R Basics for Social Welfare Researchers

Paul Yousefi, MPH

April 9, 2014

Part I: Getting Started

Goals

- Introduce what R is and what it can do
- Compare it to other competing software options
- Crash course on getting started
- Provide resources for further learning

Logistics

- Does everyone have R and Rstudio installed?
- You'll also need a dataset called `CRIME.csv`
  - This data is available for download from: https://github.com/yousefi138/Rbasics.git

What is R?

- R is a general-purpose statistical software and programming language
    - Alternative to STATA, SAS, MATLAB, SPSS, etc.
- Performs a wide variety of statistical analyses
- It also has advanced extensible graphical techniques for **visualizing data** and **reporting results**

Why choose R?

- It's **free** and open source.
- Highly customizable, that is, users can easily add and share additional functions
- The code development and help communities for R are extensive, across many disciplines and are growing
- Packages of user-submitted code are a big part of the advantage of using R
  - Often methods publications will release R packages simultaneously to allow easy replication

Why choose R?

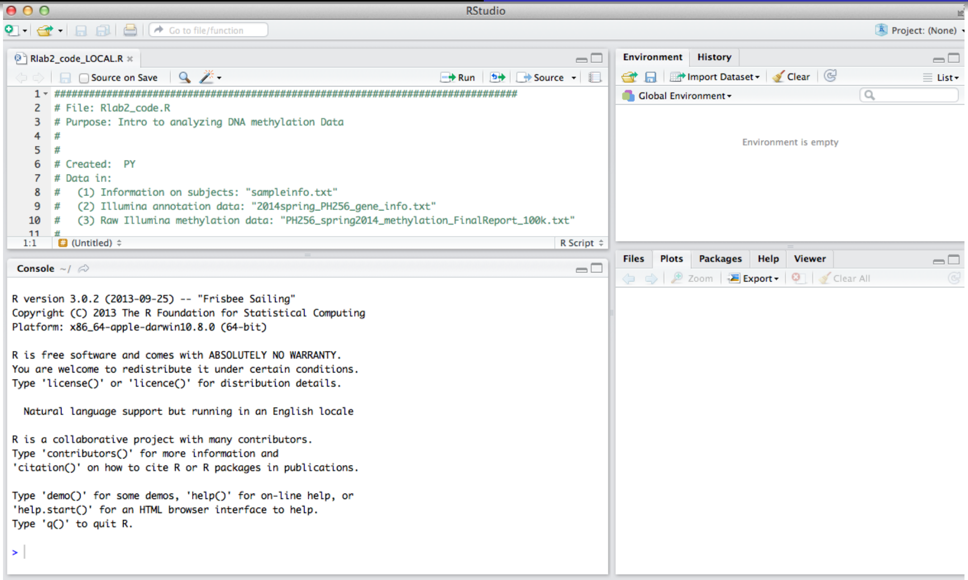
- Because R is open source, it can easily used by developers and incorporated into many different applications
  - E.g. Word-processing alternatives (like LaTeXand Markdown) can incorporate your code into journal articles, reports and presentations
- Because it's free but has the same functionality as it's competitors, it's a good place to start learning statistical programming
  - Your employer will always be able to afford the license :)
- It's quickly becoming the standard statistical computing platform across many academic disciplines

Accessing an R interface

There are several ways to run R and write R code:

- Using the GUI program on available from CRAN
- Through a unix terminal (type `R` to begin)
- Using an integrated development environment: **Rstudio**

Rstudio is a great way to begin and get a feel for the program. I suggest using this package to follow along with the workshop today.


RStudio
Go to file/function
Project: (None)
Rlab2_code_LOCAL.R
Source on Save
Run
Source
##
File: Rlab2_code.R
Purpose: Intro to analyzing DNA methylation Data
#
#
Created: PY
Data in:
(1) Information on subjects: "sampleinfo.txt"
(2) Illumina annotation data: "2014spring_PH256_gene_info.txt"
(3) Raw Illumina methylation data: "PH256_spring2014_methylation_FinalReport_100k.txt"
#
1:1
(Untitled)
R Script
Environment
History
Import Dataset
Clear
List
Global Environment
Environment is empty
Console ~/
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Files
Plots
Packages
Help
Viewer
Zoom
Export
Clear All

Submitting code

- The most important of the sub-windows in the interface is the `console`. This is where all user code is submitted and the program prints the output
- The screen just above the `console` is an embedded text-editor that allows for composing and saving user code as ".R" files. Write all of your code here to save it as text
- Send text from the editor to the `console` by highlighting code to be evaluated and typing <`command+enter`> on the keyboard or clicking the **Run** button

The `console` has a command line with a carat (>) and some info about the version of R you're running. The blinking cursor lets you know you're ready to begin

Submitting code

- At its most simple, R can work just like a simple calculator
- Try typing a calculator expression in your text editor

```
> 2+3
```

- Highlight and press <command+enter> to evaluate it in the console

Submitting code

- At its most simple, R can work just like a simple calculator
- Try typing a calculator expression in your text editor

  2+3

  ```
  > [1] 5
  ```

Data objects

- Beyond being a calculator, we can create data objects in the workspace by giving them a name
- This is done using the assign function <-
  - Text on the left side of the <- is the name of the data object
  - Whatever is on the right side is the data that gets named

So the following:

```
a <- 2
a
```

```
> [1] 2
```

assigns the object a the value of 2.

Work space

- Data objects created though assignment stay in R's active memory, called the **work space**
- To see the contents of the **work space** type the command `ls()`
  - `ls()` is R code for *list*

```
ls()
```

Work space

- Data objects created though assignment stay in R's active memory, called the **work space**
- To see the contents of the **work space** type the command `ls()`
    - `ls()` is R code for *list*

```
ls()
```

```
> [1] "a"
```

So far, we only have one data object, a

Work space

- However, if we create more data objects through assignment, they will show up in our **work space**

```
b <- log10(500)
c <- ((23-12)^2)/12 + ((17-24)^2)/24 + ((8-12)^2)/12
ls()
```

```
> [1] "a" "b" "c"
```

Work space

- The contents of any data object in the **work space** can be printed by simply submitting the name of that object

```
ls()

> [1] "a" "b" "c"

c

> [1] 13.46
```

Built in functions

- R has *many* functions built-in to perform common data operations
  - Including functions to create more complicated data objects than we've been working with

Built in functions

- All R functions have a common syntax that read from left to right:
  - function name
  - an open bracket '('
  - the arguments of the function, separated by commas
  - a closed bracket to end the function ')'

Built in functions

- For example, the c() or concatenate function makes a list of numbers called a vector
- The numbers to include in the vector are entered between the parenthesis, separated by commas as the arguments of the function

```
d <- c(1, 2, 3, 4, 5)
d
```

```
> [1] 1 2 3 4 5
```

Built in functions

- A nice feature of R is that finding help using functions is quite easy
- Two common ways to get information:
    1. `args()`
    2. `?` or `help()`

Built in functions

- So, if I wanted to learn how to use the function, `sample`, I could run the following:

```
args(sample)
```

```
> function (x, size, replace = FALSE, prob = NULL)
> NULL
```

and see the syntax of what should be submitted in the arguments for `sample`

Built in functions

- Or, by typing

```
?sample
```

I get a nice detailed help page on how to use the function sample {base} R Documentation

Random Samples and Permutations

Description

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.

Usage

```
sample(x, size, replace = FALSE, prob = NULL)

sample.int(n, size = n, replace = FALSE, prob = NULL)
```

Arguments

`x`	Either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'
`n`	a positive number, the number of items to choose from. See 'Details.'
`size`	a non-negative integer giving the number of items to choose.
`replace`	Should sampling be with replacement?
`prob`	A vector of probability weights for obtaining the elements of the vector being sampled.

Built in functions

- So far, we've made data objects that were:
  - Scalars, or single numbers, though calculator operations
  - Vectors, or lists of numbers, through the c function
  - We can also make two dimensional data objects, like matrices

Types of data objects:

Object	Description	Dimensions	Example
Scalar	a single data elements	1x1	6
Vector	a list of like data elements	1xN	1, 2, 3
Matrix	a rectangular set of like data elements with rows and columns	NxN	1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Built in functions

- We can also combine vectors into a data object called a matrix by using the `cbind()` or `rbind()` functions

```
ls()


> [1] "a" "b" "c" "d"

z <- c(21,22,23, 24, 25)
mat1 <- cbind(d,z)
mat2 <- rbind(d,z)
```

Built in functions

```
mat1

>      d  z
> [1,] 1 21
> [2,] 2 22
> [3,] 3 23
> [4,] 4 24
> [5,] 5 25

mat2

>   [,1] [,2] [,3] [,4] [,5]
> d    1    2    3    4    5
> z   21   22   23   24   25
```

Built in functions

Other useful R functions include:

Name	Description
`rbind()`	same as `cbind()` but for rows.
`dim()`	returns the dimensions of a data object, like a matrix or data frame.
`length()`	returns the length of a vector.
`rm()`	removes a particular object in the workspace (the command `rm(list=ls())` will clear all objects from the workspace)
`class()`	tells you what class of data an object is (e.g. matrix, numeric vector, etc.)

Part 2: Working with Data

Working with Data

- Vectors and matrices are 1 and 2 dimensional data objects, respectively
    - The terminology and the syntax that R uses to work with such objects are based on matrix algebra
- Matrices don't have to be numbers only
    - They can be character strings or logic statements (e.g. TRUE/FALSE). But they need to be consistent within a particular object

Indexing: Vectors

- **Indexing** is the process of referring to a subset of elements within a data object
- For vectors:
  - we can index based on the element number within brackets:

```
z

> [1] 21 22 23 24 25

z[4]

> [1] 24
```

Indexing: Vectors

- Also, a colon indicates a series of numbers, so `z[1:3]` means the first **through** third elements of `z`

```
z[1:3]
```

```
> [1] 21 22 23
```

Indexing: Matrices

- Observations stored in matrices can be accessed using similar matrix notation
  - That is, giving the coordinates of the observations by row and then column, separated by a comma: [**row, column**]

```
mat1
```

```
>      d  z
> [1,] 1 21
> [2,] 2 22
> [3,] 3 23
> [4,] 4 24
> [5,] 5 25
```

Indexing: Matrices

- Observations stored in matrices can be accessed using similar matrix notation
  - That is, giving the coordinates of the observations by row and then column, separated by a comma: [row, column]

```
mat1[1,2]
```

```
>  z
> 21
```

```
mat1[1,]
```

```
>  d  z
>  1 21
```

Data frames

- Often working with data we have variables of multiple 'types' (e.g. numbers, characters, etc.)
  - The data frame is the way to manage them in one object. We can make data frames from existing objects

```
df <- data.frame(mat1)
df

>   d  z
> 1 1 21
> 2 2 22
> 3 3 23
> 4 4 24
> 5 5 25
```

Importing data

- Often, we have data stored in files that we'd like to import into R for analysis
  - These may come in many formats, e.g. .Rdata, .csv, .txt
  - Different R commands to read in different types of data

Data format	Command
.Rdata	load()
.csv	read.csv()
.txt	read.table()

Importing data

- Let's read in example data
- To do so, first tell R which folder you saved the data in using the command 'setwd()'

```
# Example for mac users
setwd("/Users/PaulYousefi/Documents/PhD/SW_R_Tutorial/")

# Example for Windows users
setwd("C:/Rab/")
```

- Also, Rstudio will allow you to do this with the mouse by selecting Session -> Set Working Directory -> Choose Directory from the top menu

Quick side notes

1. In R, the slashes in path names must be forward-slashes (/)
   - Mac's use the "/" in their path names
   - However, Windows paths employ "\" back slashes
2. To write comments in your R code, begin the line with a pound sign (#)
   - This will pass the text to R without R interpreting it as code

Importing data

- Now we can read in the dataset using the `read.csv()` function

```
crime<-read.csv("CRIME.csv")
```

Examining a data frame

- Rather than printing all the data, we can get a feeling for what's in the dataset using some commands we already know

```
class(crime)

> [1] "data.frame"

dim(crime)

> [1] 51  8
```

Examining a data frame

- Also, new commands can help: `head()` and `tail()`

```
head(crime)
```

```
>   state violent murder metro white hsgrad poverty snglpa
> 1    AK     761    9.0  41.8  75.2   86.6     9.1    14.
> 2    AL     780   11.6  67.4  73.5   66.9    17.4    11.
> 3    AR     593   10.2  44.7  82.9   66.3    20.0    10.
> 4    AZ     715    8.6  84.7  88.6   78.7    15.4    12.
> 5    CA    1078   13.1  96.7  79.3   76.2    18.2    12.
> 6    CO     567    5.8  81.8  92.5   84.4     9.9    12.
```

```
str(crime)
```

```
> 'data.frame': 51 obs. of  8 variables:
>  $ state  : Factor w/ 51 levels "AK","AL","AR",..: 1 2 3
>  $ violent: int  761 780 593 715 1078 567 456 686 1206 72
>  $ murder : num  9 11.6 10.2 8.6 13.1 5.8 6.3 5 8.9 11.4
>  $ metro  : num  41.8 67.4 44.7 84.7 96.7 81.8 95.7 82.7
>  $ white  : num  75.2 73.5 82.9 88.6 79.3 92.5 89 79.4 83
>  $ hsgrad : num  86.6 66.9 66.3 78.7 76.2 84.4 79.2 77.5
>  $ poverty: num  9.1 17.4 20 15.4 18.2 9.9 8.5 10.2 17.8
>  $ snglpar: num  14.3 11.5 10.7 12.1 12.5 12.1 10.1 11.4
```

Indexing: Data frames

- We can still subset data in the way we did earlier with matrices and data frames that we'd created. By using both variable names and matrix notation

```
e<-crime$metro
f<-crime[1, 2:5]
f
```

```
>   violent murder metro white
> 1     761      9  41.8  75.2
```

Indexing: Data frames

- We can also subset using logic statements. So if we wanted to restrict to obsevations only taken from California, we could keep only rows of interest with a statement like:

```
CA <- crime[crime$state=="CA",]
dim(CA)
```

```
> [1] 1 8
```

Data frames: New variables

- Besides selecting subsets of data, we can also easily add variables to a dataset through assignment

```
# Make interaction variable for metro and poverty
crime$interact <- crime$metro * crime$poverty
summary(crime$interact)

>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>     270     734     900     955    1130    2640
```

Data frames: Recode variables

- We can also recode variables using logic statements and indexing

```
crime$poverty_cat[crime$poverty < 10] <- "Low"
crime$poverty_cat[crime$poverty >= 10] <- "High"
```

Data frames: Recode variables

- Or using the `ifelse()` command

```
crime$poverty_binary <- ifelse(crime$poverty >= 12,
    c(1), c(0))
```

Data frames: Sorting

- To sort, we again use matrix notation and the order() command

```
crime[order(crime$murder),]
```

```
>     state violent murder metro white hsgrad poverty snglpa
> 21     ME     126    1.6  35.7  98.5   78.8    10.7    10.
> 28     ND      82    1.7  41.6  94.2   76.7    11.2     8.
> 30     NH     138    2.0  59.4  98.0   82.2     9.9     9.
> 12     IA     326    2.3  43.8  96.6   80.1    10.3     9.
> 13     ID     282    2.9  30.0  96.7   79.7    13.1     9.
> 26     MT     178    3.0  24.0  92.6   81.0    14.9    10.
> 44     UT     301    3.1  77.5  94.8   85.1    10.7    10.
> 23     MN     327    3.4  69.3  94.0   82.4    11.6     9.
> 41     SD     208    3.4  32.6  90.2   77.1    14.2     9.
```

Part 3: Example analysis

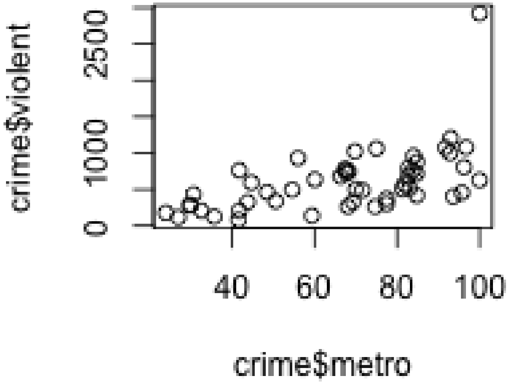
Example

- Beyond learning to manipulate data objects in R, there are some core analysis functions that you'll likely need to get up to speed
- The next slides will give a quick overview of the commands needed for:
  - Plotting
  - Regression
  - Correlation
  - Chi-squared

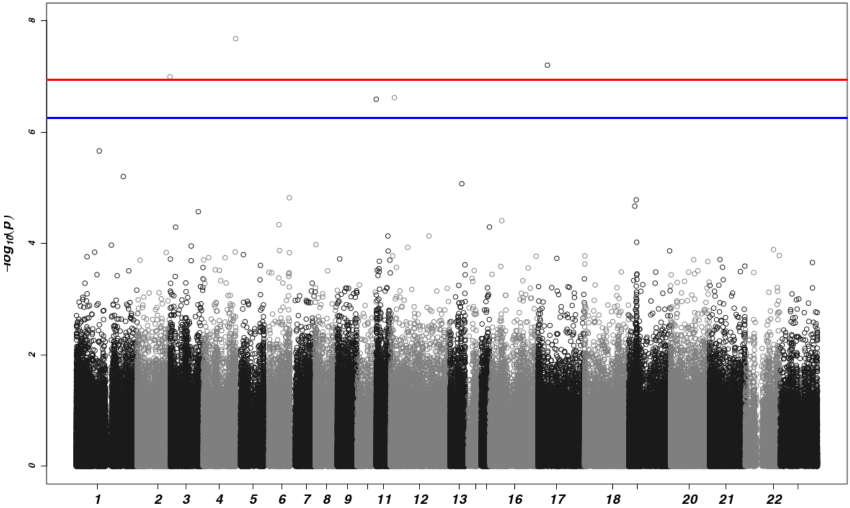
Example: Plotting

- Running simple plots is easy in R using the `plot()` command
- There are also several common add-on packages for making more complicated graphics (e.g. ggplot2, lattice, etc.)

```
plot(crime$violent, crime$metro)
```


crime$violent
0
1000
2500
40
60
80
100
crime$metro

Example: Plotting



Example: Linear regression

- The `glm()` command allows one to run many different types of linear models, including OLS regression:

```
fit1 <- glm(violent ~ metro + poverty + snglpar,
    data = crime, family = "gaussian")

summary(fit1)
```

```
>
> Call:
> glm(formula = violent ~ metro + poverty + snglpar, family
>     data = crime)
>
> Deviance Residuals:
>    Min       1Q   Median       3Q      Max
> -523.0   -99.5      9.4    107.3    426.1
>
> Coefficients:
>             Estimate Std. Error t value Pr(>|t|)
> (Intercept) -1666.44     147.85  -11.27  5.9e-15 ***
> metro           7.83       1.25    6.24  1.2e-07 ***
> poverty        17.68       6.94    2.55    0.014 *
> snglpar       132.41      15.50    8.54  4.0e-11 ***
> ---
```

Example: Logistic regression

- By changing the 'family' to binomial, we can use glm() to perform logistic regression

```
fit2 <- glm(poverty_binary ~ hsgrad,
    data = crime, family = "binomial")

summary(fit2)
```

```
>
> Call:
> glm(formula = poverty_binary ~ hsgrad, family = "binomial
>
> Deviance Residuals:
>    Min       1Q   Median       3Q      Max
> -2.088  -0.877   0.235    0.782    1.879
>
> Coefficients:
>             Estimate Std. Error z value Pr(>|z|)
> (Intercept)  24.2574     7.5199    3.23   0.0013 **
> hsgrad       -0.3083     0.0963   -3.20   0.0014 **
> ---
> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1
>
> (Dispersion parameter for binomial family taken to be 1)
```

Example: Logistic regression

```
exp(coef(fit2)) # exponentiated coefficients

> (Intercept)        hsgrad
>   3.426e+10    7.347e-01

exp(confint(fit2)) # 95% CI for exponentiated coefficients

> Waiting for profiling to be done...

>                    2.5 %     97.5 %
> (Intercept) 1.070e+05 1.051e+18
> hsgrad      5.894e-01 8.647e-01
```

Example: Correlation

- Correlation matrices can be made using the `cor()` command:

```
variables <- crime[,c("violent", "metro", "poverty")]
cor(variables)
```

```
>           violent    metro  poverty
> violent  1.0000  0.54404  0.50951
> metro    0.5440  1.00000 -0.06054
> poverty  0.5095 -0.06054  1.00000
```

Example: Chi squared

- Chi squared tests are easy to perform after using the `table()` function to make a 2x2 table of your data

```
# Create another binary variable
crime$murder_binary <- ifelse(crime$murder >= 6,
    c(1), c(0))

# Make a table
pv <- table(crime$poverty_binary, crime$murder_binary)
```

Example: Chi squared

```
chisq.test(pv)
```

```
>
>     Pearson's Chi-squared test with Yates' continuity corre
>
> data:   pv
> X-squared = 13.33, df = 1, p-value = 0.0002614
```

Part 4: Packages

Installing packages

- An appealing feature of R is the ease of using and distributing 3rd party functions through packages
- It is useful to search for R packages that perform specialized analyses before custom coding something because packages undergo quality control and often come with examples
- The command install.packages() accesses the repository (CRAN) and downloads the requested package(s) from a mirror
- Once installed, the library() command needs to be used at the beginning of every new R session to make the functions in the package available for use

Installing packages

- Install and load the gdata package in R. You will likely be prompted to choose the CRAN mirror (USA CA1 is Berkeley)

```
install.packages("gdata")
library(gdata)
```

- The help file for the downloaded package can be accessed by typing help(package=gdata))
- Visit http://cran.r-project.org/web/packages/ for lists of available packages

Packages for specific analysis

- Increasingly, methodologists are distributing code from their research as R packages
  - Example: The `twang` package
    - Toolkit for Weighting and Analysis of Non-equivalent Groups
    - Developed by RAND researchers
    - `http://cran.r-project.org/web/packages/twang/index.html`

Part 5: Resources

Online resources:

- **R help**: Remember all functions have a help page that can be accessed by typing ? followed by the function name
- **Quick R**: `http://www.statmethods.net/` This is a great resource for finding help on common analysis or data management issues
- **UCLA R**: `http://www.ats.ucla.edu/stat/r/` UCLA hosts resources on statistical computing with a variety of software. Their R page is another great place to look for help
- **Google**: Seriously. Most R answers are only a search away.

Online resources cont.:

Two good tutorials cover similar and some additional material to today's workshop:

- **R for Beginners** `http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf`
- **A Short R Tutorial** `http://strata.uga.edu/software/pdf/RTutorial.pdf`

Also, the O'Reilly series of programming books are available to all Berkeley students electronically at

- `http://proquest.safaribooksonline.com/`
- search "R" to find many titles on detailed analyses

Recommended Books:

- Faraway, J.J. 2004. Linear Models with R. Chapman & Hall/CRC Texts in Statistical Science.
- Murrell, P. 2011. R Graphics: 2nd Edition. Chapman & Hall/CRC The R Series.
- Venables, W.N. and Ripley, B.D. 2002. Modern Applied Statistics with S: 4th Edition. Springer.

In Person

- The D-lab in Barrows Hall holds r tutorial sessions regularly, often with options for all levels of R knowledge
  - http://dlab.berkeley.edu/
- They also have drop-in tutoring sessions available with several tutors familiar with R

But mostly. . .

THANK YOU!

Also, please email if you have questions or need additional advice: yousefi@berkeley.edu