Economic Research Experience for Undergraduates

Basic Programming Principles and Data Analysis (Part 2)

Winnie van Dijk

University of Chicago / Becker Friedman Institute

July 6th, 2017

Outline I

Administration I

- You can contact me at `wlvandijk@uchicago.edu`.

- Today's workshop:
  - Part 1 (morning): Introduction to R and Matlab.
  - Part 2 (afternoon): Selected topics for data analysis in R.

- Part 2 learning objectives:
  - Useful data sources: IPUMS, Web APIs
  - Importing data: Census API, `foreign` package
  - Shaping data: `data.tables`
  - Describing data: `ggplot2` and `ggmap` packages
  - Quick overview of methods for reporting on your results (R Notebooks, knitr, Shiny)
  - Some useful packages, why and how to build your own package

- Example code and slides used today can be found at
  `http://sites.google.com/site/reu2017vandijk/`.

Administration II

- Required software:

  R and RStudio Can be downloaded free of charge from `http://www.rstudio.com/`.

- I will show you example code, switching between slides and RStudio. We will go slowly, so that you can follow along on your own laptops.

- Afterwards, I will distribute these slides, along with example code and practice problems that you can try at home.

- There's a list of additional resources at the end of this slide deck, in case you want to learn more.

- If you have suggestions on how to improve this workshop, you can always email me. Your feedback is highly appreciated!

IPUMS I

- IPUMS = Integrated Public Use Microdata Series.
- IPUMS description: *"provides census and survey data from around the world integrated across time and space. IPUMS integration and documentation makes it easy to study change, conduct comparative research, merge information across data types, and analyze individuals within family and community context. Data and services available free of charge."*
- It's an easy way to obtain survey data from the Census, CPS, ATUS, ACS, and more.
- You can access IPUMS here: `https://www.ipums.org/`.

IPUMS II

To use IPUMS:

1. Go to the IPUMS website. You will see an overview of all the surveys they offer.
2. Select the data source you're interested in. Let's say you want to use IPUMS USA (Census microdata).
3. At the top of the page you will see the option to log in, click that.
4. On the Login page you will see a button to create new account. Go ahead and do so.
5. After you create an account you can get data here: `https://usa.ipums.org/usa-action/variables/group`. Click "select samples" to pick the years of data you want, and click the variables you want from the drop-down menus or using the search option.
6. When you are done, checkout your datacart! (To make your life easier remember to ask IPUMS to give you a `.csv` file during checkout)

IPUMS III

Advantages of using IPUMS:

- Harmonized variables! Surveys may slightly change format over the years (e.g., questions asked or spatial definitions in the ACS), and so you would have to clean your data to make everything fit together. IPUMS does this for you.
- Easy to select variables and samples, automatically generated data documentation.
- Easy to update your data set by editing past sample selection settings associated with your account.
- Great support.
- IPUMS includes:
  - CPS (Current Population Survey) microdata
  - NHGIS: Census and ACS data combined with with GIS-compatible boundary files
  - American Time Use Survey
  - Census microdata from 81 other countries
  - Lots of other stuff to explore

IPUMS IV

Exercise 2.1 (Using IPUMS/NHGIS)

(a) Use the NHGIS section of IPUMS to find county-level per-capita income data from the 2013 ACS. The variable you are looking for is `B19301. Per Capita Income in the Past 12 Months (in 2013 Inflation-Adjusted Dollars).`

(b) Add this variable to your data cart and download it in .csv format.

(c) Save the file in a convenient folder.

If this didn't work, you can download the file `NHGIS0009_DS199_2013_COUNTY.csv` from http://sites.google.com/site/reu2017vandijk/data. The file `NHGIS0009_DS199_2013_COUNTY_CODEBOOK.txt` is the codebook that should have come with your NHGIS download.

Importing saved data files

- For "simple" rectangular files like `.csv` and `.txt`, use the `readr` package. This reads in data much faster than base R.
- If you have foreign files like `.dta` files (from Stata) or `.sav` files (from SPSS), you can use the `haven` package, which can handle up to Stata 14.
- The `readxl` package is useful for reading in Excel files.
- For large files, the function `fread` from the `data.table` package can be useful.
- For previously saved R objects (with extension `.RData`), use the base function `load()`.

Exercise 2.2 (ACS data)

Set your working directory to the folder where you previously saved the ACS data, and read it into R. Save the object R creates under the name `county_data`.

- To use a package, you need to first install and then load it:

```
install.packages(<package name>)
library(<package name>)
```

Web APIs I

- API = application programming interface.
- There are packages available that let you communicate with an API to import data.
  - Such a package provides methods that allow you to specify a URL, and the information in that location is automatically interpreted by R.
- For example:
  - The `rtimes` package lets you import data from a number of different APIs maintained by the New York Times: https://cran.r-project.org/web/packages/rtimes/vignettes/rtimes_vignette.html
  - The `OECD` can be used to import data from the OECD API: https://cran.r-project.org/web/packages/OECD/OECD.pdf
  - OpenStreetMap and Google Maps API (`RgoogleMaps`)
  - The `RFacebook` packages allows you to analyze your own or public Facebook posts
  - `UbeR` to analyze your own Uber data
- In the next example we'll use the Census API to obtain the county-level income data from the ACS.

Web APIs II

Exercise 2.3 (The Census API)

(a) Go to `http://api.census.gov/data/key_signup.html` and request a US Census API key. You will get an email with the key, which you need to activate.

(b) Install and load the `acs`, `choroplethr`, `choroplethrMaps`, and `ggplot2` packages (make sure you have the more recent version of ggplot2). Then type `api.key.install("<your key>")` into the command line, copy-pasting your personal key.

(c) Obtain 2013 county-level demographics and save them in a dataframe by typing `mydata <- get_county_demographics(2013)`. To get a sense of what the data looks like, you can view the first entries of the dataframe with the command `head(mydata)`.

(d) Save a new dataframe object that conforms to the requirements of the plotting command in the next step: `df <- data.frame(region=mydata$region,value=mydata$per_capita_income)`.

(e) Plot: `county_choropleth(df, title="County-level per capita income, 2013")`.

If using the API didn't work, you can download the file `county_data.csv` from `http://sites.google.com/site/reu2017vandijk/data`.

Web APIs III

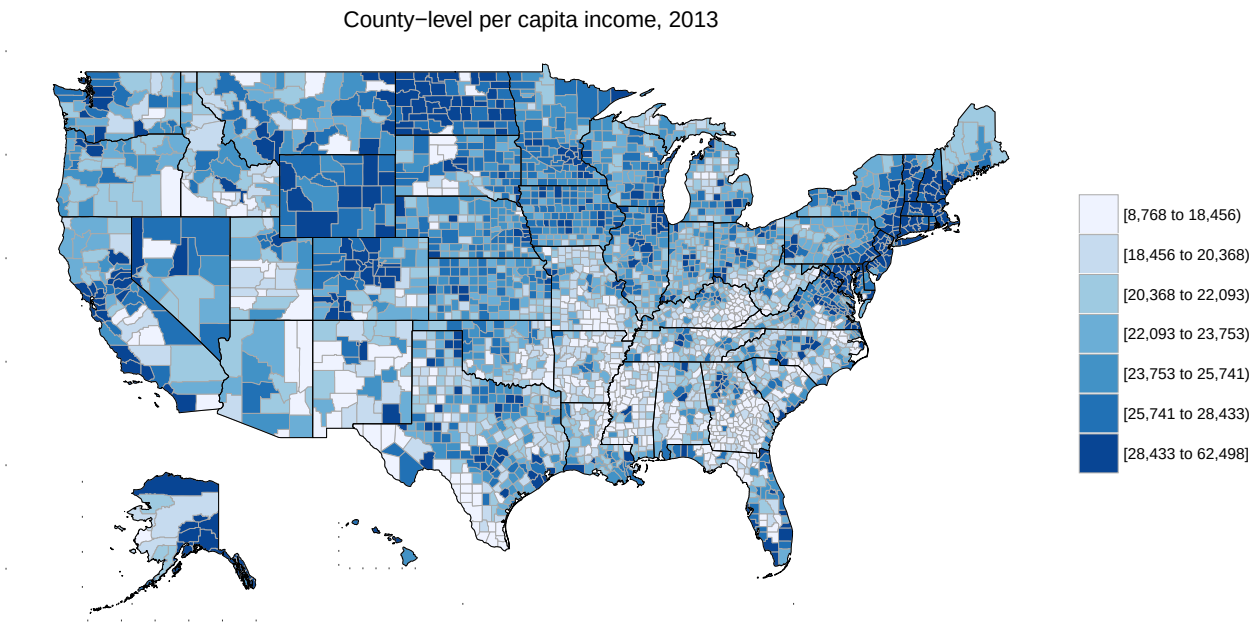


Figure 2.1: County per capita income.

Object types for storing data sets

- Two object types for storing data sets in R are `data.frame` and `data.table`.
- They come with their own syntax for manipulating data: `dplyr` is great for `data.frame`, and `data.table` has its own syntax.
- `data.table` is better for big data. Because of this, if you are new to R, you should probably focus on learning `data.table`.

data.frame and dplyr

Exercise 2.4 (ChickWeight, data.frame)

(a) Download and attach the `datasets` package. The `ChickWeight` data frame has data from an experiment on the effect of diet on early growth of chicks. To load it as a `data.frame`: `df <- ChickWeight`.

(b) Inspect the structure of the data set, for example using the command `head()`, or the command `describe()` in the `Hmisc` package.

(c) Load the `dplyr` package. What are the results of the following commands?

(i) `df$Diet`

(ii) `df[df$Diet == 1, ]`

(iii) `df$Chick[df$Diet == 1]`

(iv)
```
df2 <- df %>% group_by(Chick) %>% summarise(weight_gain = max(weight)
- min(weight), diet = Diet[1])
```

(v)
```
df2 %>% group_by(diet) %>% summarise(mean_weight_gain =
mean(weight_gain), sd_weight_gain = sd(weight_gain))
```

data.table

Exercise 2.5 (ChickWeight, data.table)

(a) Download and attach the `datasets` package. The `ChickWeight` data frame has data from an experiment on the effect of diet on early growth of chicks. Create a `data.table`: `dt <- data.table(ChickWeight)`.

(b) Inspect the structure of the data set, for example using the command `head()`, or the command `describe()` in the `Hmisc` package.

(c) What are the results of the following commands?

(i) `dt[Diet == 1, ]`

(ii) `dt[Diet == 1, list(Time,Chick)]`

(iii) `dt[Diet == 1, .(Time,Chick)]`

(iv) `dt[ , .(.N), by = Diet]` ← you'll need this command again soon

(v)
```
dt2 <- dt[, list(weight_gain = max(weight) - min(weight), diet =
Diet[1]), by = Chick]
```

(vi)
```
dt2[, list(mean_weight_gain = mean(weight_gain), sd_weight_gain =
sd(weight_gain)), by=list(diet)]
```

(d) What is the variation in weight on day 2 of the study?

The ggplot2 package I

Exercise 2.6 (The Daily Show)

In this exercise we'll reproduce the graph in this article: https://fivethirtyeight.com/datalab/every-guest-jon-stewart-ever-had-on-the-daily-show/.

(a) FiveThirtyEight has a package that allows us to import the data: `fivethirtyeight`. We will also use the `ggthemes` package to recreate the style. Please install these two packages.

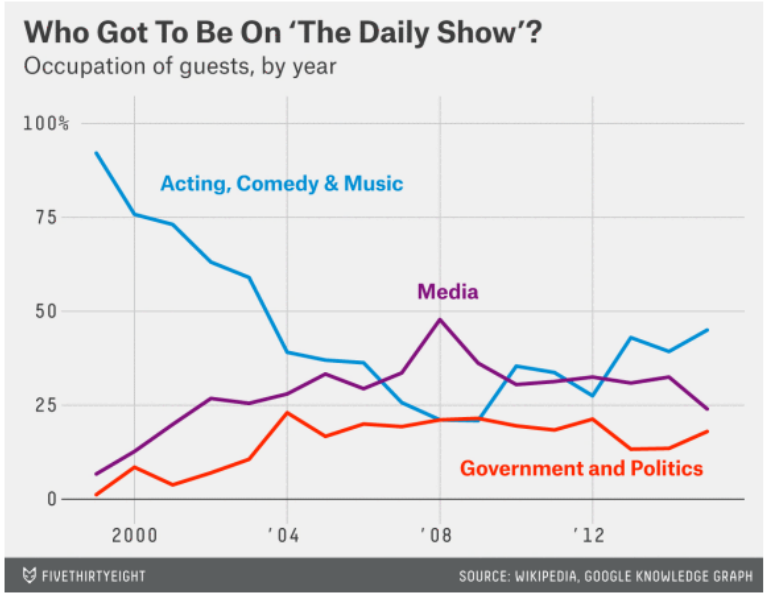


Figure 2.2: Taken from the August 6, 2015 FiveThirtyEight article "Every Guest Jon Stewart Ever Had On The Daily Show".

The ggplot2 package II

Exercise 2.6, continued (The Daily Show)

(b) Create a `data.table` object '`dt`' from the `daily_show_guests` data set.

(c) Remove rows that have NAs in the `group` field.

(d) Create a new `data.table` object '`dt1`' that collapses `dt` to have a (year, group) combination in every row, and a count of the number of appearances that belong to that category. Name that new variable `total_appearances`.

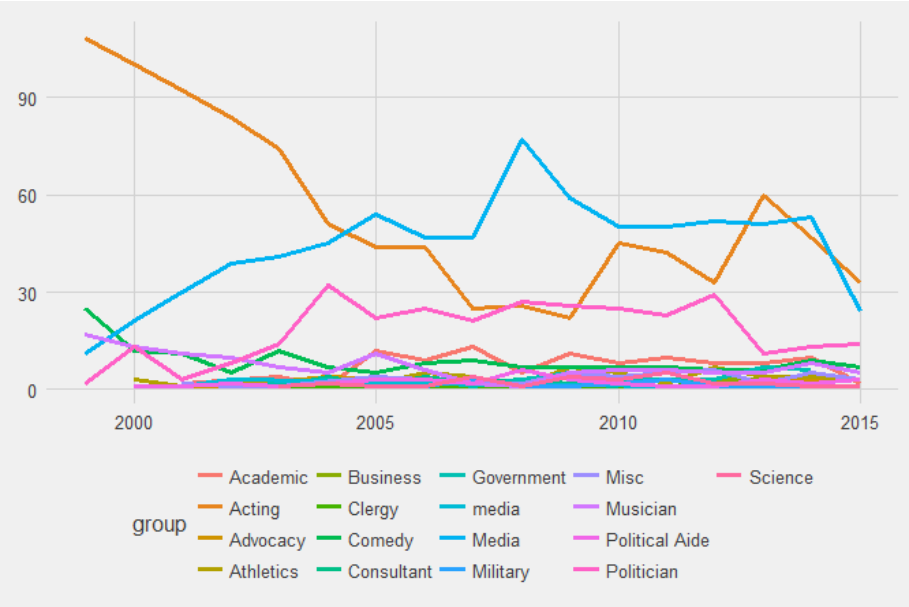
(e) Plot time trends in total number of appearances for all groups:

```
ggplot(dt1, aes(x=year, y=total_appearances, colour=group)) + geom_line()
+ theme_fivethirtyeight()
```

The ggplot2 package III

All commands from this example:

```
dt <- data.table(daily_show_guests)

dt <- dt[!is.na(dt$group),]

dt1 <- dt[, total_appearances := .(.N), .(year,group)]

ggplot(dt1, aes(x=year, y=total_appearances, colour=group)) + geom_line() +
theme_fivethirtyeight()
```



The ggmap package I

- We'll walk through an example of how to make maps with `ggmap`.
- Working with geospatial data is slightly more complicated than working with the data sets stored in .csv files that you are probably used to:
  - You need special data files that can tell R how to produce maps for the area you're interested in.
  - These files are called **shape files**.
- You can get shape files from the Census, or open data portals like the one provided by the city of Chicago: `https://data.cityofchicago.org/`.
- When you download shape files (e.g. myworld.shp), you will need to make sure you also have the DBF, Projection, and SHX files (e.g. myworld.dbf and myworld.prj and mywold.shx).
  - Make sure all the files have the same name (except for the extension) and are in the same directory.

The ggmap package II

Exercise 2.7 (Chicago grocery stores)

We will plot the locations of Chicago grocery stores, and the community areas of Chicago.

(a) Load the following packages (remember you need to install them first if you have never used them before):

```
library(rgdal) # used to load the shapefiles
library(ggmap) # used to plot at the end
library(sp) # should have been loaded as a required package of rgdal
```

(b) Download data on the location of grocery stores here: https://data.cityofchicago.org/Community-Economic-Development/Grocery-Stores-2013/53t8-wyrc/data. This is called "point data". It pinpoints the lat and long of all grocery stores. (You may learn better if you don't replicate my analysis exactly, maybe download Chicago crime or other data besides grocery stores.)

(c) Download shape files on the division of community areas in Chicago here: https://data.cityofchicago.org/api/geospatial/cauq-8yn6?method=export&format=Shapefile. This is the cookie cutter file that has the information to produce outlines of community areas. You will see that this is a zip file that contains all of the files outlined above. Extract all files in the zip file and save in the same folder as the grocery store data.

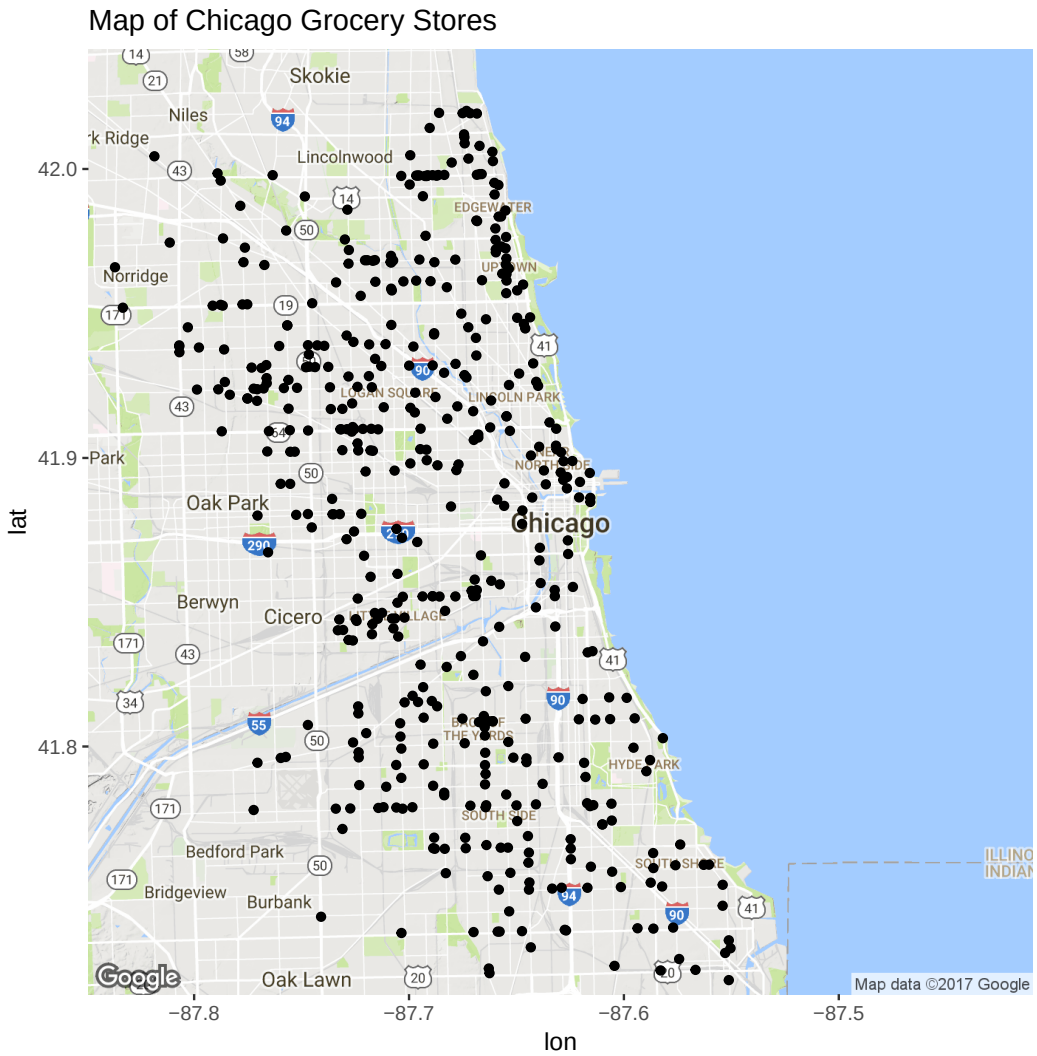
The ggmap package III

Exercise 2.7, continued (Chicago grocery stores, cont'd)

(d) The point data we downloaded from the Chicago data portal is a .csv file. We can use ggmap to import a map (instead of supplying a shape file), and show the points representing the locations of grocery stores, using ggmap. First, set your working directory to the folder where you saved the grocery store data. Then:

```
pointdf <- read_csv(file="Grocery_Stores_-_2013.csv") # import data
chicago <- get_map(location = "chicago", zoom = 11) # import map
ggmap(chicago) +
geom_point(data = pointdf, aes(x = LONGITUDE, y = LATITUDE)) +
ggtitle("Map of Chicago Grocery Stores")
```

The ggmap package IV



The ggmap package V

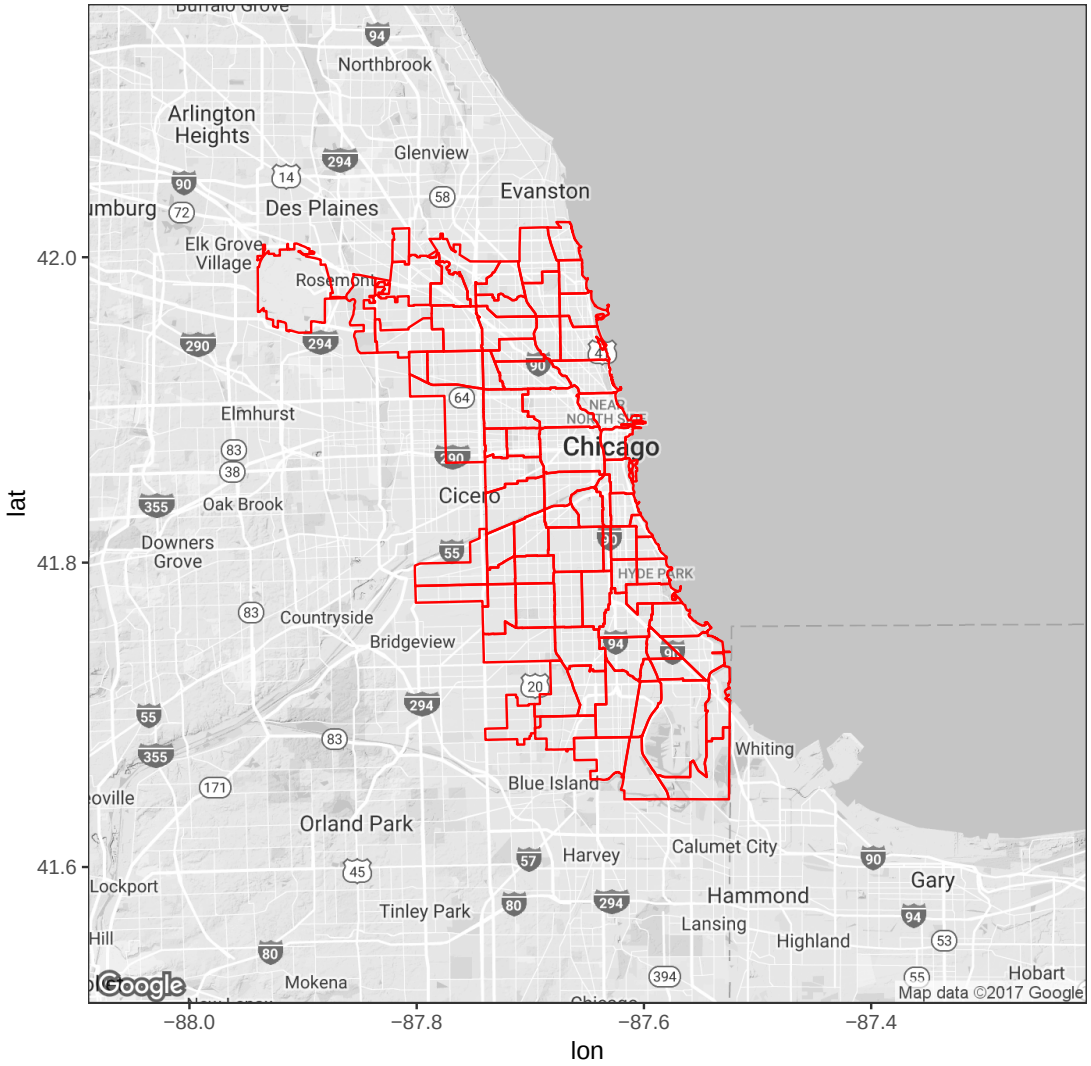
Exercise 2.7, continued (ggmap, cont'd)

(e) Set your working directory to the folder that holds the shape files. Then, plot the polygon file:

```
setwd("~/Desktop/BoundariesCommunityAreas")
overlay <- readOGR(".","geo_export_1490575f-394f-47c5-87b9-54097ef1b26f")
# your file may have a different name
overlay <- spTransform(overlay, CRS("+proj=longlat +datum=WGS84"))
overlay <- fortify(overlay)
location <- unlist(geocode('4135 S Morgan St, Chicago, IL 60609')) +
c(0,.02)
gmap <- get_map(location=location, zoom = 10, maptype = "terrain", source
= "google", col="bw")
gg <- ggmap(gmap) + geom_polygon(data=overlay, aes(x=long, y=lat,
group=group), color="red", alpha=0)
gg
```

[Example based on https://stackoverflow.com/questions/26344510/overlaying-polygons-on-ggplot-map.]

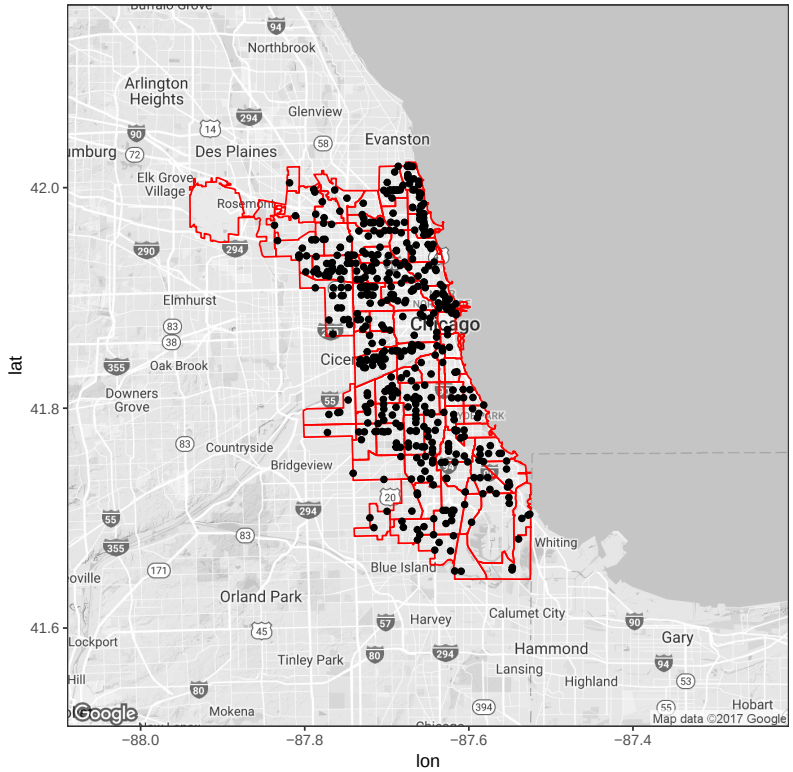
The ggmap package VI



The ggmap package VII

Exercise 2.7, continued (ggmap, cont'd)

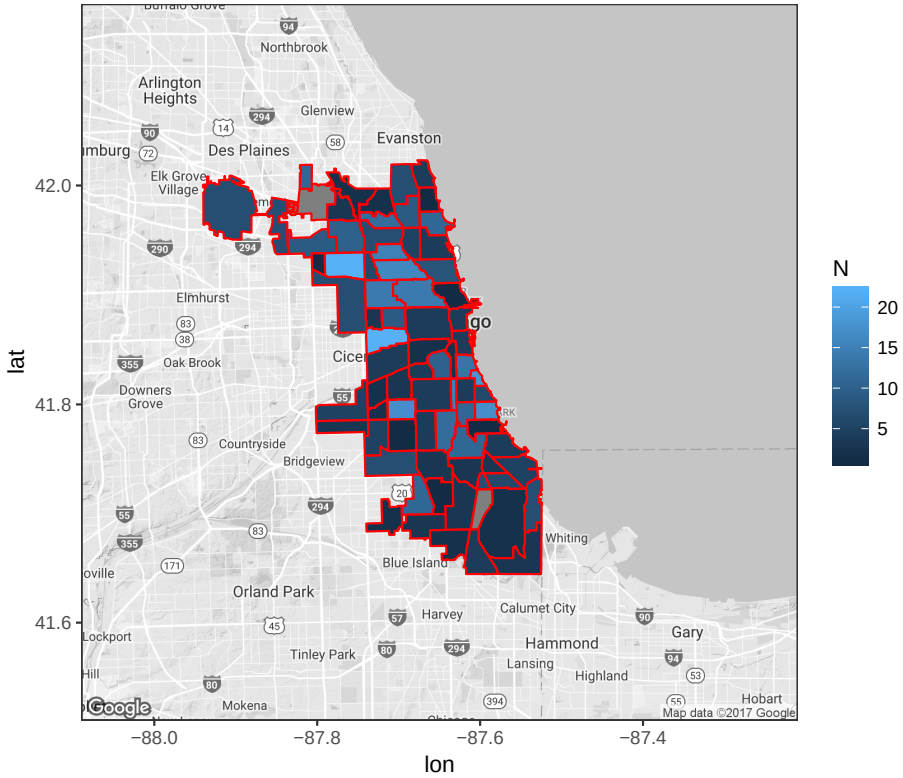
(f) Create a map with community areas as in (e), overlaying the locations of grocery stores as in (d).



The ggmap package VIII

Exercise 2.7, continued (ggmap, cont'd)

(g) Create a map with community areas as in (e), but with the areas shaded according to the number of grocery stores in that area.



Code used in part (g):

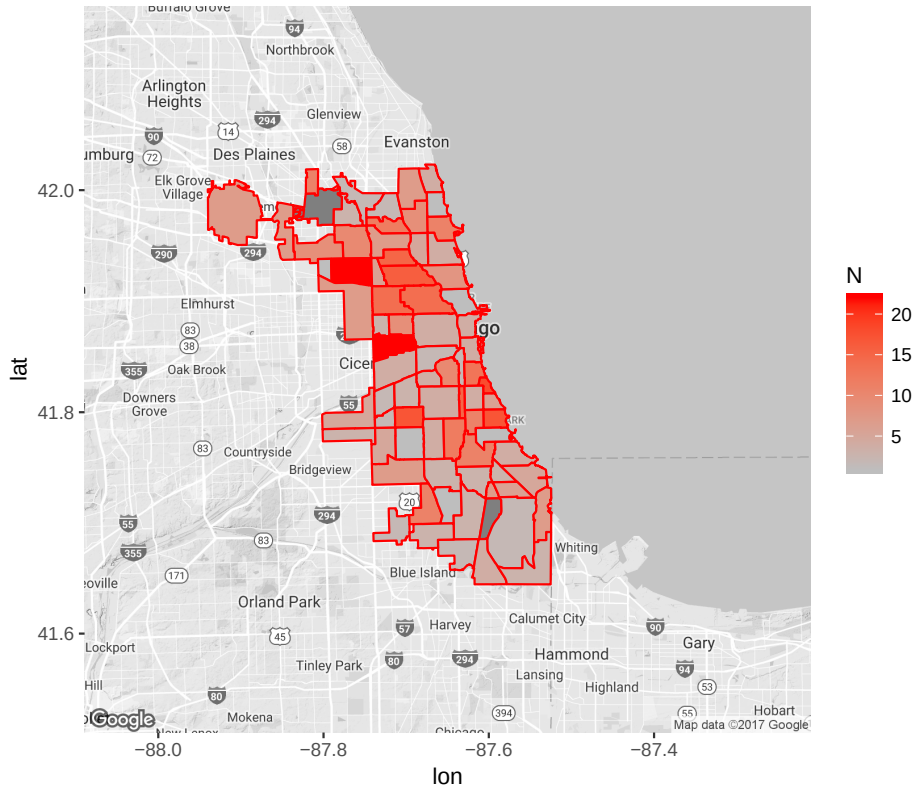
The ggmap package IX

```
pointdt <- data.table(pointdf) # convert to data.table for convenience

ca_storecount <- pointdt[, .(.N), by = `COMMUNITY AREA`] # count number of stores per community area

overlay <- readOGR(".","geo_export_a3feb55a-afb1-4aca-839e-6882c81c8c08")  your file may have a different
name

overlaydata <- data.table(overlay@data)  extract the data part

overlaydata <- merge(overlaydata, ca_storecount, by.x = "area_numbe", by.y = "COMMUNITY AREA", all.x = TRUE,
all.y = FALSE) # merge in the store counts

overlaydata$id <- sapply(slot(overlay, "polygons"), function(x) slot(x, "ID")) # add ids that correspond
to the slot ids in overlay

overlay <- data.table(fortify(overlay))

overlay <- merge(overlay, overlaydata, by = 'id') # merge the data back in

gmap <- get_map(location = location, zoom = 10, maptype = "terrain", source = "google", col = "bw")

gg <- ggmap(gmap) + geom_polygon(data = overlay, aes(x = long, y = lat, group = group, fill = N), color =
"red", alpha = 1)
```

The ggmap package X

Exercise 2.7, continued (ggmap, cont'd)

(h) Change the color scheme: `gg + scale_fill_gradient(low = 'grey', high = 'red')`.



There are several options available for reporting on your R output, including:

- R Notebooks
- Shiny
- Knitr

Some useful packages

A couple of packages you may find useful:

- `AER`: Data sets used for examples in several textbooks
- `psidR`: Creating data sets from PSID data
- `Rcpp`: Integrate C++ code to speed up computation
- `RColorBrewer`: Make custom color schemes for plotting
- `parallel`: Parallel processing in R, will help you speed up computation
- `zoo`: for working with time series
- `wru`: *"Who are You? Bayesian prediction of racial category using surname and geolocation"*

You can also build your own packages, to bundle functions and documentation built to perform specific tasks, or just to collect your own auxiliary functions that you often use. Here are some guidelines: `http://r-pkgs.had.co.nz/`.

Recommended resources I

R:

W.N. Venables, D.M. Smith and the R Core Team - An Introduction to R

2017 version

`http://cran.r-project.org/doc/manuals/R-intro.pdf`

RStudio: Tips for learning R

`http://www.rstudio.com/online-learning/`

Google's R Style Guide

`https://google.github.io/styleguide/Rguide.xml`

TryR

Online interactive tutorial for first steps in R

`http://tryr.codeschool.com/`

Stack Overflow, Cross Validated

Forums for programming and statistics questions

`http://stackoverflow.com/` and `http://stats.stackexchange.com/`

Recommended resources II

Google Developers' Intro to R

Youtube tutorial videos – geared towards Mac users

`http://www.youtube.com/playlist?list=PLOU2XLYxmsIK9qQfztXeybpHvru-TrqAP`

Hadley Wickham - Advanced R

Companion website for Advanced R book

`http://adv-r.had.co.nz/`

Garrett Grolemund & Hadley Wickham - R for Data Science

Companion website for R for Data Science book

`http://r4ds.had.co.nz/`

Tom Short - R Reference Card

A useful cheat sheet, though somewhat outdated

`http://cran.r-project.org/doc/contrib/Short-refcard.pdf`

Recommended resources III

Knitr page, by Yihui Xie

Instructions for installing and using Knitr, a tool for integrating R code into LaTeX documents

`http://yihui.name/knitr/`