Introduction to R

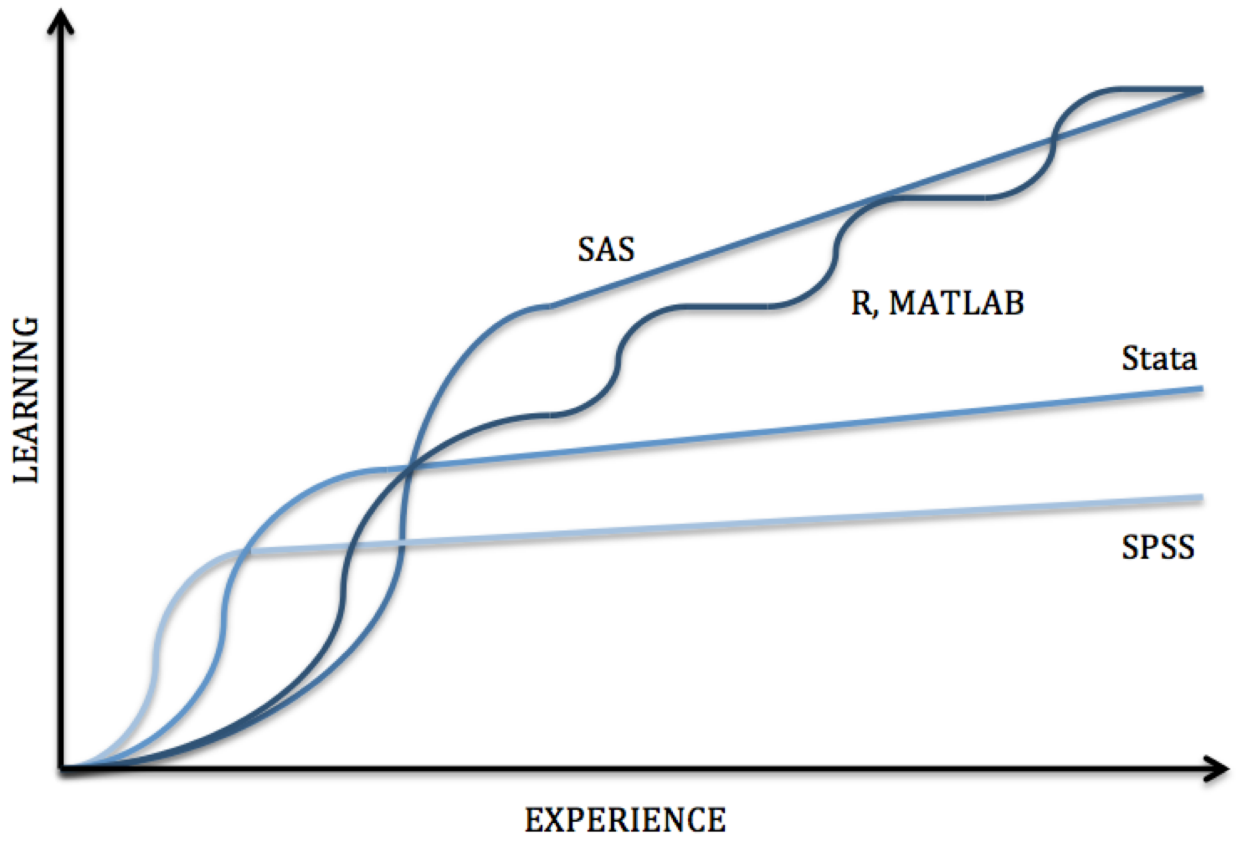
Adrian Rohit Dass
Institute of Health Policy, Management, and Evaluation
Canadian Centre for Health Economics
University of Toronto

September 21st, 2018

Outline

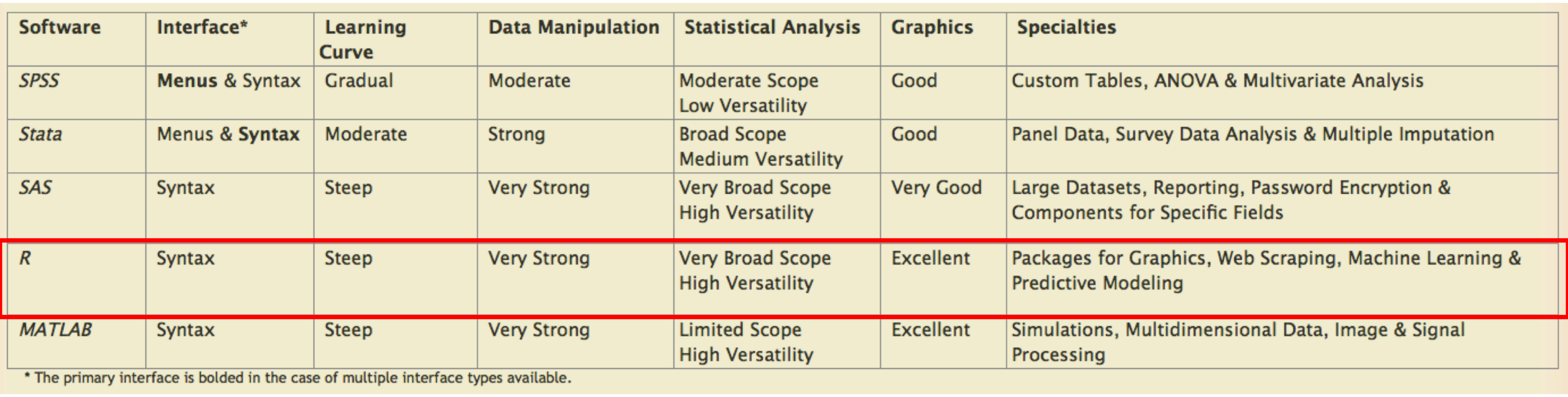
- Why use R?
- R Basics
- R for Database Management
  - Reading-in data, merging datasets, reshaping, recoding variables, sub-setting data, etc.
- R for Statistical Analysis
  - Descriptive Analysis, Regression Based Approaches
- Other topics in R
  - Parallel Processing
  - R Studio
- R Resources

Learning Curves of Various Software Packages



Source: https://sites.google.com/a/nyu.edu/statistical-software-guide/summary

Summary of Various Statistical Software Packages



Software	Interface*	Learning Curve	Data Manipulation	Statistical Analysis	Graphics	Specialties
SPSS	**Menus** & Syntax	Gradual	Moderate	Moderate Scope Low Versatility	Good	Custom Tables, ANOVA & Multivariate Analysis
Stata	Menus & **Syntax**	Moderate	Strong	Broad Scope Medium Versatility	Good	Panel Data, Survey Data Analysis & Multiple Imputation
SAS	Syntax	Steep	Very Strong	Very Broad Scope High Versatility	Very Good	Large Datasets, Reporting, Password Encryption & Components for Specific Fields
R	Syntax	Steep	Very Strong	Very Broad Scope High Versatility	Excellent	Packages for Graphics, Web Scraping, Machine Learning & Predictive Modeling
MATLAB	Syntax	Steep	Very Strong	Limited Scope High Versatility	Excellent	Simulations, Multidimensional Data, Image & Signal Processing

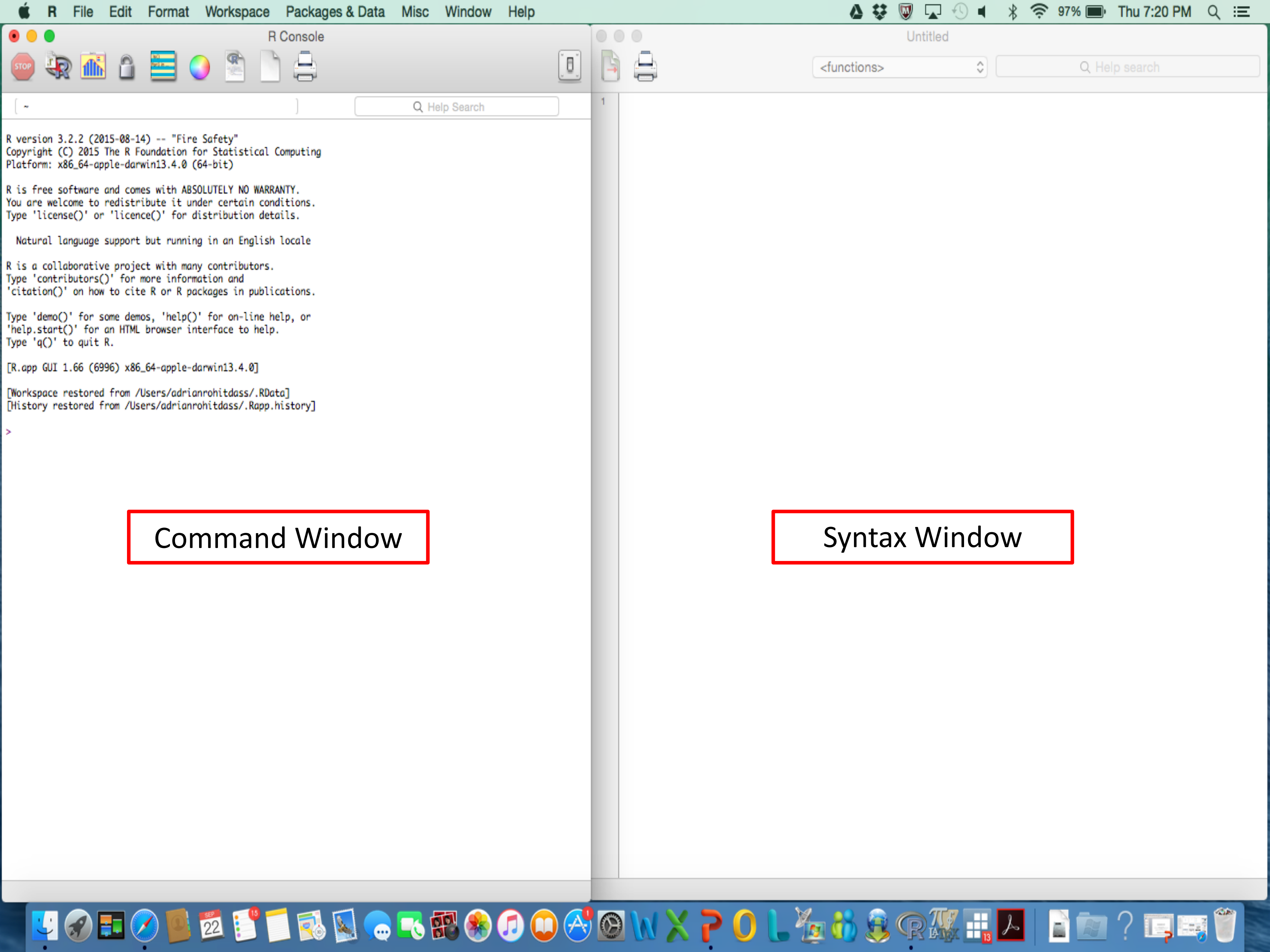
* The primary interface is bolded in the case of multiple interface types available.

Source: https://sites.google.com/a/nyu.edu/statistical-software-guide/summary

Goals of Today’s Talk

- Provide an overview of the use of R for database management
  - By doing so, we can hopefully lower the learning curve of R, thereby allowing us to take advantage of its “very strong” data manipulation capabilities
- Provide an overview of the use of R for statistical analysis
  - This includes descriptive analysis (means, standard deviations, frequencies, etc.) as well as regression analysis
  - R contains a wide number of pre-canned routines that we can use to implement the method we’d like to use

Part I
R Basics



```
R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (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.

[R.app GUI 1.66 (6996) x86_64-apple-darwin13.4.0]

[Workspace restored from /Users/adrianrohitdass/.RData]
[History restored from /Users/adrianrohitdass/.Rapp.history]

>
```

Programming Language

- Programming language in R is *object oriented*
  - Roughly speaking, this means that data, variables, vectors, matrices, characters, arrays, etc. are treated as “objects” of a certain “class” that are created throughout the analysis and stored by name.
  - We then apply “methods” for certain “generic functions” to these objects
- Case sensitive (like most statistical software packages), so be careful

Classes in R

- In R, every object has a *class*
  - For example, character variables are given the class of factor or character, whereas numeric variables are integer
- Classes determine how objects are handled by generic functions. For example:
  - the mean(x) function will work for integers but not for factors or characters - which generally makes sense for categorical (with more than two categories) and ordinal variables)

Packages in R

- Functions in R are stored in *packages*
  - For example, the command for OLS (lm) is accessed via the “stats” package, which is available in R by default
  - Only when a package is *loaded* will its contents be available. The full list of packages is <u>not</u> loaded by default for computational efficiency
  - Some routines in R are not installed (and thus loaded) by default, meaning that we will have to install packages that we will need beforehand, and then load them later on

Packages available (and loaded) in R by default

Package	**Description**
base	Base R functions (and datasets before R 2.0.0).
compiler	R byte code compiler (added in R 2.13.0).
datasets	Base R datasets (added in R 2.0.0).
grDevices	Graphics devices for base and grid graphics (added in R 2.0.0).
graphics	R functions for base graphics.
grid	A rewrite of the graphics layout capabilities, plus some support for interaction.
methods	Formally defined methods and classes for R objects, plus other programming tools, as described in the Green Book.
parallel	Support for parallel computation, including by forking and by sockets, and random-number generation (added in R 2.14.0).
splines	Regression spline functions and classes.
stats	R statistical functions.
stats4	Statistical functions using S4 classes.
tcltk	Interface and language bindings to Tcl/Tk GUI elements.
tools	Tools for package development and administration.
utils	R utility functions.

Source: https://cran.r-project.org/doc/FAQ/R-FAQ.html

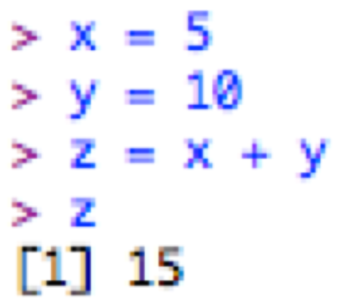
For database management, we usually won't need to load or install any additional packages, although we might need the "foreign" package (available in R by default, but not initially loaded) if we're working with a dataset from another statistical program (SPSS, SAS, STATA, etc.)

Packages in R (Continued)

- To load a package, type require(packagename)
  - Ex: To load the foreign package, I would type require(foreign) before running any routines that require this package
- To install a package in R:
  - Type install.packages("*packagename*") in command window
  - For example, the package for panel data econometrics is plm in R. So, to install the plm package, I would type install.packages("plm").
    - Note that, although installed, a package will not be loaded by default (i.e. when opening R). So, you'll need require(package) at the top of your code (or at least sometime before the package is invoked).
  - Some packages will draw upon functions in other packages, so those packages will need to be installed as well. By using install.packages(" "), it will automatically install dependent packages

Some Basic Operations in R

- Q: If x = 5, and y = 10, and z = x + y, what is the value of z?
- Let’s get R to do this for us:



```
> x = 5
> y = 10
> z = x + y
> z
[1] 15
```

- In this example, we really only used the ‘+’ operator, but note that ‘-’, ‘/’, ‘*’, ‘^’, etc. work the way they usually do for scalar operations

Some Basic Operations in R

- Now suppose we created the following vectors:

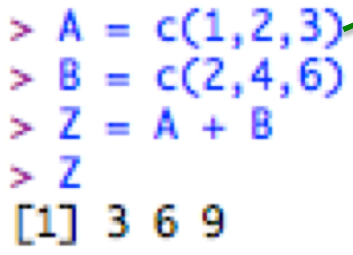
A =

1
2
3

B =

2
4
6

- What is A + B?



```
> A = c(1,2,3)
> B = c(2,4,6)
> Z = A + B
> Z
[1] 3 6 9
```

In R, c() is used to combine values into a vector or list. Since we have a list of values, we need to use it here

- Note that with vectors, '+', '-', '/', '*', '^' perform element-wise calculations when applied to vectors. So, vectors need to be the same length.

Working with Matrices in R

- A matrix with typical element (i,j) takes the following form:

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	(2,3)
(3,1)	(3,2)	(3,3)

- Where i = row number and j = column number
- In R, the general formula for extracting elements (i.e. single entry, rows, columns) is as follows:
  - matrixname[row #, column #]
- If we leave the terms in the brackets blank (or leave out the whole bracket term) R will spit out the whole matrix

Working with Matrices in R (Continued)

- Example: Suppose we had the following matrix:

1	4	7
2	5	8
3	6	9

- To create this matrix in R, type:

```
> matrix = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3, ncol=3)
```

- Extract the element in row #2, column #3

```
>   matrix[2,3]
8
```

- Extract the second row

```
>   matrix[2,]
2 5 8
```

- Extract the last two columns

```
>   matrix[,c(2,3)]
4 7
5 8
6 9
```

Since we have a list of columns, we need to use c() here

Working with Matrices in R (Continued)

- Example: Suppose now we had the following vector, with typical element 'i':

1
2
3

- Extract the third element of the vector

```
> vector[3]
3
```

- Suppose the 2nd element should be 5, not 2. How do we correct this value?

```
> vector[2] = 5
> vector
1
5
3
```

But wait a minute…

- Q: If this is a tutorial on the use of R for database management/statistical analysis, then why are we learning about vectors/matrices?
- A: The way we work with data in R is very similar/identical to how we work with vectors/matrices
  - This is different from other statistical software packages, which may be a contributing factor to the "high" learning curve in R
- The importance of vector/matrices operations will become more clear as we move

Part II
R for Database Management

Capability of R with Different Types of Data

- Feature rich software for analyzing various types of data:
  - Cross-sectional data: A collection of individuals (persons, countries, etc.) in one time period. Quite common form of micro-data, like surveys.
  - Time Series Data: Many points in time, but for one individual entity. Usually in aggregated form, like rates or percentages.
  - Panel Data: Combination of cross-sectional and time series data. Ex: survey of the same individuals over many years, or aggregate data on murder rates for each province in Canada over many years.
- R has routines for analyzing virtually any type of data

Reading Data into R

What format is the data in?

- Data from Comma Separated Values File (.csv)
  - Package: utils
  - Formula: read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", ...)
- Data from Excel File (.xlsx)
  - Package: xlsx
  - Formula: read.xlsx(file, sheetIndex, sheetName=NULL, rowIndex=NULL, startRow=NULL, endRow=NULL, colIndex=NULL, as.data.frame=TRUE, header=TRUE, colClasses=NA, keepFormulas=FALSE, encoding="unknown", ...)
- Data from STATA (.dta)
  - Package: foreign
  - read.dta(file, convert.dates = TRUE, convert.factors = TRUE, missing.type = FALSE, convert.underscore = FALSE, warn.missing.labels = TRUE)

Other Formats: See package “foreign”
https://cran.r-project.org/web/packages/foreign/foreign.pdf

Reading Data into R

Examples:

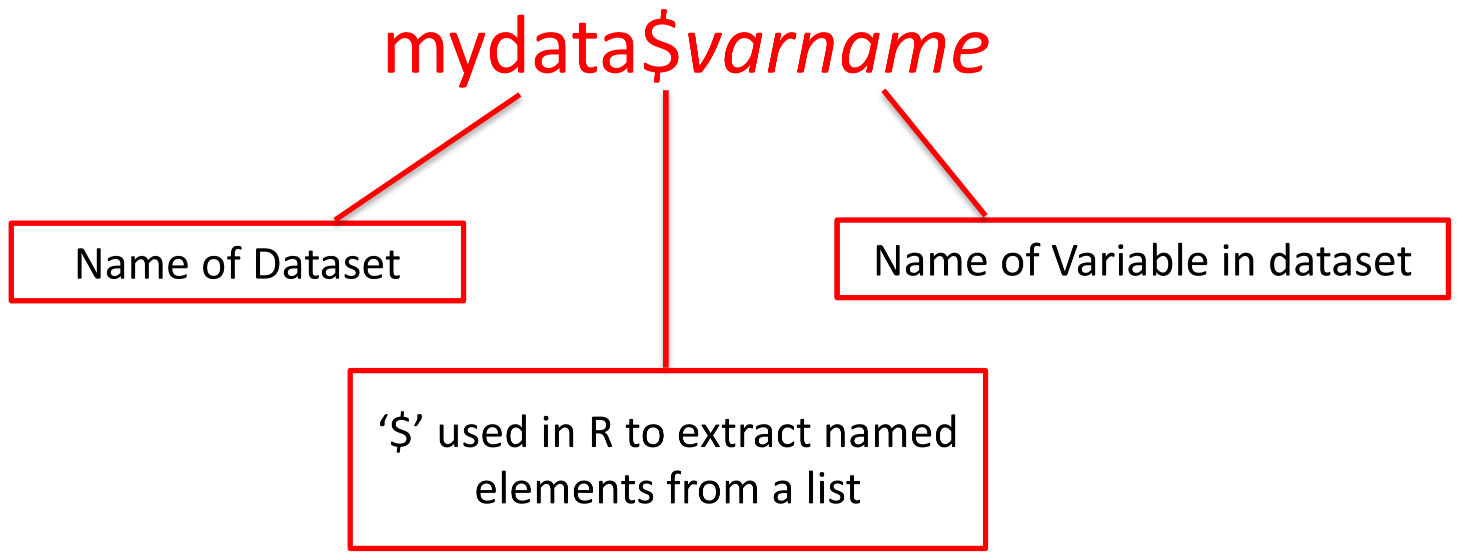
- CSV file with variable names at top
  - data = read.csv(“C:/Users/adrianrohitdass/Documents/R Tutorial/data.csv”)
- CSV file with no variable names at top
  - data = read.csv(“C:/Users/adrianrohitdass/Documents/R Tutorial/data.csv”, header=F)
- STATA data file (12 or older)
  - require(foreign)
  - data = read.dta(“C:/Users/adrianrohitdass/Documents/R Tutorial/data.dta”)
- STATA data file (13 or newer)
  - require(readstata13)
  - data = read.dta13(“C:/Users/adrianrohitdass/Documents/R Tutorial/data.dta”)

Comparison and Logical Operators

Operator	Description	Example
=	Assign a value	x = 5
==	Equal to	sex ==1
!=	Not equal to	LHIN != 5
>	Greater than	income >5000
<	Less than	healthcost < 5000
>= or <=	Greater than or equal to Less than or equal to	income >= 5000 healthcost <= 5000
&	And	sex==1 & age>50
|	Or	LHIN==1 | LHIN ==5

Referring to Variables in a Dataset

- Suppose I had data stored in “mydata” (i.e an object created to store the data read-in from a .csv by R). To refer to a specific variable in the dataset, I could type



mydata$*varname*

Name of Dataset
Name of Variable in dataset
‘$’ used in R to extract named elements from a list

Creating a new variable/object

- No specific command to generate new variables (in contrast to STATA's "gen" and "egen" commands) but may involve a routine depending on what's being generated
  - x = 5 generates a 1x1 scalar called "x" that is equal to 5
  - data$age = year – data$dob creates a new variable "age" in the dataset "data" that is equal to the year – the person's date of birth (let's say in years)

Looking at Data

- Display the first or last few entries of a dataset:
  - Package: utils
  - First few elements of dataset (default is 5):
    - head(x, n, …)
  - Last few elements of dataset (default is 5):
    - tail(x, n, …)
- List of column names in dataset
  - Package: base
  - Formula: colnames(x)

Missing Values

Missing Values are listed as “NA” in R

- Count number of NA’s in column

```
sum(is.na(x))
```

- Recode Certain Value’s as NA (i.e. non responses coded as -1)

```
x[x==-1] = NA
```

Renaming Variables (Columns)

A few different ways to do this:

- To rename the 'ith' column in a dataset
  - colnames(data)[i] = "My Column Name"
- Can be cumbersome – especially if don't know column # of the column you want to rename (just it's original name)
- Alternative:
  - colnames(data)[which(colnames(data) == 'R1482600')] = 'race'

Grabs column names from specified dataset

Look-up that returns the column #

New column name

Subsetting Data

- Subsetting can be used to restrict the sample in the dataset, create a smaller data with fewer variables, or both
- Recall: extracting elements from a matrix in R
  - matrixname[row #, column #]
- What's the difference between a matrix and a dataset?
  - Both have row elements
    - Typically the individual records in a dataset
  - Both have column elements
    - Typically the different variables in the dataset
- If we think of our dataset as a matrix, then the concept of subsetting in R becomes a lot easier to digest

Subsetting Data (Continued)

Examples:

- Restrict sample to those with age >=50

```
> datas1 = data[data$age >=50,]
```

- Create a smaller dataset with just ID, age, and height

```
> datas2 = data[, c("ID", "age", "height")]
```

- Create a smaller dataset with just ID, age, and height; with age >=50

```
> datas3 = data[data$age>=50, c("ID", "age", "height")]
```

Recoding Variables in R

- Usually done with a few lines of code using comparison and logical operators
- Ex: Suppose we had the following for age:
  > data$age = [19, 20, 25, 30, 45, 55]
- If we wanted to create a categorical variable for age (say, <20, 20-39, 40-59), we could do the following:
  > data$agecat[data$age <20] = 1
  > data$agecat[data$age >=20 & data$age <40] = 2
  > data$agecat[data$age >=40 & data$age <60] = 3
  > data$agecat
  > [1, 2, 2, 2, 3, 3]

Merging Datasets

Suppose we had the following 2 datasets:

Data1

Id	Age	Income
1	55	49841.65
2	63	46884.78
3	65	45550.87
4	69	26254.15
5	52	22044.73

Data2

Id	Health Care Cost
1	188.1965
2	172.2420
3	102.8355
4	150.2247

Our first dataset contains some data on age and income, but not health care costs to the public system. Dataset 2 contains this data, but was not initially available to us. It also doesn't have age or income.

The common element between the two datasets is "ID", which uniquely identifies the same individuals across the two datasets.

Note that, for some reason, individual 5 does not have a reported health care cost

Merging Datasets (Continued)

- Command: merge
  - Package: base
- For our example:
  - Datam = merge(Data1, Data2, by="id", all=T)

Unique identifier across datasets

Optional, but default is F, meaning those who can't be matched will be excluded

  - Resulting Dataset

Datam

Id	Age	Income	Health Care Cost
1	55	49841.65	188.1965
2	63	46884.78	172.2420
3	65	45550.87	102.8355
4	69	26254.15	150.2247
5	52	22044.73	NA

<u>Part II</u>

R for Statistical Analysis

Descriptive Statistics in R

- Mean
  - Package: base
  - Formula: mean(x, trim = 0, na.rm = FALSE, ...)
- Standard Deviation
  - Package: stats
  - Formula: sd(x, na.rm = FALSE)
- Correlation
  - Package: stats
  - Formula: cor(x, y = NULL, use = "everything”, method = c("pearson", "kendall", "spearman"))

Descriptive Statistics (Example)

- Suppose we had the following data column in R (transposed to fit on slide):
  - Vector = [5,5,6,4]
- What is the mean of the vector?
- In R, I would type

```
> mean(Vector)
> 5
```

Descriptive Statistics (Example)

- Suppose now we had the following:
  - Vector = [5,5,6,4,NA]
- What is the mean of the vector?
- In R, I would type

```
> mean(Vector)
> NA
```

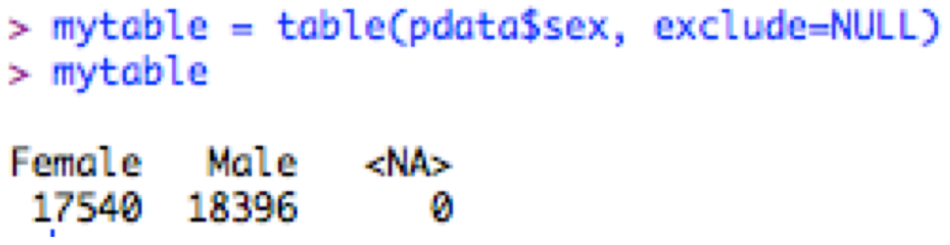
- Why did I get a mean of NA?
  - Our vector included a missing value, so R couldn’t compute the mean as is.
- To remedy this, I would type

```
> mean(Vector, na.rm=T)
> 5
```

Tabulations R

- Tabulations of categorical/ordinal variables can be done with R's *table* command:
  - Package: base
  - Formula: table(..., exclude = if (useNA == "no") c(NA, NaN), useNA = c("no", "ifany", "always"), dnn = list.names(...), deparse.level = 1)

Ex: Table Sex Variable, with extra column for missing values (if any)



```
> mytable = table(pdata$sex, exclude=NULL)
> mytable

Female   Male   <NA>
 17540  18396      0
```

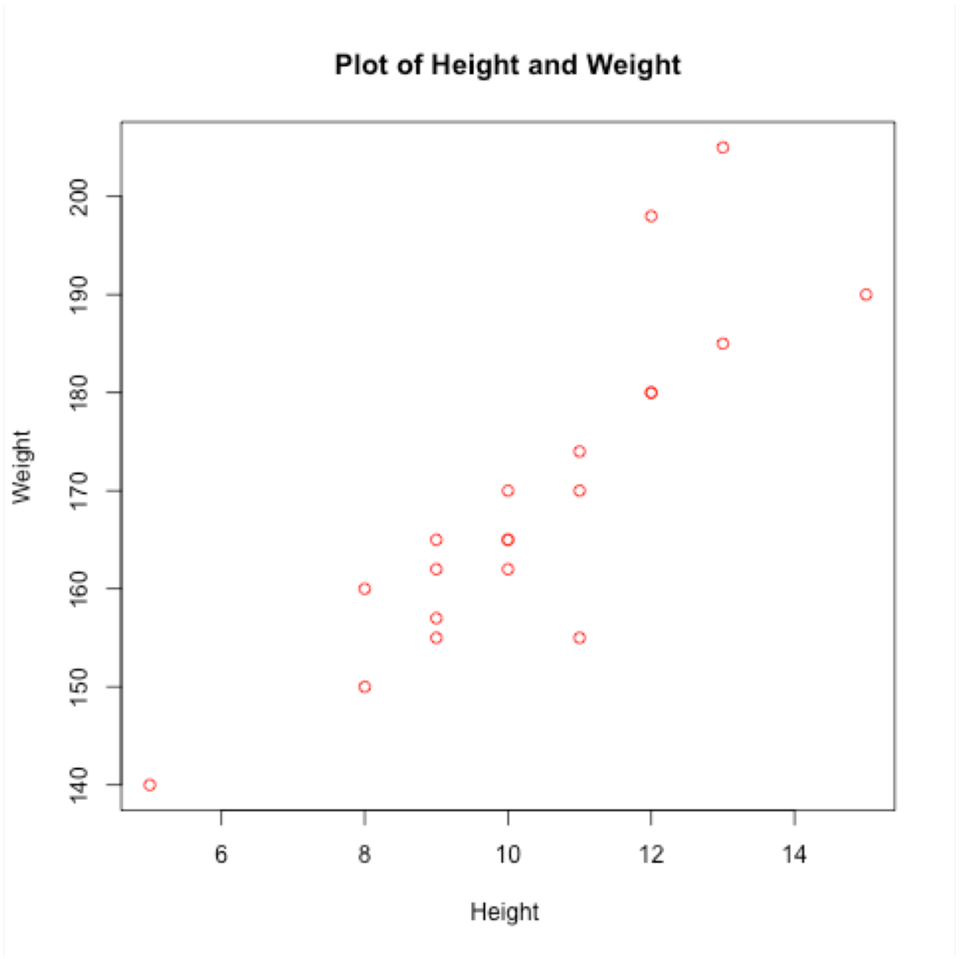
Graphing Data in R

- Basic plot in R
  - Package: graphics
  - Formula: plot(x, y, ...)

Example:

```
plot(lmdata$height, lmdata$weight, main = "Plot of Height and
Weight", xlab="Height ",ylab="Weight", col=2)
```

Resulting Graph



Ordinary Least Squares

- The estimator of the regression intercept and slope(s) that minimizes the sum of squared residuals (Stock and Watson, 2007).
  - Package: stats
  - Formula: lm(formula, data, subset, weights, na.action, method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset, ...)

Examples:

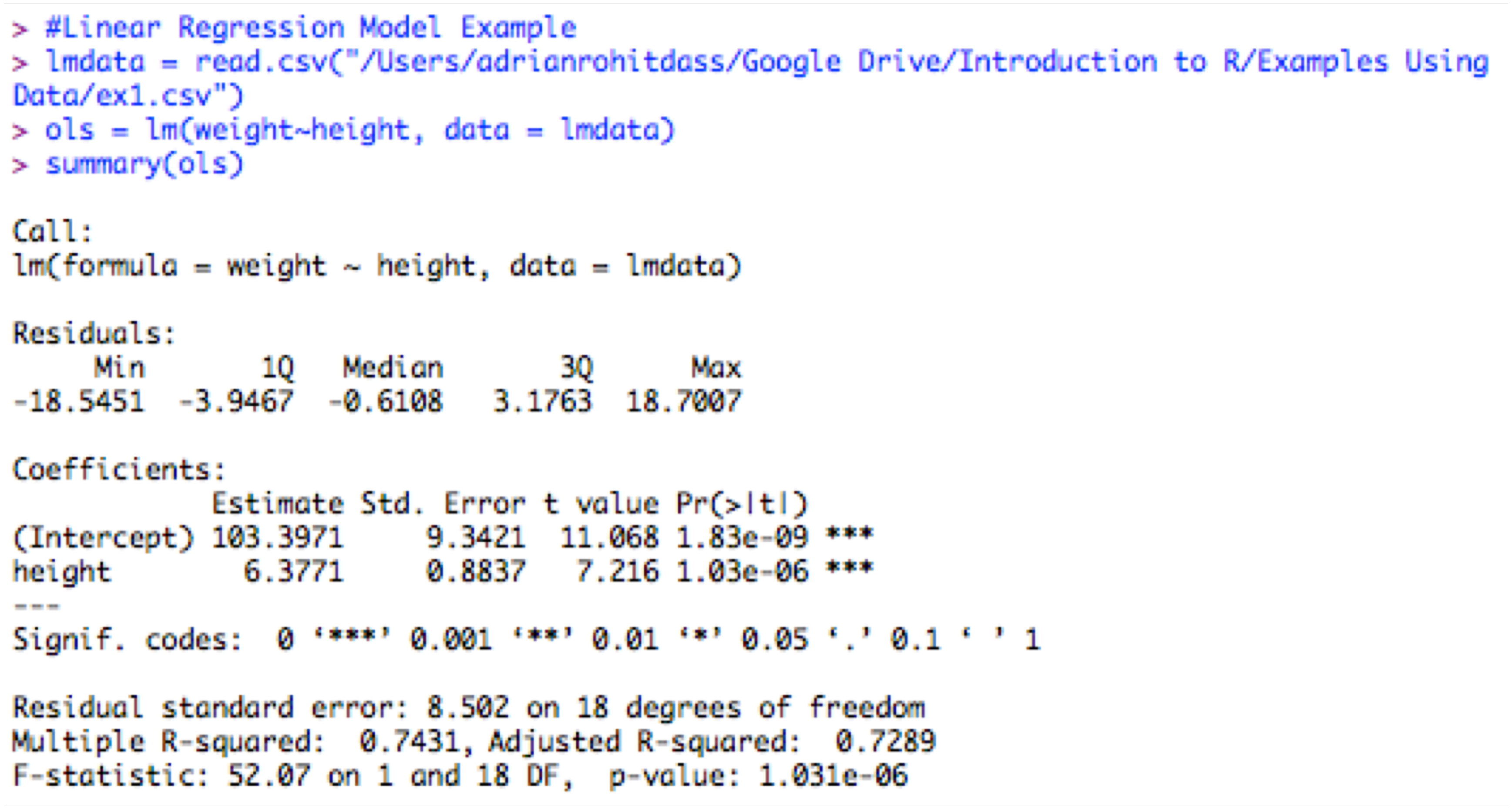
Regression of “weight” on “height” using dataset “lmdata”

```
ols = lm(weight~height, data=lmdata)
```

Online Help File
https://stat.ethz.ch/R-manual/R-devel/library/stats/html/lm.html

Ordinary Least Squares

Example: OLS regression of Weight on Height (Univariate Analysis)



```
> #Linear Regression Model Example
> lmdata = read.csv("/Users/adrianrohitdass/Google Drive/Introduction to R/Examples Using
Data/ex1.csv")
> ols = lm(weight~height, data = lmdata)
> summary(ols)

Call:
lm(formula = weight ~ height, data = lmdata)

Residuals:
     Min       1Q   Median       3Q      Max 
-18.5451  -3.9467  -0.6108   3.1763  18.7007 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 103.3971     9.3421  11.068 1.83e-09 ***
height        6.3771     0.8837   7.216 1.03e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 8.502 on 18 degrees of freedom
Multiple R-squared:  0.7431, Adjusted R-squared:  0.7289 
F-statistic: 52.07 on 1 and 18 DF,  p-value: 1.031e-06
```

Post-Estimation

Package: lmtest

- Breusch-Pagan test against heteroskedasticity.

bptest(formula, varformula = NULL, studentize = TRUE, data = list())

- Ramsey’s RESET test for functional form.

resettest(formula, power = 2:3, type = c("fitted", "regressor", "princomp"), data = list())

- Variance Inflation Factor (VIF)

Package: car

vif(*model*)

Exporting Regression Results

Outputting regression results to look at, modify, or insert into document.

Package: estout

Formula: esttab(t.value = FALSE, p.value = FALSE, round.dec = 3, caption = NULL,label = NULL, texfontsize = NULL, sig.levels = c(0.1, 0.05, 0.01), sig.sym=c("*","**","***"), filename=NULL, csv=FALSE, dcolumn=NULL, table="table", table.pos="htbp", caption.top=FALSE, booktabs=FALSE, var.order=NULL, sub.sections=NULL,var.rename=NULL, resizebox=c(0,0),colnumber=FALSE, store="default")

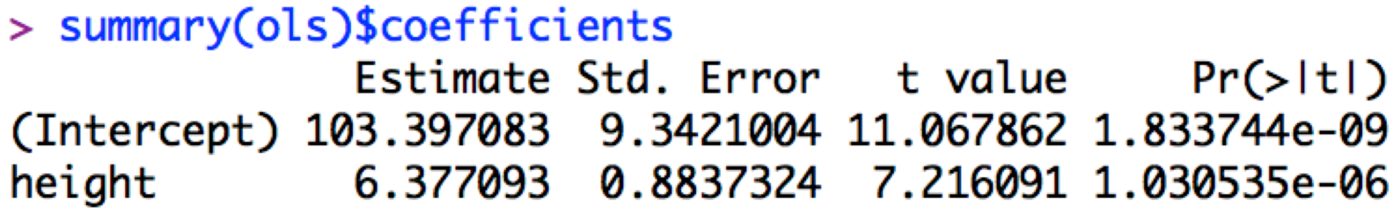
NB: This package was designed for LaTeX, but works with excel as well. To export to a .csv, be sure to have csv = T

Example:

esttab(filename = "/Users/adrianrohitdass/Google Drive/Introduction to R/Examples Using Data/out", csv=T)

Extracting Beta estimates, standard errors, etc. from model

- A couple of ways to do this, but most of the information we're after is stored in the coefficients object returned from summary:



```
> summary(ols)$coefficients
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 103.397083  9.3421004 11.067862 1.833744e-09
height        6.377093  0.8837324  7.216091 1.030535e-06
```

- The above is a matrix, so we can get the information we need through column extractions:
  - Beta coefficients: summary(ols)$coefficients[,1]
  - Standard errors: summary(ols)$coefficients[,2]
  - T-value: summary(ols)$coefficients[,3]
  - P-value: summary(ols)$coefficients[,4]

Models for Binary Outcomes

- R does not come with different programs for binary outcomes. Instead, it utilizes a unifying framework of generalized linear models (GLMs) and a single fitting function, glm() (Kleiber & Zeileis (2008))

Package: stats

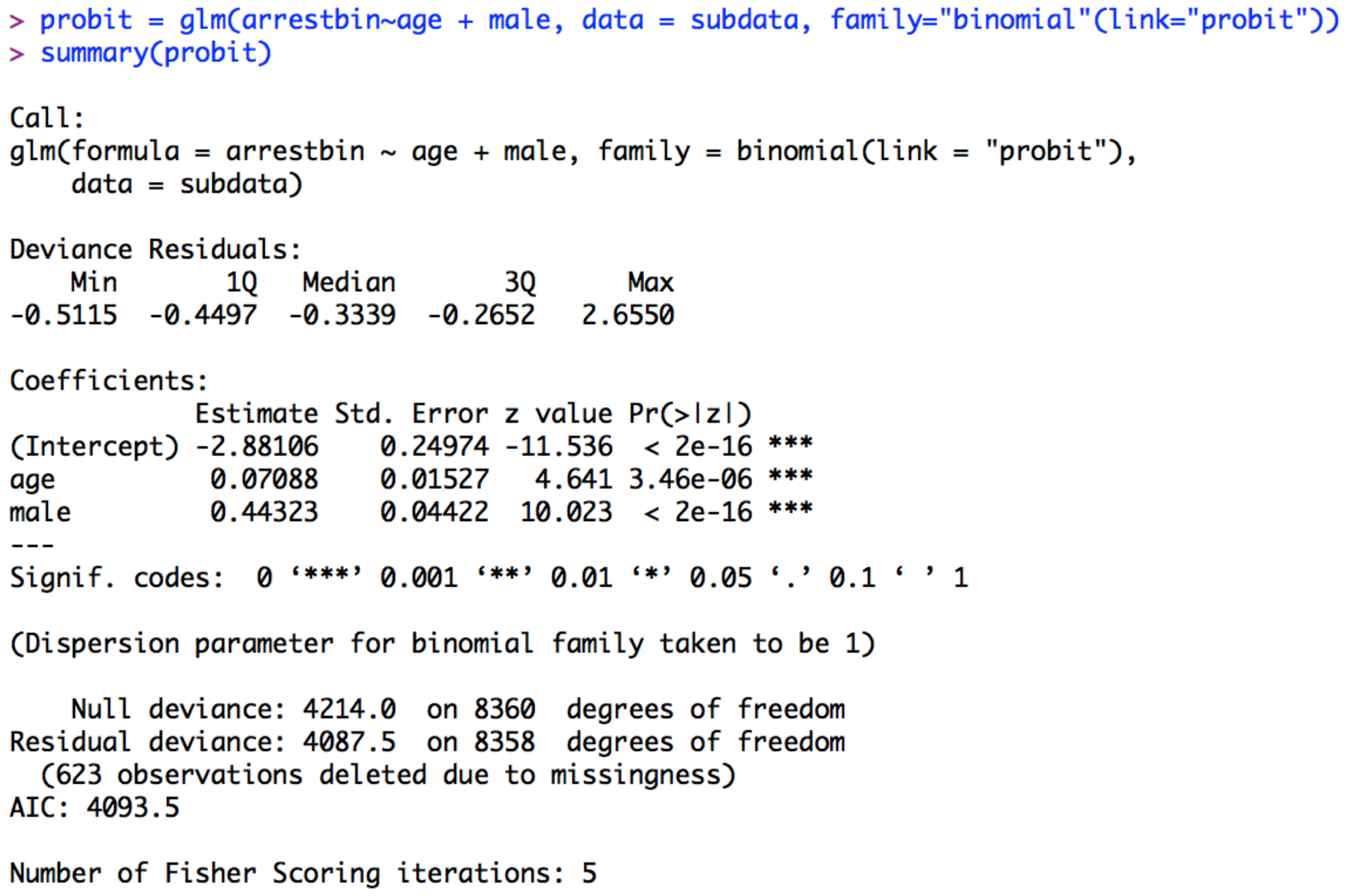
Formula: glm(formula, family = gaussian, data, weights, subset, na.action, start = NULL, etastart, mustart, offset, control = list(...), model = TRUE, method = "glm.fit”, x = FALSE, y = TRUE, contrasts = NULL, ...)

- For binary outcomes, we specify family=“binomial” and link= “logit” or “probit”
- Can be extended to count data as well (family=“poisson”)

Online help: http://www.inside-r.org/r-doc/stats/glm

Models for Binary Outcomes

Example: Probit Analysis: factors associated with being arrested



```
> probit = glm(arrestbin~age + male, data = subdata, family="binomial"(link="probit"))
> summary(probit)

Call:
glm(formula = arrestbin ~ age + male, family = binomial(link = "probit"),
    data = subdata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.5115  -0.4497  -0.3339  -0.2652   2.6550

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.88106    0.24974 -11.536  < 2e-16 ***
age          0.07088    0.01527   4.641 3.46e-06 ***
male         0.44323    0.04422  10.023  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4214.0  on 8360  degrees of freedom
Residual deviance: 4087.5  on 8358  degrees of freedom
  (623 observations deleted due to missingness)
AIC: 4093.5

Number of Fisher Scoring iterations: 5
```

Instrumental Variables

A way to obtain a consistent estimator of the unknown co-efficiencts of the population regression function when the regressor, X, is correlated with the error term, *u*. (Stock and Watson, 2007).

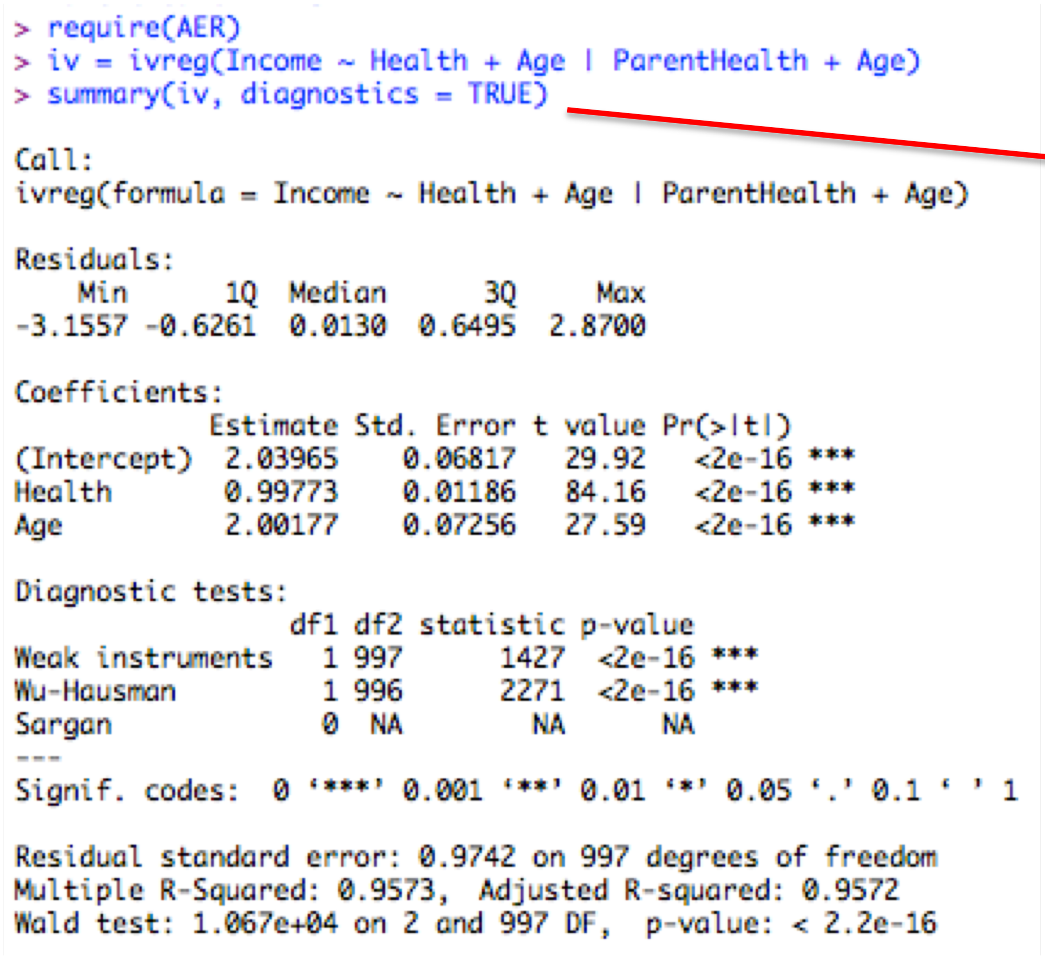
Package: AER

Formula: ivreg(formula, instruments, data, subset, na.action, weights, offset, contrasts = NULL, model = TRUE, y = TRUE, x = FALSE, ...)

Online documentation: https://cran.r-project.org/web/packages/AER/AER.pdf

IV Example

Example: Determinants of Income (As a function of Health)



```
> require(AER)
> iv = ivreg(Income ~ Health + Age | ParentHealth + Age)
> summary(iv, diagnostics = TRUE)

Call:
ivreg(formula = Income ~ Health + Age | ParentHealth + Age)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.1557 -0.6261  0.0130  0.6495  2.8700 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.03965    0.06817   29.92   <2e-16 ***
Health       0.99773    0.01186   84.16   <2e-16 ***
Age          2.00177    0.07256   27.59   <2e-16 ***

Diagnostic tests:
                 df1 df2 statistic p-value    
Weak instruments   1 997      1427  <2e-16 ***
Wu-Hausman         1 996      2271  <2e-16 ***
Sargan             0  NA        NA      NA    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.9742 on 997 degrees of freedom
Multiple R-Squared: 0.9573,  Adjusted R-squared: 0.9572 
Wald test: 1.067e+04 on 2 and 997 DF,  p-value: < 2.2e-16
```

Prints out F-test for Weak Instruments, Hausman Test Statistic (vs ols) and Sargan's Test for Over-identifying Restrictions (if more than one instrument use)

Other Regression Models

- Panel Data Econometrics
  - Package: plm
  - https://cran.r-project.org/web/packages/plm/vignettes/plm.pdf
- Linear and Generalized Linear Mixed Effects Models
  - Package: lme4
  - https://cran.r-project.org/web/packages/lme4/lme4.pdf
- Quantile Regression
  - Package: quantreg
  - https://cran.r-project.org/web/packages/quantreg/quantreg.pdf

<u>Part III</u>

Other topics in R

Parallel Processing in R

- Central Processing Unit (CPU): Carries out instructions of a computer program
- Parallel computing: From Wikipedia: “Parallel computing is a type of computation in which many calculations or the execution of processes are carried out simultaneously. Large problems can often be divided into smaller ones, which can then be solved at the same time.”
  - See here for more: https://en.wikipedia.org/wiki/Parallel_computing
- Modern day computers typically contain:
  - Single-core
  - Dual-core
  - Quad-core
- May also contain hyperthreading

Parallel Processing in R (Continued)

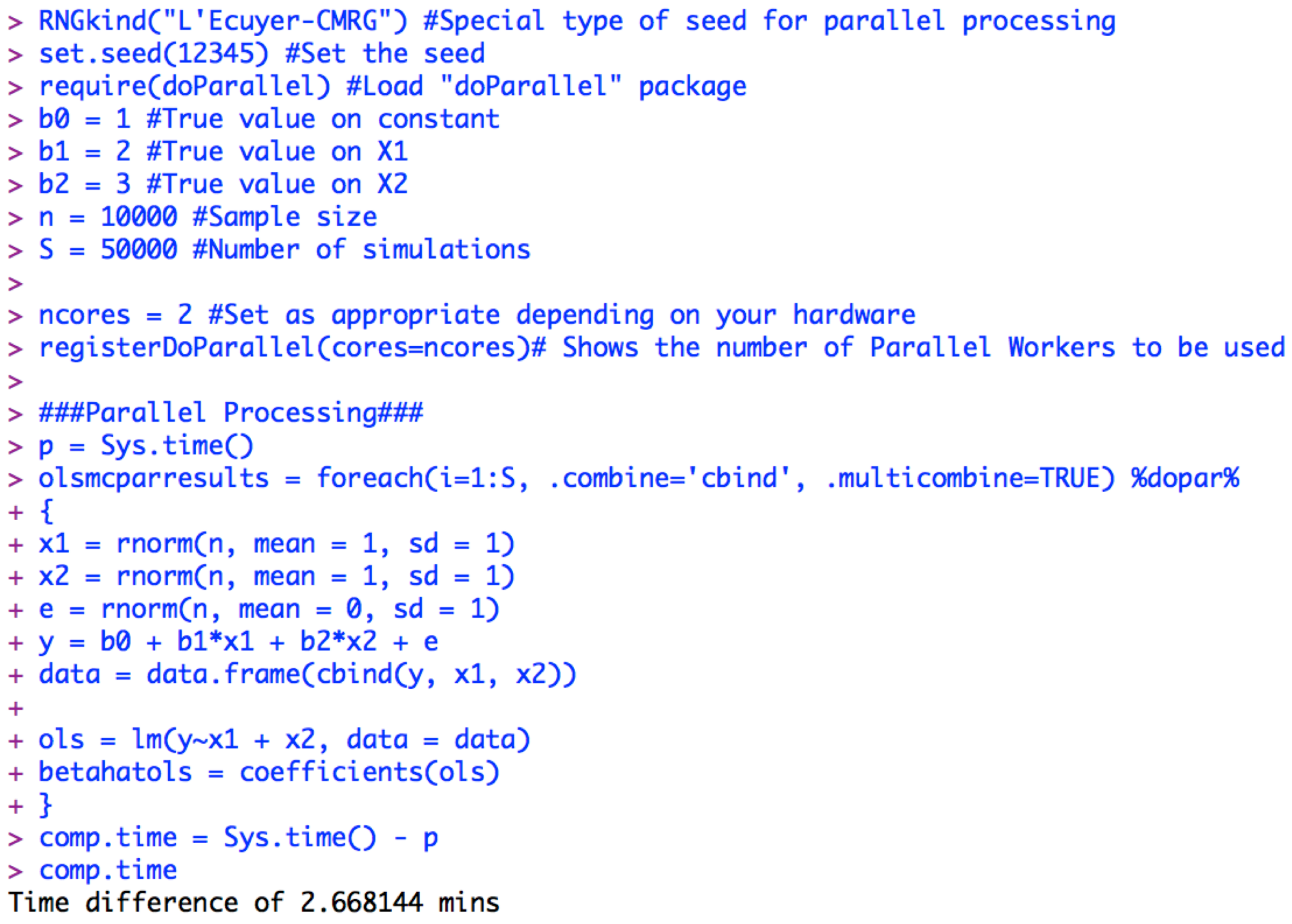
- Parallel processing can be used in many situations, including:
  - Bootstrapping
  - Microsimulation models
  - Monte Carlo experiments
- By utilizing parallel processing, we can significantly speed up the processing time of our calculations

Parallel Processing in R (Continued)

- There are many packages to perform parallel processing in R, including
- parallel
  - Available in R by default
  - Handles large chunks of computations in parallel
  - https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

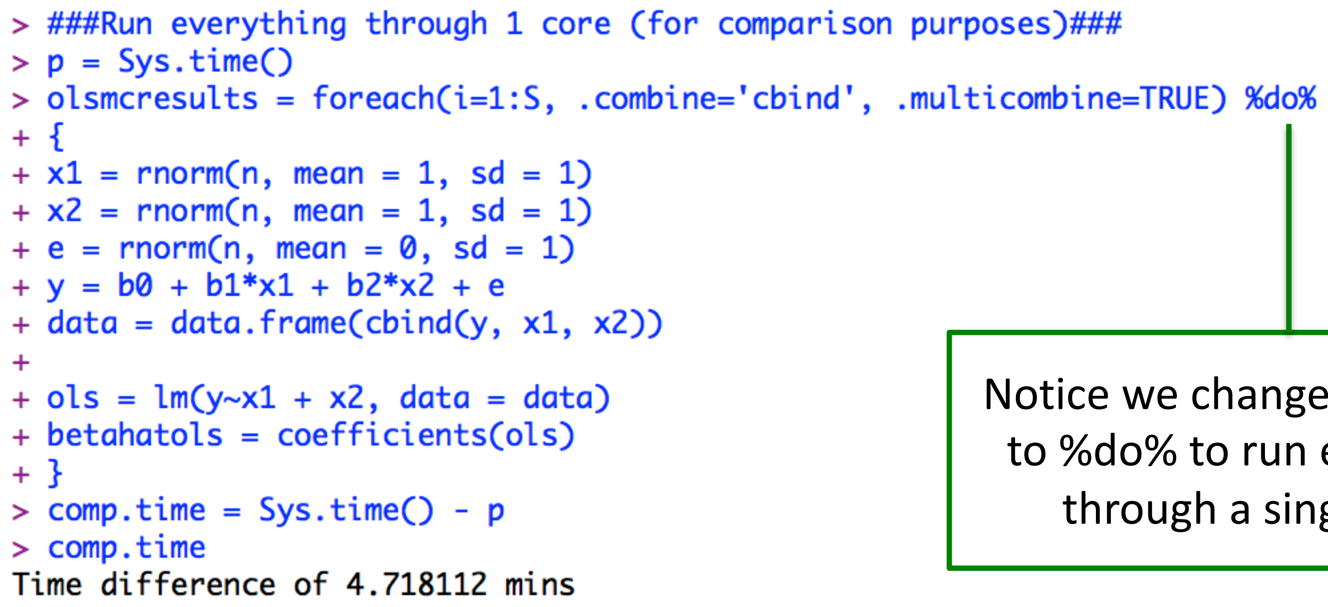
- doParallel
  - "parallel backend" for the "foreach" package
  - provides a mechanism needed to execute foreach loops in parallel
  - https://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf

Example: Monte Carlo Experiment



```
> RNGkind("L'Ecuyer-CMRG") #Special type of seed for parallel processing
> set.seed(12345) #Set the seed
> require(doParallel) #Load "doParallel" package
> b0 = 1 #True value on constant
> b1 = 2 #True value on X1
> b2 = 3 #True value on X2
> n = 10000 #Sample size
> S = 50000 #Number of simulations
>
> ncores = 2 #Set as appropriate depending on your hardware
> registerDoParallel(cores=ncores)# Shows the number of Parallel Workers to be used
>
> ###Parallel Processing###
> p = Sys.time()
> olsmcparresults = foreach(i=1:S, .combine='cbind', .multicombine=TRUE) %dopar%
+ {
+ x1 = rnorm(n, mean = 1, sd = 1)
+ x2 = rnorm(n, mean = 1, sd = 1)
+ e = rnorm(n, mean = 0, sd = 1)
+ y = b0 + b1*x1 + b2*x2 + e
+ data = data.frame(cbind(y, x1, x2))
+
+ ols = lm(y~x1 + x2, data = data)
+ betahatols = coefficients(ols)
+ }
> comp.time = Sys.time() - p
> comp.time
Time difference of 2.668144 mins
```

Example: Monte Carlo Experiment (Continued)



```
> ###Run everything through 1 core (for comparison purposes)###
> p = Sys.time()
> olsmcresults = foreach(i=1:S, .combine='cbind', .multicombine=TRUE) %do%
+ {
+ x1 = rnorm(n, mean = 1, sd = 1)
+ x2 = rnorm(n, mean = 1, sd = 1)
+ e = rnorm(n, mean = 0, sd = 1)
+ y = b0 + b1*x1 + b2*x2 + e
+ data = data.frame(cbind(y, x1, x2))
+
+ ols = lm(y~x1 + x2, data = data)
+ betahatols = coefficients(ols)
+ }
> comp.time = Sys.time() - p
> comp.time
Time difference of 4.718112 mins
```

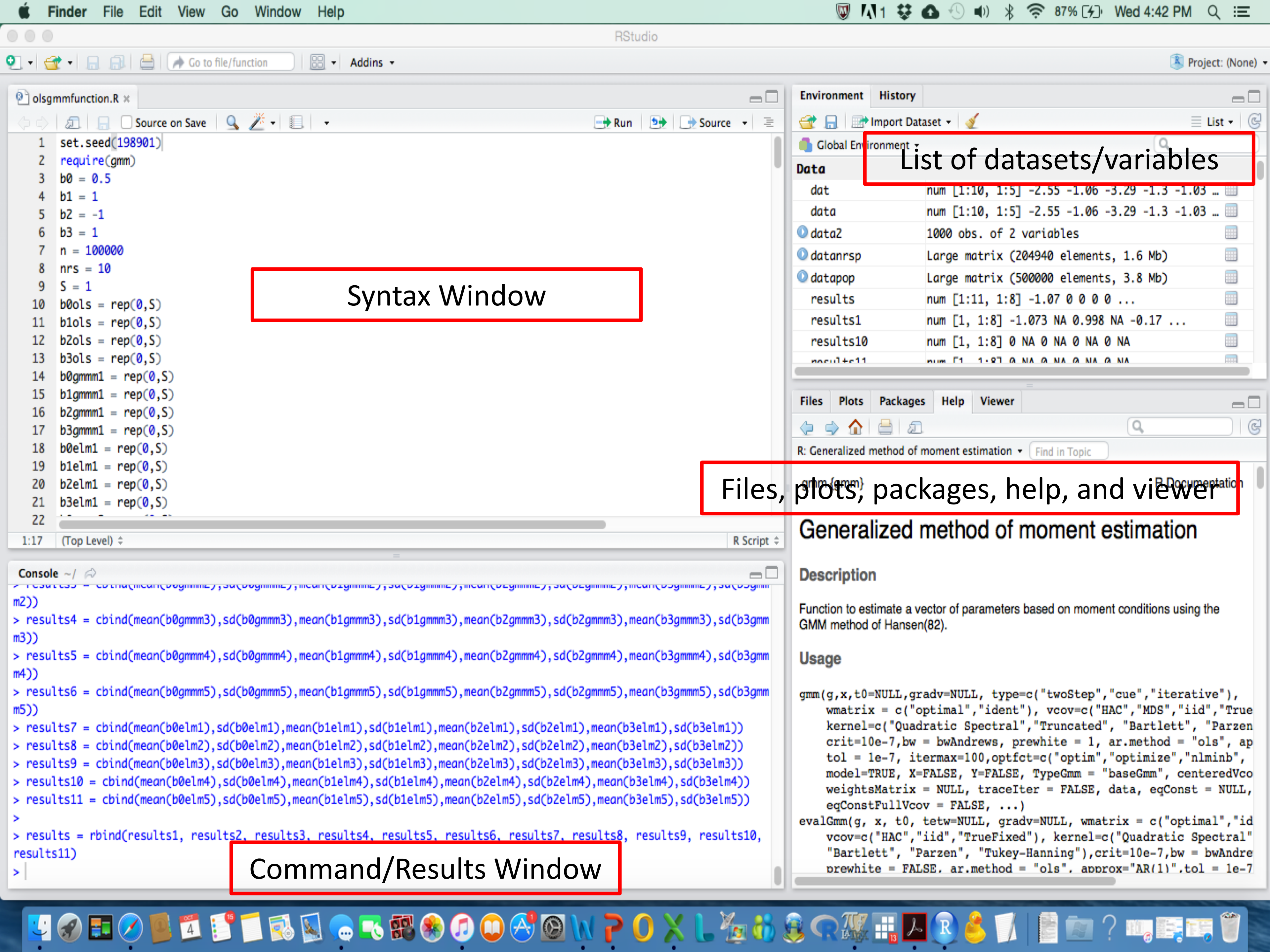
Notice we changed %dopar% to %do% to run everything through a single core

R Studio

What is R Studio?

From R Studio Website:

- An integrated development environment (IDE) for R. Includes:
  - A console
  - Syntax highlighting editor
  - Tools for plotting, history, debugging, and workspace history
- Can think of it as a more user friendly version of R
- Open Source Edition available free of charge
- For more information, see https://www.rstudio.com



```
set.seed(198901)
require(gmm)
b0 = 0.5
b1 = 1
b2 = -1
b3 = 1
n = 100000
nrs = 10
S = 1
b0ols = rep(0,S)
b1ols = rep(0,S)
b2ols = rep(0,S)
b3ols = rep(0,S)
b0gmmm1 = rep(0,S)
b1gmmm1 = rep(0,S)
b2gmmm1 = rep(0,S)
b3gmmm1 = rep(0,S)
b0elm1 = rep(0,S)
b1elm1 = rep(0,S)
b2elm1 = rep(0,S)
b3elm1 = rep(0,S)
```

Data

dat	num [1:10, 1:5] -2.55 -1.06 -3.29 -1.3 -1.03 …
data	num [1:10, 1:5] -2.55 -1.06 -3.29 -1.3 -1.03 …
data2	1000 obs. of 2 variables
datanrsp	Large matrix (204940 elements, 1.6 Mb)
datapop	Large matrix (500000 elements, 3.8 Mb)
results	num [1:11, 1:8] -1.07 0 0 0 0 ...
results1	num [1, 1:8] -1.073 NA 0.998 NA -0.17 ...
results10	num [1, 1:8] 0 NA 0 NA 0 NA 0 NA

```
m2))
> results4 = cbind(mean(b0gmmm3),sd(b0gmmm3),mean(b1gmmm3),sd(b1gmmm3),mean(b2gmmm3),sd(b2gmmm3),mean(b3gmmm3),sd(b3gmm
m3))
> results5 = cbind(mean(b0gmmm4),sd(b0gmmm4),mean(b1gmmm4),sd(b1gmmm4),mean(b2gmmm4),sd(b2gmmm4),mean(b3gmmm4),sd(b3gmm
m4))
> results6 = cbind(mean(b0gmmm5),sd(b0gmmm5),mean(b1gmmm5),sd(b1gmmm5),mean(b2gmmm5),sd(b2gmmm5),mean(b3gmmm5),sd(b3gmm
m5))
> results7 = cbind(mean(b0elm1),sd(b0elm1),mean(b1elm1),sd(b1elm1),mean(b2elm1),sd(b2elm1),mean(b3elm1),sd(b3elm1))
> results8 = cbind(mean(b0elm2),sd(b0elm2),mean(b1elm2),sd(b1elm2),mean(b2elm2),sd(b2elm2),mean(b3elm2),sd(b3elm2))
> results9 = cbind(mean(b0elm3),sd(b0elm3),mean(b1elm3),sd(b1elm3),mean(b2elm3),sd(b2elm3),mean(b3elm3),sd(b3elm3))
> results10 = cbind(mean(b0elm4),sd(b0elm4),mean(b1elm4),sd(b1elm4),mean(b2elm4),sd(b2elm4),mean(b3elm4),sd(b3elm4))
> results11 = cbind(mean(b0elm5),sd(b0elm5),mean(b1elm5),sd(b1elm5),mean(b2elm5),sd(b2elm5),mean(b3elm5),sd(b3elm5))
>
> results = rbind(results1, results2, results3, results4, results5, results6, results7, results8, results9, results10,
results11)
>
```

R: Generalized method of moment estimation

gmm {gmm} R Documentation

Generalized method of moment estimation

Description

Function to estimate a vector of parameters based on moment conditions using the GMM method of Hansen(82).

Usage

```
gmm(g,x,t0=NULL,gradv=NULL, type=c("twoStep","cue","iterative"),
    wmatrix = c("optimal","ident"), vcov=c("HAC","MDS","iid","True
    kernel=c("Quadratic Spectral","Truncated", "Bartlett", "Parzen
    crit=10e-7,bw = bwAndrews, prewhite = 1, ar.method = "ols", ap
    tol = 1e-7, itermax=100,optfct=c("optim","optimize","nlminb",
    model=TRUE, X=FALSE, Y=FALSE, TypeGmm = "baseGmm", centeredVco
    weightsMatrix = NULL, traceIter = FALSE, data, eqConst = NULL,
    eqConstFullVcov = FALSE, ...)
evalGmm(g, x, t0, tetw=NULL, gradv=NULL, wmatrix = c("optimal","id
    vcov=c("HAC","iid","TrueFixed"), kernel=c("Quadratic Spectral"
    "Bartlett", "Parzen", "Tukey-Hanning"),crit=10e-7,bw = bwAndre
    prewhite = FALSE, ar.method = "ols", approx="AR(1)",tol = 1e-7
```

Conclusions

- R has extremely powerful database management capabilities
  - Is fully capable of performing the same sort of tasks as commercial software programs
- R is very capable of statistical analysis
  - Is fully capable of calculating summary statistics and performing regression analysis right out of the box
  - Can install additional packages to perform other sorts of analysis, depending on the research question of the user
  - Performance can be improved by the use of parallel processing
- R, and the additional packages available to enhance the use of R, are available <u>free of charge</u>

R Resources

R Online Resources

- A list of R packages is contained here:

https://cran.r-project.org/web/packages/available_packages_by_date.html

- By clicking on a particular package, you'll be taken to a page with more details, as well as a link to download the documation
- Typing help(topic) in R pulls up a brief help file with synax and examples, but the online manuals contain more detail

R Online Resources

UCLA Institute for Digital Research and Education

- List of topics and R resources (getting started, data examples, etc.) can be found here:

http://www.ats.ucla.edu/stat/r/

Other R Resources

1. Kleiber, C., & Zeileis, A. (2008). *Applied econometrics with R*. Springer Science & Business Media.
   - Great reference for the applied researcher wanting to use R for econometric analysis. Includes R basics, linear regression model, panel data models, binary outcomes, etc.
2. CRAN Task View: Econometrics
   - A listing of the statistical models used in econometrics, as well as the R package(s) needed to perform them. Available at: https://cran.r-project.org/view=Econometrics

Thanks for Listening

Good luck with R!