Workshop 1: Introduction to Data Analysis in R for Public Policy

My goals for the course

- Learn how to perform basic data analysis tasks in R
- Learn some fundamental programming concepts that will help with learning new tools
- Learn *when* and *how* writing code is better than Excel: choosing the right tool for the job

About your instructors

Cecile:

- 2nd year MSCAPP student at Harris
- Formerly:
  - Senior Research Assistant at The Brookings Institution's Metropolitan Policy Program
  - Research Assistant at the Center on Budget and Policy Priorities (CBPP)
- Alumna of the College (2015): econ + public policy, UCIPSS scholar

You can reach me at cmmurray@uchicago.edu. You can also find me on LinkedIn or Twitter.

Lilian:

- 2nd year MSCAPP student at Harris
- Formerly:
  - Research Assistant at NORC at the University of Chicago
  - Research Assistant at Chapin Hall at the University of Chicago
- Alumna of the College (2016): econ + polisci, UCIPSS scholar, UCPIP fellow

You can reach me at lilianhj@uchicago.edu. I also have a personal website, a LinkedIn, and a Twitter.

Overview of today

1. Talk about when doing data analysis programmatically is advantageous
2. Introduce the tools we'll be using in this series
3. Start playing with data!

Key terms: script, package, working directory, environment

Question for the room: What do you want to get out of this workshop series?

Why write code?

Question for the room: Who has ever programmed before?

Advantages over Excel:

- easier to check (in some ways):
  - The flow of your analysis is reflected in your code
  - If following good coding practices, errors are likely to be systematic rather than idiosyncratic
- reproducible:

    - other people can replicate what you did = good social science
    - also, you can replicate what you did
    - repeating your analysis with updated data is very easy
- scalable:
    - you can automate many tedious steps of data cleaning
    - trying several alternative methods is more manageable
    - will work with 1M or 100K rows just as well as with 1K
- perform more sophisticated analyses (e.g. regression, machine learning)

Disadvantages:

- less accessible to broad audiences
    - other people may not be able to understand/check your work
- there is a whole alternative set of common errors
    - you spend less time looking at the values, so you may not recognize erroneous outliers
    - copy and paste danger: Excel alerts you when it thinks your formula doesn't match a pattern
- can be more time-consuming on the first pass
    - this becomes less true as you get more practice

Why use R?

- Open source (free!) but still offers nice interface and project management tools through RStudio
- Great functionality for performing many kinds of descriptive analysis and modeling
- Excellent visualization tools
- Allows for some general-purpose programming
- Vibrant, engaged user community

Table 1: Comparing common data analysis tools

	Excel	Stata	Python	R
Free?	No*	No	Yes	Yes
Nice user interface	Yes	Yes	No	Yes
Ease of data assembly	Good	Good	Good	Excellent
Ease of descriptive stats	Very good	Excellent	Good	Very good
Quality of visualization	Good	Mediocre	Good	Excellent
More complex modeling	No	Yes	Yes	Yes
General-purpose programming	No	No	Yes	Yes

[a] *But your organization probably already paid for MS Office

R and RStudio

R = the programming language

RStudio = an integrated development environment (IDE) for R, aka a nice interface

Getting to know RStudio

- different panels: script, console, environment, files, packages, help
- console vs. scripts
- Run commands by hitting enter (in the console) or highlighting and either clicking run (clunky) or hitting command-enter

R Basics

Basic mathematical operations in R

Performing basic computations works pretty much as you'd expect: you can use the normal math operators and functions.

```r
# R can do basic arithmetic
1 + 1
```

```
## [1] 2
```

```r
1 * 5
```

```
## [1] 5
```

```r
(20 / 4) * 2
```

```
## [1] 10
```

```r
2 ^ 2
```

```
## [1] 4
```

```r
2**2 # alternative way to do exponents
```

```
## [1] 4
```

```r
# this is how you save a value in a variable: we are saying x is 5
# think of this as associating a name with some object
x <- 5
x
```

```
## [1] 5
```

```r
# we can also change the value of x
x <- 7
x == 5
```

```
## [1] FALSE
```

Comparisons and logical operators

Programming languages are based on logical statements: expressions that are either true or false. Comparison operators and logical operators help us construct such statements.

Comparison operators compare two things: for example, are they equal, or is one greater than or equal to the other.

```r
# what's that double equals sign thing? it is a comparison operator.
# now we can make logical statements (either true or false)
1 == 1
```

```
## [1] TRUE
```

```r
1 == 2
```

```
## [1] FALSE
```

```r
1 != 2
```

```
## [1] TRUE
```

```r
5 >= 4
```

```
## [1] TRUE
```

```r
-1 < 0
```

```
## [1] TRUE
```

```r
# side note on computers and decimal accuracy (this happens in Excel too)
1.000000000000001 == 1
```

```
## [1] FALSE
```

```r
1.0000000000000001 == 1
```

```
## [1] TRUE
```

Logical operators allow you to combine true/false statements.

```r
# we can combine these logical statements with logical operators
# & is AND: means both are true
1 == 1 & 2 == 2
```

```
## [1] TRUE
```

```r
1 == 1 & 1 == 2
```

```
## [1] FALSE
```

```r
# | is OR: means at least one is true
1 == 1 & 1 == 2
```

```
## [1] FALSE
```

```r
1 == 2 & 2 == 3
```

```
## [1] FALSE
```

```r
# ! is NOT: means the statement is *not* true
!(1 == 1)
```

```
## [1] FALSE
```

```r
!(1 == 2)
```

```
## [1] TRUE
```

Types of objects

Basic types:

- Numeric: 1, 3.14, 2020
- Character, aka strings: ‘a’, “data”
- Logical: TRUE or FALSE

Data structures:

- Scalar: a single value (e.g. the number 5)
- Vectors: a list of values (numbers, character strings, etc)
- Matrices: a list of vectors of the same type (numeric, character, etc). You can think of a matrix like a table where each vector is a column.
- Data frame: a table, similar to a single Excel sheet, with named columns. Every row has the same number of columns and every column has the same number of rows.

In R, I most often work with scalars, vectors, and dataframes. We'll focus on these, but are some other kinds of data structures not listed above that you may encounter.

```r
5 # numeric
```

```
## [1] 5
```

```r
"a" # character/string
```

```
## [1] "a"
```

```r
'a' # doesn't matter if you use single or double quotes, still a character
```

```
## [1] "a"
```

```r
# one reason why we care about types:
# we can't add numbers if one is stored as a string (this happens a lot)
1 + 1
```

```
## [1] 2
```

```r
# '1' + 1 doesn't work!

# this is a vector: it's basically a list of values
vec1 <- c(1, 2, 3, 4)
vec1
```

```
## [1] 1 2 3 4
```

```r
# to get the second value, index in with square brackets
vec1[2]
```

```
## [1] 2
```

```r
# collection of vectors = a matrix
vec2 <- c(4, 5, 6, 7)
m1 <- matrix(c(vec1, vec2)) # default is just to make a column
m1
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
## [5,]    4
## [6,]    5
## [7,]    6
## [8,]    7
```

```r
m2 <- matrix(c(vec1, vec2), nrow = 2, ncol = 4) # specify another shape
m2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    4    6
## [2,]    2    4    5    7
```

```r
# get values out of a matrix with indexing
m2[1, 2]
```

```
## [1] 3
```

```r
m2[, 2] # access second column
```

```
## [1] 3 4
```

```
m2[1, ] # access first row
```

```
## [1] 1 3 4 6
```

```
m2[1, 1:3] # use this colon syntax if you don't want all columns/rows
```

```
## [1] 1 3 4
```

Functions:

We'll talk a lot more about functions later in the series, but for now, think of a function as a canned bit of code that performs some common operation.

To use a grammar analogy, a data structure is a noun - it's *what* you are manipulating. A function is a verb - it's *how* you are manipulating that data.

As a note on vocabulary, when people use a function, they often say they have "called" that function.

Base R and Packages: Introducing the Tidyverse

The base R language is pretty powerful on its own. But the true power of R comes from user-built packages, which are essentially a bundle of related bits of code that help automate or streamline common tasks. Often these bits of code are functions.

The most well-known group of packages for data analysis is called the tidyverse. This group of packages includes functions for the entire data analysis pipeline, from data ingestion and cleaning to analysis and visualization. They share a common design philosophy, syntax, and data structures.

Loading data

Most data is available in a csv file format or something else that a spreadsheet program like Excel can open. Loading this into R is very simple using functions from the `readr` package.

First, though, some key concepts:

- **working directory:** where on your computer R is looking to find files (scripts, data, etc.)
- **environment:** this is all the objects R is actively holding in memory for you, which you can access again by typing the object name

```
# first we need to tell R where to look for files
getwd() # tells us where we are
```

```
## [1] "/Users/cecilemurray/Documents/coding/fried-R-workshop/01-workshop"
```

```
# setwd('<your working directory here>') # sets the working directory
# R projects organize this process nicely, but aren't strictly necessary

list.files() # another way to see what's in your working directory
```

```
## [1] "01_explore_data.R"   "01_introduction.pdf" "01_introduction.Rmd"
## [4] "01_R_basics.R"       "Crimes_-_2018.csv"   "Week1_Intro.Rmd"
```

```
# if you have never used the package before, you need to install it
# install.packages("tidyverse")

# then we can load it
```

```r
library(tidyverse)

# load the data: note that if this file were saved elsewhere, this would fail
crimes <- read_csv("Crimes_-_2018.csv")
```

We can see that `crimes` is now an object in our environment. We can click on it to view it, or use `View(crimes)`.

Crimes is a data frame. Again, think of data frames like Excel spreadsheets, except that you're guaranteed your cells will have a rectangular shape (all rows have the same number of columns, all columns have the same number of rows).

One special thing about data frames is that columns have names. If we want to "index" (i.e. go to) a column, we can do that using the name of the data frame, a `$`, and the column name. Suppose we want the Date column: instead of needing to know the position of that column in the data frame, we can just use the name.

```r
# this is a way of looking at it in the console
glimpse(crimes)
```

```
## Observations: 267,687
## Variables: 22
## $ ID                     <dbl> 11323155, 11230111, 11499496, 11499495, 1147...
## $ `Case Number`          <chr> "JB273997", "JB151151", "JB487217", "JB49745...
## $ Date                   <chr> "05/11/2018 03:27:00 PM", "02/12/2018 03:00:...
## $ Block                  <chr> "016XX N TRIPP AVE", "049XX W KINZIE ST", "0...
## $ IUCR                   <chr> "1752", "0266", "2024", "1812", "1812", "026...
## $ `Primary Type`         <chr> "OFFENSE INVOLVING CHILDREN", "CRIM SEXUAL A...
## $ Description            <chr> "AGG CRIM SEX ABUSE FAM MEMBER", "PREDATORY"...
## $ `Location Description` <chr> "RESIDENCE", "RESIDENCE", "RESIDENCE", "APAR...
## $ Arrest                 <lgl> FALSE, FALSE, TRUE, TRUE, TRUE, FALSE, TRUE,...
## $ Domestic               <lgl> TRUE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE...
## $ Beat                   <chr> "2534", "1532", "2222", "0421", "0822", "161...
## $ District               <chr> "025", "015", "022", "004", "008", "016", "0...
## $ Ward                   <dbl> 26, 37, 21, 7, 14, 38, 24, 34, 46, 35, 5, 49...
## $ `Community Area`       <dbl> 23, 25, 73, 43, 63, 76, 29, 49, 3, 21, 43, 1...
## $ `FBI Code`             <chr> "17", "02", "18", "18", "18", "02", "01A", "...
## $ `X Coordinate`         <dbl> 1147801, 1143287, NA, NA, NA, NA, 1148551, N...
## $ `Y Coordinate`         <dbl> 1910610, 1902272, NA, NA, NA, NA, 1894943, N...
## $ Year                   <dbl> 2018, 2018, 2018, 2018, 2018, 2018, 2018, 20...
## $ `Updated On`           <chr> "12/16/2019 03:48:59 PM", "12/16/2019 03:48:...
## $ Latitude               <dbl> 41.91067, 41.88787, NA, NA, NA, NA, 41.86766...
## $ Longitude              <dbl> -87.73247, -87.74926, NA, NA, NA, NA, -87.73...
## $ Location               <chr> "(41.910667557, -87.732469532)", "(41.887872...
```

```r
# each column is a vector, but we can refer to them by name
# crimes$Date  # commenting this out in lecture notes to minimize printing

# two ways of indexing: with $ and with []
crimes$Date[1:10]
```

```
##  [1] "05/11/2018 03:27:00 PM" "02/12/2018 03:00:00 PM" "10/23/2018 03:38:11 PM"
##  [4] "10/31/2018 07:31:36 AM" "05/03/2018 03:07:21 PM" "01/20/2018 10:00:00 PM"
##  [7] "11/15/2018 01:37:00 AM" "02/01/2018 12:00:00 AM" "10/01/2018 09:00:00 AM"
## [10] "01/10/2018 12:00:00 PM"
```

Basic cleaning and exploration (more to come)

Notice above that some of these names have backticks surrounding them. That's because they contain spaces, which are not a valid part of names in R.

Fixing this and standardizing them will make working with the data much easier. I could do that manually, but instead I'm going to use a function from the `janitor` package.

```r
# we can see the column names like this:
names(crimes)
```

```
##  [1] "ID"                   "Case Number"          "Date"                
##  [4] "Block"                "IUCR"                 "Primary Type"        
##  [7] "Description"          "Location Description" "Arrest"              
## [10] "Domestic"             "Beat"                 "District"            
## [13] "Ward"                 "Community Area"       "FBI Code"            
## [16] "X Coordinate"         "Y Coordinate"         "Year"                
## [19] "Updated On"           "Latitude"             "Longitude"           
## [22] "Location"
```

```r
# the spaces/caps in these names will be annoying:
# crimes$Location Description  # this won't work
crimes$`Location Description`[1:10] # we need these backtick things if there are spaces
```

```
##  [1] "RESIDENCE" "RESIDENCE" "RESIDENCE" "APARTMENT" "APARTMENT" "APARTMENT"
##  [7] "STREET"    "RESIDENCE" "RESIDENCE" "APARTMENT"
```

```r
# to get rid of spaces and caps, we could type out a new vector of names and assign them
# original_names <- names(crimes)
# names(crimes) <- c("id", "case_number", "date", "block",...)

# this is tedious and error-prone so instead, let's use a function from the janitor package:
crimes <- janitor::clean_names(crimes)

# much better
names(crimes)
```

```
##  [1] "id"                   "case_number"          "date"                
##  [4] "block"                "iucr"                 "primary_type"        
##  [7] "description"          "location_description" "arrest"              
## [10] "domestic"             "beat"                 "district"            
## [13] "ward"                 "community_area"       "fbi_code"            
## [16] "x_coordinate"         "y_coordinate"         "year"                
## [19] "updated_on"           "latitude"             "longitude"           
## [22] "location"
```

Some other simple exploration using commands (functions!) available in base R (no packages):

```r
# how many crimes were reported last year?
nrow(crimes)
```

```
## [1] 267687
```

```r
# how many arrests?
sum(crimes$arrest)
```

```
## [1] 53456
```

```r
# how many kinds of crime?
length(unique(crimes$primary_type))
```

```
## [1] 32
```

```r
# count of crimes by type
table(crimes$primary_type)
```

```
##
##                             ARSON                           ASSAULT
##                               373                             20400
##                           BATTERY                          BURGLARY
##                             49805                             11744
## CONCEALED CARRY LICENSE VIOLATION               CRIM SEXUAL ASSAULT
##                               149                              1638
##                   CRIMINAL DAMAGE                 CRIMINAL TRESPASS
##                             27822                              6907
##                DECEPTIVE PRACTICE                          GAMBLING
##                             19274                               201
##                          HOMICIDE                 HUMAN TRAFFICKING
##                               591                                12
##  INTERFERENCE WITH PUBLIC OFFICER                      INTIMIDATION
##                              1306                               169
##                        KIDNAPPING               LIQUOR LAW VIOLATION
##                               171                               268
##               MOTOR VEHICLE THEFT                         NARCOTICS
##                              9984                             13436
##                      NON-CRIMINAL  NON-CRIMINAL (SUBJECT SPECIFIED)
##                                36                                 3
##                         OBSCENITY        OFFENSE INVOLVING CHILDREN
##                                85                              2270
##           OTHER NARCOTIC VIOLATION                     OTHER OFFENSE
##                                 1                             17225
##                      PROSTITUTION                   PUBLIC INDECENCY
##                               718                                14
##             PUBLIC PEACE VIOLATION                           ROBBERY
##                              1371                              9680
##                       SEX OFFENSE                          STALKING
##                              1130                               207
##                             THEFT                 WEAPONS VIOLATION
##                             65241                              5456
```

Exercise:

1. Go to data.cityofchicago.org. This is the city's open data portal. It hosts a lot of interesting datasets on topics from crime to 311 requests to rodent complaints. Find and download a dataset you think is interesting.

2. Load the data into R.

3. Explore it - try to learn 3-5 facts from summarizing the data.

Some best practices to follow: - keep your files organized in a folder system: for example, create a main project folder with one subfolder for raw data, one subfolder for R scripts, etc. - keep a record of how you downloaded the data, and keep a copy of your raw data that you do not modify - using the console is fine for exploration and debugging, but keep your successful commands in a script file - use comments to annotate your work - consider creating a final version of your script that contains only the code necessary to back up your key insights.