MPP-E1180 Lecture 6: Automatic Data Gathering + Cleaning

Christopher Gandrud

7 October 2016

Objectives for the lesson

- Assignment info
- Review
- Benefits and challenges of open public data
- Automatic Data Gathering
- Tidying, Cleaning, and Merging data

Assignment 1 General Times for Source Code Files

- Avoid including `?FUNCTION`, `search`, `View` and similar in your source files.
- Avoid straight `install.packages` calls.
- Remember to set the working directory!
- "dofile.R" is redundant. Use file names to indicate contents/position within file hierarchy.
- Use human readable labels on your plots.

Working directory tip

You can have R automatically switch the working directory using `set_valid_wd` from the *repmis* package.

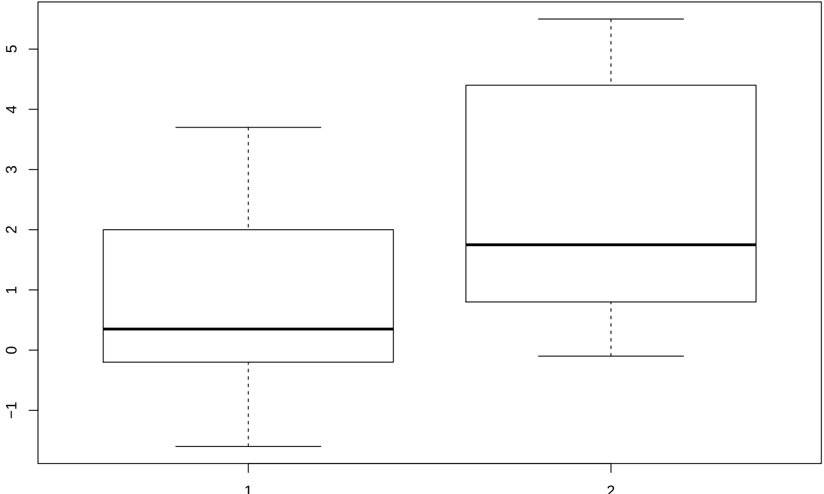
```r
# Create list of commonly used working directories
possible_dir <- c('/git_repos/Assignment1', 'C:\class\Assig

# Set to first valid directory in the possible_dir vector
repmis::set_valid_wd(possible_dir)
```

Splitting tip

You can split a vector by values of another variable using `split`.

```
library(dplyr)
split(sleep$extra, sleep$group) %>% boxplot()
```



Serving HTML file tip

Serve an HTML file hosted on GitHub by putting its *raw* version through `https://rawgit.com/`.

Assignment 2

Proposal for your Collaborative Research Project.

Deadline: 28 October

Submit: A (max) 2,000 word proposal created with **R Markdown**. The proposal will:

- Be written in R Markdown.
- State your research question. And justify why it is interesting.
- Provide a basic literature review (properly cited with BibTeX).
- Identify data sources and appropriate research methodologies for answering your question.

As always, submit the entire GitHub repo.

Review

- Why is literate programming useful for reproducible research?
- What is a markup language?
- What is a code chunk?
- What is BibTeX?
- How is it an example of working hard so you can be lazy?
- What is caching?

What is open public data?

Janssen, Charalabidis, and Zuiderwijk (2012, 258):

"Non-privacy-restricted and non-confidential data which is produced with public money and is made available without any restrictions on its usage or distribution."

Some benefits of open public data

- **Greater returns** on public investment in data gathering.
- Better **coordination** within government.
- Provides **policy-makers with information** needed to address complex problems.
- ''**Mends the traditional separation** between public organizations and users".

See Janssen, Charalabidis, and Zuiderwijk (2012, 261) for more.

Mending separation between public and users

- **Assumes** that government **considers outside** views and information (including opposing) views as **constructive**.
- Results in government **giving up** (some) control and more **actively interacting** with its environment.

An ideal for open public data

Not only should data be published, but potential **data users in society** should be **actively** sought for input on **improving government**.

Challenges to open data

- **Lack of technological competence** to implement open data that is **useful**.
- **Worry** by bureaucrats that open data will be used to **criticise** them.
- **No incentives** for users to access the data. **Lack of skills** needed to use and understand the data.
- Balancing individual **privacy** concerns.

See Janssen, Charalabidis, and Zuiderwijk (2012, 262–63) for more.

Recommended podcast

Max Ogden on the Request for Commits podcast discussing challenges faced working for the City of Boston with Code for America (`https://changelog.com/rfc-6/`).

Accessing data

Social science and public data is becoming **increasingly open** and **accessible**.

However, the level of **accessibility varies**:

- use restrictions
- format
- documentation
- version control

So . . .

We are only going to begin **scratching the surface** of the data access **obstacles** you are likely to encounter.

Tie your research to your data

Do as much **data gathering** and **cleaning** as possible in R scripts:

- Fully document for reproducible research.
- Can find (inevitable) mistakes.
- Easy to update when the data is updated.
- Can apply methods to other data sets.

''Easy" automatic data gathering

1. Plain-text data (e.g. CSV) stored at non-secure (http) URL, not embedded in a larger HTML marked-up website.
2. Plain-text data (e.g. CSV) stored at secure (https) URL, not embedded in a larger HTML marked-up website.
3. Data stored in a database with a well structured API (Application Programming Interface), that has a corresponding R package.

Non-Secure URL Plain-text data

Use `read.table` or `read.csv` (just a wrapper for `read.csv` with `sep = ','`).

Include the URL rather than the file path.

```r
read.table('http://SOMEDATA.csv')
```

Loading compressed plain-text data

You can download and load data files stored in compressed formats.

1. Download the compressed file into a **temporary file**.
2. Uncompress the file and pass it to `read.table`, `import`, etc.

Loading compressed plain-text data

Load data from Pemstein, Meserve, and Melton (2010) in a file called *uds_summary.csv*.

```r
# For simplicity, store the URL in an object called 'URL'.
URL <- "http://bit.ly/1jXJgDh"

# Create a temporary file called 'temp' to put the zip fil

temp <- tempfile()

# Download the compressed file into the temporary file.
download.file(URL, temp)

# Decompress the file and convert it into a data frame
UDSData <- read.csv(gzfile(temp, "uds_summary.csv"))

# Delete the temporary file.
unlink(temp)
```

Secure (https) URL Plain-text data

Use source_data from the repmis package.

Data on GitHub is stored at secure URLs. Select the **RAW** URL:

```
URL <- paste0('https://raw.githubusercontent.com/christophe
        'LegislativeViolence/master/Data/LegViolenceDescrip
main <- repmis::source_data(URL)
```

```
## Downloading data from: https://raw.githubusercontent.com

## SHA-1 hash of the downloaded data file is:
## 01cff579b689cea9ef9c98e433ce3122745cc5cb
```

Versioning and reproducible research

Data maintainers (unfortunately) often change data sets with little or no documentation.

`source_data` allows you to notice these changes by assigning each file a unique SHA1 Hash.

Each download can be checked against the Hash

```r
main <- repmis::source_data(URL,
                sha1 = '01cff579b689cea9ef9c98e433ce3122745
```

```
## Downloading data from: https://raw.githubusercontent.com

## Specified SHA-1 hash matches downloaded data file.
```

Can also use rio

You can also use the rio package (but no sha1 hashing):

```
main <- rio::import(URL)
```

Excel Files

The `source_XlsxData` function in repmis does the same thing as `source_data`, but for Excel files.

Builds on `read.xlsx` for loading locally stored Excel files.

Can also use `import` from rio.

Note: Excel data often needs **a lot of cleaning** before it is useful for statistical/graphical analyses.

Caching

`source_data` allows you to **cache** data with `cache = TRUE`.

This is useful if you are downloading a large data set/to speed up knitting.

Creates a locally stored version of the data set which is useful for reproducibility.

You can also cache data when you *knit* your R Markdown files.

Note: if you are caching a "large" data set in GitHub terms (>50 mb) consider ignoring it in a *.gitignore* file. Advanced users can store it using Git Large File Storage.

Data APIs

API = Application Programming Interface, a documented way for programs to talk to each other.

Data API = a documented way to access data from one program stored with another.

R and Data APIs

R can interact with most data APIs using the httr package.

Even easier: users have written API-specific packages to interact with particular data APIs.

World Bank Development Indications with WDI

Access the World Bank's Development Indicators with the WDI package.

Alternative Energy Use example:

```r
# Load WDI package
library(WDI)

# Per country alternative energy use as % of total energy
AltEnergy <- WDI(indicator = 'EG.USE.COMM.CL.ZS')
```

Note: The indicator ID is at the end of the indicator's URL on the World Bank site.

Financial Data with quantmod

The quantmod package allows you to access financial data from a variety of sources (e.g. Yahoo Finance, Google Finance, US Federal Reserve's FRED database).

```r
library(quantmod)

# Download Yen/USD exchange rate
YenDollar <- getSymbols(Symbols = 'DEXJPUS', src = 'FRED')
```

Other API-R packages

There are many more R packages that interact with web data APIs.

For a good beginner list see: `http://cran.r-project.org/web/views/WebTechnologies.html`

Working with non-table data

Format	R packages
Excel, Stata, SPSS, SAS	rio
JSON	jsonlite
MySQL	RMySQL, more info
couchDB	R4CouchDB

Data Cleaning

The data you need for your analysis is often **not clean**.

Perhaps **80%** of data analysis is typically spent cleaning and preparing data (Dasu and Johnson 2003).

- This doesn't include the time taken to gather the data.

To help streamline this process Wickham (2014) laid out **principles of data tidying**.

- Links the **physical structure** of a data set to its **meaning** (semantics).

Data structure

Many (not all) statistical data sets are organised into **rows** and **columns**.

Rows and columns have **no inherent meaning**.

Data structure

Person	treatmentA	treatmentB
John Smith		2
Jane Doe	16	11
Mary Johnson	3	1

Treatment	John Smith	Jane Doe	Mary Johnson
treatmentA		16	3
treatmentB	2	11	1

Data semantics

Data sets are **collections of values**.

All values are assigned to a **variable** and an **observation**.

- **Variable**: all values measuring the same attribute across units
- **Observation**: all values measured within the same unit across attributes.

Tidy data semantics + structure

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

Tidy data

Person	treatment	result
John Smith	a	
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Messy to Tidy data

First identify what your observations and variables are.

Then use R tools to convert your data into this format.

tidyr and its predecessor **reshape2** are particularly useful.

Messy to tidy data

```r
# Create messy (wide) data
messy <- data.frame(
  person = c("John Smith", "Jane Doe", "Mary Johnson"),
  a = c(NA, 16, 3),
  b = c(2, 11, 1)
)

messy
```

```
##         person  a  b
## 1   John Smith NA  2
## 2     Jane Doe 16 11
## 3 Mary Johnson  3  1
```

Messy to tidy data

```r
library(tidyr)

# Gather the data into tidy long format
tidy <- gather(messy, treatement, result, a:b)

tidy
```

```
##         person treatement result
## 1   John Smith          a     NA
## 2     Jane Doe          a     16
## 3 Mary Johnson          a      3
## 4   John Smith          b      2
## 5     Jane Doe          b     11
## 6 Mary Johnson          b      1
```

Tidy to messy data

Sometimes it is useful to reverse this operation with `spread`.

```r
messyAgain <- spread(data = tidy, key = treatement,
                     value = result)

messyAgain
```

```
##          person  a  b
## 1      Jane Doe 16 11
## 2    John Smith NA  2
## 3 Mary Johnson  3  1
```

Other issues cleaning data

Always **look at** and **poke your data**.

For example, see if:

- Missing values are designated with `NA`
- Variable classes are what you expect them to be.
- Distributions are what you expect them to be.

testdat can be useful for this, though not actively maintained.

Merging data

Once you have tidy data frames, you can merge them for analysis.

In general: **each observation** must have a **unique identifier** to merge them on.

These identifiers **must match exactly across the data frames**.

Merging data

```
tail(AltEnergy, n = 3)
```

```
##      iso2c  country EG.USE.COMM.CL.ZS year
## 1741    ZW Zimbabwe          4.858643 2007
## 1742    ZW Zimbabwe          4.666912 2006
## 1743    ZW Zimbabwe          4.396196 2005
```

```
tail(UDSData, n = 3)
```

```
##            country year cowcode      mean        sd
## 9135 Western Samoa 2006     990 0.2485397 0.2155926 0.24
## 9136 Western Samoa 2007     990 0.2439135 0.2151686 0.24
## 9137 Western Samoa 2008     990 0.2407623 0.2192563 0.24
##         pct975
## 9135 0.6648156
## 9136 0.6571918
## 9137 0.6622615
```

Create unique ID: country codes

Unique identifier will be iso 2 letter country code and **year**.

Use the countrycode package to turn UDS data's Correlates of War Country Code (`cowcode`) to `iso2c`.

```r
library(countrycode)

# Assign iso2c codes base on correlates of war codes
UDSData$iso2c <- countrycode(UDSData$cowcode, origin = 'cow
                             destination = 'iso2c', warn =
```

NOTE: Always check the data to make sure the correct codes have been applied!

Creating IDs: geocodes

countrycode clearly only works for standardising country IDs

Other packages can be useful for standardising other unit IDs.

For example, geocode from ggmap can be used to create latitude/longitudes for other geographic units:

```
places <- c('Bavaria', 'Seoul', '6 Parisier Platz, Berlin')

ggmap::geocode(places, source = 'google')
```

```
##         lon      lat
## 1  11.49789 48.79045
## 2 126.97797 37.56654
## 3  13.37854 52.51701
```

Creating IDs: Time

Time units may be important components of observation IDs.

Use the lubridate package to standardise dates.

Creating IDs: Time

```r
library(lubridate)

# Create time data
times <- c('Sep. 17 1980', 'March 23 2000', 'Nov. 3 2003')

mdy(times)
```

```
## [1] "1980-09-17" "2000-03-23" "2003-11-03"
```

Note: Times should always go from **longest to shortest** unit (e.g. YEAR-MONTH-DAY). Makes dates **sortable**.

Merge data

```
# Keep only desired variables
UDSData <- UDSData[, c('iso2c', 'year', 'median')]

names(UDSData)
```

```
## [1] "iso2c"  "year"   "median"
```

```
Combined <- merge(AltEnergy, UDSData,
                  by = c('iso2c', 'year'))

head(Combined, n = 3)
```

```
##   iso2c year country EG.USE.COMM.CL.ZS    median
## 1    AD 2005 Andorra                NA 0.7785676
## 2    AD 2006 Andorra                NA 0.7860388
## 3    AD 2007 Andorra                NA 0.7853250
```

Some merge details

By default, only observations in both data sets are kept. Use `all`, `all.x`, or `all.y` to keep non-merged observations.

Always **check your data** after a merge to see if you did what you wanted to do!

Clean up

You many want to do some post merge cleaning. For example assign new variable names:

```
names(Combined) <- c('iso2c', 'year', 'country',
                     'alt_energy_use',  'uds_median')
```

or

```
Combined <- dplyr::rename(Combined, new_year = year)
```

And reorder variables

```
Combined <- DataCombine::MoveFront(Combined, 'country')
```

Seminar: Access web-based data

Thinking of your pair research project, write an R script to download **two or more** data sets from the web.

Either in the same or a linked R script **clean** and **merge** the data.

References

Dasu, Tamraparni, and Theodore Johnson. 2003. *Exploratory Data Mining and Data Cleaning*. Hoboken, NJ: John Wiley & Sons.

Janssen, Marijn, Yannis Charalabidis, and Anneke Zuiderwijk. 2012. "Benefits, Adoption Barriers and Myths of Open Data and Open Government." *Information Systems Management* 29 (4): 258–68.

Pemstein, Daniel, Stephen A. Meserve, and James Melton. 2010. "Democratic Compromise: A Latent Variable Analysis of Ten Measures of Regime Type." *Political Analysis* 18 (4): 426–49.

Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software* 59 (10): 1–23.