Lab Exercise: Spatial Data Analysis - Mapping Brexit

Todd K Hartman

Last updated 2019-11-19

Getting Started

In this lab exercise, we're going to be downloading the 2016 EU Referendum voting records from the Electoral Commission to investigate how each of the 382 UK areas voted. We'll be ranking the them by the percentage of Leave voters and using a Google package (`googleVis`) to visualize the results. Then we'll download a UK shapefile to create the outline map, merge that data with the voting records, and create a *choropleth*, which is a thematic map with shaded regions in proportion to some variable in the dataset. In our case, you might have guessed that the shading will correspond to Leave vote share. It's an action packed session!

Before we can begin with the data, let's load the data manipulation and visualization packages needed to make the various figures. This is also a good time to make sure that we've set the working directory. You can do this in two steps or just use the `pacman` package manager to do it in one (but you'd need to install that package if you haven't already done so).

```
## Option A: You could do it this way, but I wouldn't...
list <- c("ggplot2", "googleVis", "sf")
install.packages(list)
lapply(list, library, character.only = TRUE)

## Option B: My preferred option to is load packages via 'pacman' package manager
pacman::p_load(ggplot2, googleVis, sf)

## Set the working directory
setwd("ENTER YOUR WORKING DIRECTORY HERE")
```

We're going to be downloading data directly from the EU Referendum Results section of the Electoral Commission website: https://goo.gl/nwRH8J

The .csv file can be downloaded from by clicking the link 'Download EU referendum results data in full (CSV)'. Make sure that you download the file in your working directory or R will be unable to see it.

```
## Load the 2016 EU Referendum (Brexit) vote share data
## Data is also available at Github
## https://github.com/tkhartman/mapping-brexit/blob/master/EU-referendum-result-data.csv
brexit <- read.csv("EU-referendum-result-data.csv")
brexit <- brexit[order(brexit$id), ]  # Sort the data by id (note the comma!)
head(brexit)
```

Downloading and Loading Data Directly in R [Optional]

In the steps above, we downloaded the data manually and imported it into R. However, we can also use R to automatically scrape the Electoral Commission website. There are many different ways to do this, but one easy way is to use the download.file() function. We're going to specify the location of the file we want to download (`url`), what we want to name the file once it's downloaded (`destfile`), and how we're going to write the file (`mode`). This last option is important; without specifying `mode = "wb"`, the file won't download correctly (I only know this because I received errors without this option).

We're going to do one other thing, which is to tell R that we only want to download the file if it doesn't already exist in your working directory. You'll find that it can be time consuming to download data from the web

every time you run your R script. Notice that the R code below uses the `if()` function: `if (!file.exists() download.file()`. That is, if the file does not exist, then download it.

```
## Enter the URL and extract the file name
url.df <- "https://bit.ly/2QwtpUo"
file.df <- "EU-referendum-result-data.csv"

## Only download the file if it doesn't exist in the working directory
if (!file.exists(file.df))
    download.file(url = url.df, destfile = file.df)

## Load the brexit data
brexit <- read.csv(file.df)
head(brexit)
```

Ranking UK Constituencies by 'Leave' Vote Share

Now we're ready to examine the proportion of Leave voters, and it might make sense to sort these from highest to lowest. To do this, we'll use the `order()` function, which by default sorts the data from lowest to highest. By adding the minus sign, we can reverse that ordering from highest to lowest. By storing that ordering called `rank` as an object, we can sort the data by row using brackets. What do you see?

```
## Order Leave vote share from highest to lowest
rank <- order(-brexit$Pct_Leave)  # Create vector of rankings (note the minus sign!)
brexit.rank <- brexit[rank, ]  # Sort data by Leave rank, highest to lowest
head(subset(brexit.rank, select = c("Area", "Pct_Leave")), n = 10)  # Top 10 Leave areas
tail(subset(brexit.rank, select = c("Area", "Pct_Leave")), n = 10)  # Bottom 10 Leave areas
```

Next, we're going to plot this Leave ranking in a nice interactive figure using the `googleVis` package. You'll notice that R will open a new window on your Internet browser to display the results. Try hovering over a bar and see what happens. Pretty cool, right?

```
## Create the interactive figure
bar.plot <- gvisBarChart(brexit.rank, xvar = "Area", yvar = "Pct_Leave")
plot(bar.plot)
```

The base plot is ok, but we can significantly improve it by adding a few options (which were decided upon by reading the help files and a little trial and error). Now what do you think?

```
## Create the interactive figure
bar.plot.2 <- gvisBarChart(brexit.rank, xvar = "Area", yvar = "Pct_Leave",
                           options = list(legend = "none",
                                          vAxes = "[{textStyle:{fontSize: '16'}}]",
                                          chartArea = "{left:250,top:10,bottom:10}",
                                          width= 800, height = 10000) )
plot(bar.plot.2)
```

Creating the Brexit Map

To make a map, we first need to draw the boundaries so that we can overlay the points or shading for our spatial data analysis. R has libraries for many of these maps; however, not all maps are available. Sometimes it's necessary to load what's call a *shapefile*, which contains the longitude and latitude for all of the borders of the map we want to draw. This is what we'll need to use to draw the EU Referendum constituencies (excluding Gibraltar and Northern Ireland).

The shapefile is contained in a compressed .zip format and can be downloaded directly from Github (the file is originally from Ordinance Survey). Make sure that the file is downloaded to your working directory or R

will be unable to import it.

We're also going to be using the `read_sf()` function from the `sf` package to load the unzipped file into R. The code below will automate this process (download the file, unzip it, and import the .shp into R), but you can do this manually if you encounter any errors.

```
## Download location and file name
url.shp <- "https://github.com/tkhartman/mapping-brexit/raw/master/eu_ref_esri.zip"
file.zip <- "eu_ref_esri.zip"

## Download shapefile if it's not present
if (!file.exists(file.zip))
    download.file(url = url.shp, destfile = file.zip, mode = "wb")

## Unzip the file
unzip(file.zip, exdir = "shp")

## Once unzipped, import the .shp file into R
## dsn = folder name; layer = file names begin with...
uk.shp <- read_sf(dsn ="shp", layer = "district_borough_unitary_region")
```

We're almost ready to create the map, but we need to fix a few things. For instance, we need to remove Gibraltar because it's too far away from the rest of the UK to be mapped properly. And we'll remove Northern Ireland because it's not in the UK shapefile we downloaded from Ordinance Survey.

```
## Convert factor variable to string for easy recoding later
brexit$Area_Code <- as.character(brexit$Area_Code)

## Remove Gibraltar and Northern Ireland
brexit2 <- brexit[!(brexit$Area == "Gibraltar" | brexit$Area == "Northern Ireland"), ]
```

Next, we need to prepare the data for merging, which means that we need to verify that there's a common identifier in both datasets so R knows how to combine the data. The identifiers must be unique to each observation and match perfectly; otherwise, the merging process won't work.

```
## Check that 'Area_Code in brexit data matches 'CODE' in shapefile
table(brexit2$Area_Code %in% uk.shp$CODE)

## Print any mismathes
uk.shp$CODE[!uk.shp$CODE %in% brexit2$Area_Code]  # By Area Code
uk.shp$NAME[!uk.shp$CODE %in% brexit2$Area_Code]  # By Name
```

Hmm. It turns out that there are still two mismatched identifiers in datasets, so we'll need to correct this before we can merge the data. The results above tell us that 1) Perth and Kinross and 2) Fife have Area Codes that differ in the two datasets. We could either drop these cases entirely and risk making an incomplete map or change the identifiers in the brexit data to match the ones in the shapefile. It's probably better in this case to do the latter. So, let's see if we can fix these two inconsistencies.

```
## Area code for Perth and Kinross in both datasets
brexit2$Area_Code[brexit2$Area == "Perth and Kinross"]  # Id in the brexit data
uk.shp$CODE[uk.shp$NAME == "Perth and Kinross"]  # Id in the shapefile

## Change id in brexit dataframe to match the shapefile
brexit2$Area_Code[brexit2$Area == "Perth and Kinross"] <- "S12000048"

## Area code for Fife in both datasets
brexit2$Area_Code[brexit2$Area == "Fife"]  # Id in the brexit data
```

```
uk.shp$CODE[uk.shp$NAME == "Fife"]  # Id in the shapefile

## Change id in brexit dataframe to match the shapefile
brexit2$Area_Code[brexit2$Area == "Fife"] <- "S12000047"
```

Now let's merge the Electoral Commission data (brexit2) with our shapefile (uk.shp). That's fairly straightforward using the `merge()` function provided we have a common identifier in each dataframe – in this case, we'll use the alphaumeric Area Code. Currently, this common identifier is labelled differently in each dataset – it's called 'Area_Code' in the `brexit2` dataframe and 'CODE' in the `uk.shp` shapefile. So we'll need to change the labels in our `brexit2` dataframe to match the shapefile.

```
## Only keep variables needed for mapping
brexit.sub <- subset(brexit2, select = c("Area_Code", "Pct_Leave"))

## Change names to match shapefile for merging
names(brexit.sub) <- c("CODE", "Pct_Leave")

uk.df <- merge(brexit.sub, uk.shp, by = "CODE")
```

Now we're ready to create the basic map using `ggplot`. It's a powerful package for data visualization written by Hadley Wickham. The plot takes some time to complete, so be patient! Or as I have done below, save gthe plot as a .pdf to avoid long wait times (it will be saved as a .pdf file in your working directory; if it appears blank, try opening it with Adobe Reader). We use the `ggplot()` function and input the dataframe `uk.df`. Next we use the `geom_sf()` function to add the choropleth to the raw boundary shapefile (e.g., `fill = Pct_Leave`).

```
## Mapping Brexit vote share using ggplot
brexit.map <- ggplot(uk.df) +
    geom_sf(data = uk.df,
            aes(geometry = geometry,
                fill = Pct_Leave))
brexit.map
ggsave("brexit_map.pdf", brexit.map)  # Export to .pdf to save time!
```

Again, we can make this look nicer by tweaking the options for the `ggplot()` function. Most of this invovles adding labels, but we're also going to change the colour scheme so that Leave areas are identified in shades of red, while Remain areas are blue. How might you change or improve the map? Could you add the shapefile for Northern Ireland? Or add Gibraltar to the plot? Or change the colour scheme?

```
## Mapping Brexit vote share again using ggplot
brexit.map2 <- ggplot(uk.df) +
    geom_sf(data = uk.df,
            aes(geometry = geometry,
                fill = Pct_Leave)) +
  ## Add other options to make the map look nicer
    scale_fill_distiller(palette = "RdBu") +  # Choose colours
    ggtitle("Brexit Map: 2016 EU Referendum Vote Share") +  # Title
    labs(fill = "Leave (%)") +  # Legend label
    theme(plot.title = element_text(lineheight = .8, face = "bold"),  # Bold title
          axis.title.x = element_blank(),  # Remove x-axis title
          axis.text.x = element_blank(),  # Remove x-axis labels
          axis.title.y = element_blank(),  # Remove y-axis title
          axis.text.y = element_blank(),  # Remove y-axis labels
          axis.ticks = element_blank())  # Remove axis tick marks

ggsave("brexit_map2.pdf", brexit.map2)  # Export to .pdf to save time!
```