GRS80 +towgs84=0,0,0,0,0,0,0
```

- Using sf package (s3 class)

```
projout <- st_crs(shape.sf)
projout
```

```
## Coordinate Reference System:
```

```
## EPSG: 4269
```

```
## proj4string: "+proj=longlat +datum=NAD83 +no_defs"
```

Plot command

- ▶ The most simple way to make a map in R
- ▶ Works for any s4 class file
- ▶ R knows it's a map!

```
plot(mapin)
```

Try Today's Tutorial

- Ask questions if the command doesn't make sense
- Go forth!

Next Lecture

- Next week: no class
- But student consultations end of this week
- Next lecture is April 2
- In-class workshop April 9