

An Introduction to Using

Dino Christenson & Scott Powell

Ohio State University

November 20, 2007

Introduction to R Outline

- I. What is R?
- II. Why use R?
- III. Where to get R?
- IV. GUI & scripts
- V. Objects in R
- VI. Matrices in R
- VII. Reading datasets in R

What is R?

- “R is a language and environment for statistical computing and graphics.”
- Software used for data manipulation, data analysis, and pretty graphical output
- Elements of the “environment”: programming language, run-time environment, graphics, and a debugger
- Bottom Line: It’s a statistics package.

Why use R?

- Flexibility
 - Design based on computer language (similar to S)
 - No reliance on preexisting tools/functions
 - Users can program their own code
 - Packages
- Flexibility is well suited to statistical simulation

Why use R?

- Graphical capabilities
 - Publication quality
 - High degree of manipulation
- Highly Interactive – User has to know what’s going on “under the hood”
- It’s Free
- All the kids are doing it

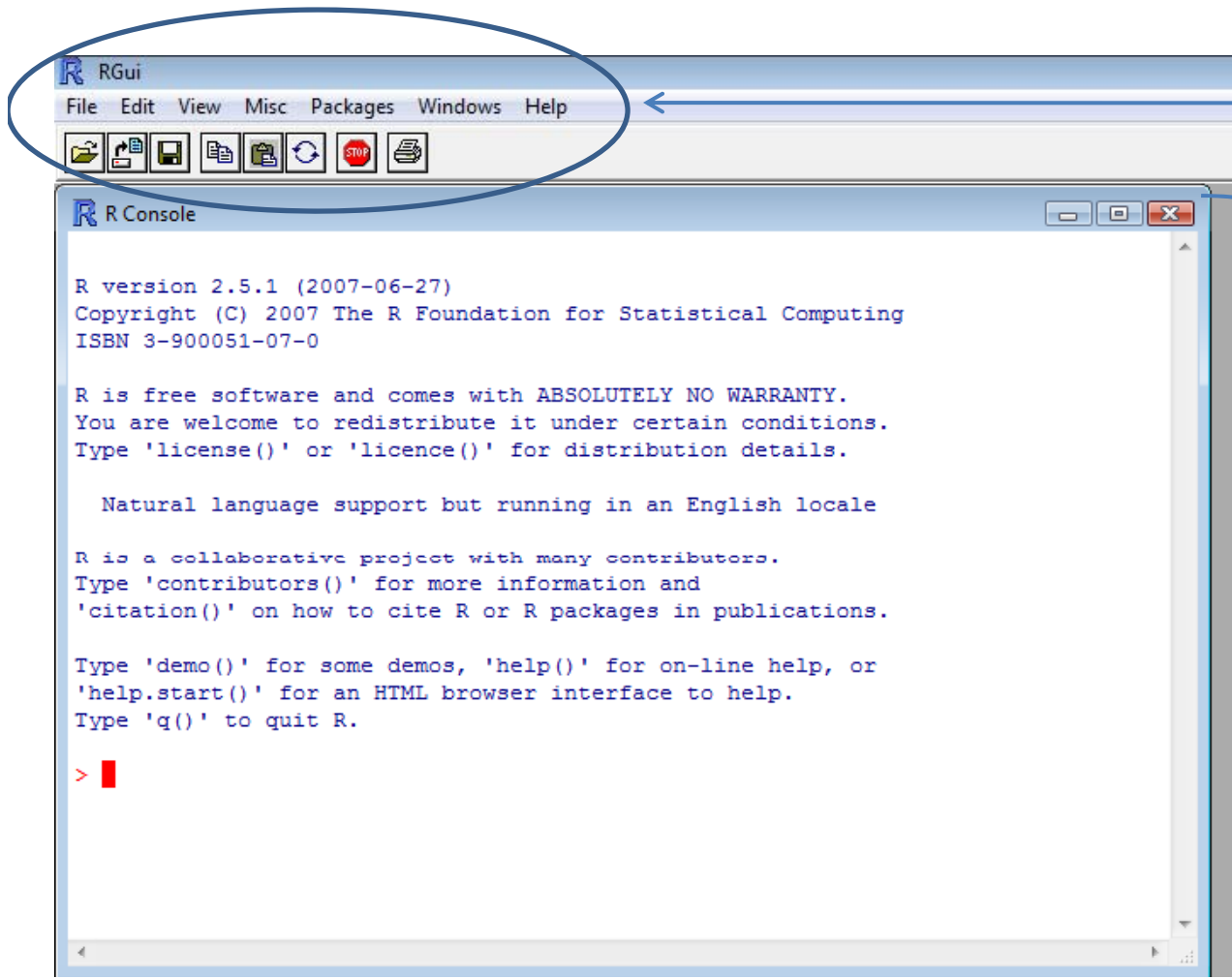
Why NOT use R?

- Data Management
 - Manipulation of data can be very cumbersome
 - Example: TSCS functions in Stata
- Start-Up Costs
 - It takes time to learn R
 - Need to be familiar with code and matrices

Where to get R?

- The R Project web page
 - <http://www.r-project.org/>
- Downloading the software
 - Pick a mirror and download
- Downloading packages
 - New packages available both randomly on the internet and at the site

R's GUI



R's GUI

- Allows you to interact with R using graphical icons, as opposed to pure commands

- However R is primarily command driven

R's Console

- Type your commands

- Receive your results

- Graphs are opened in new window

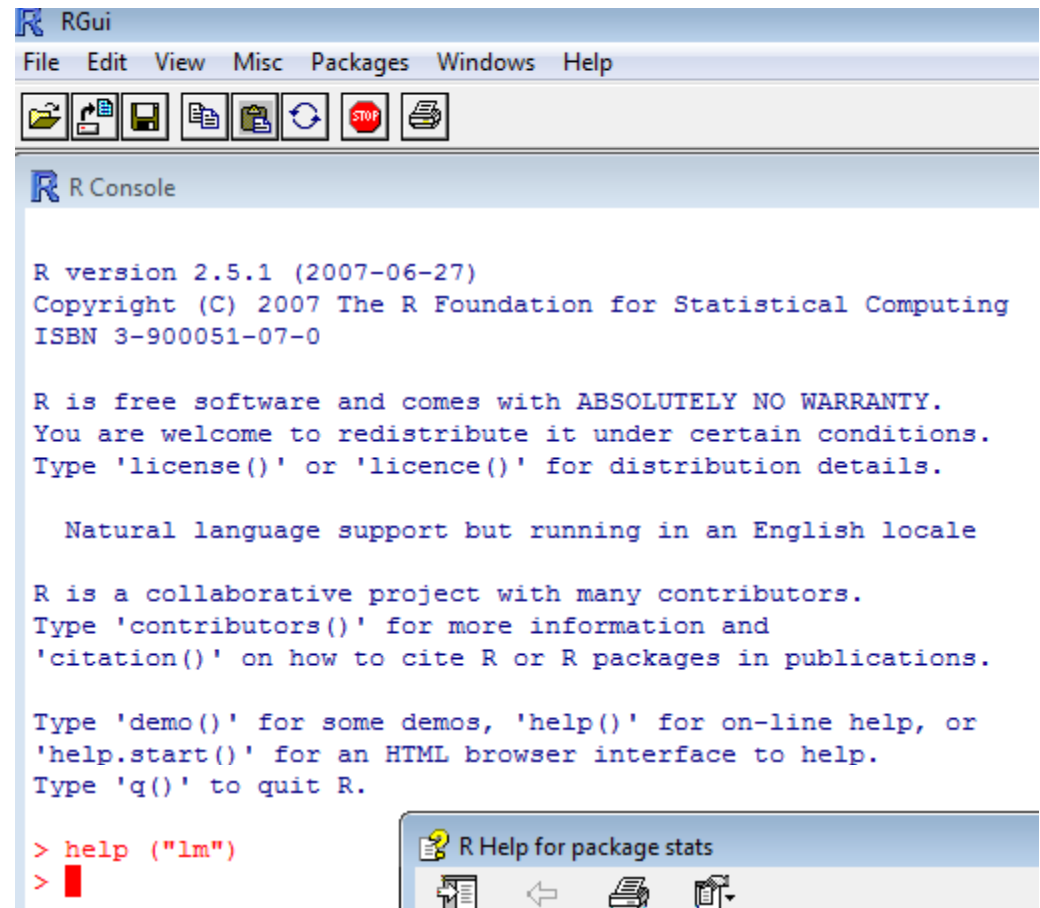
R's GUI

- R's GUI is very limited:
 - File: open, load, print and set working directory
 - Edit: copy, save and select
 - Packages: install and update
 - Help: functions (very helpful, sometimes)
 - Eg. Go to Help -> R fuctions -> (type) lm
 - A helpful guide on linear models is displayed



A Note on GUI

- R is command driven
- There isn't much you can do with a button that you cannot do with a command, if anything
- For eg, we could also get help on the lm function by typing `help ("lm")` in the console →



The screenshot shows the RGui window with a menu bar (File, Edit, View, Misc, Packages, Windows, Help) and a toolbar. Below the toolbar is the R Console, which displays the following text:

```
R version 2.5.1 (2007-06-27)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

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

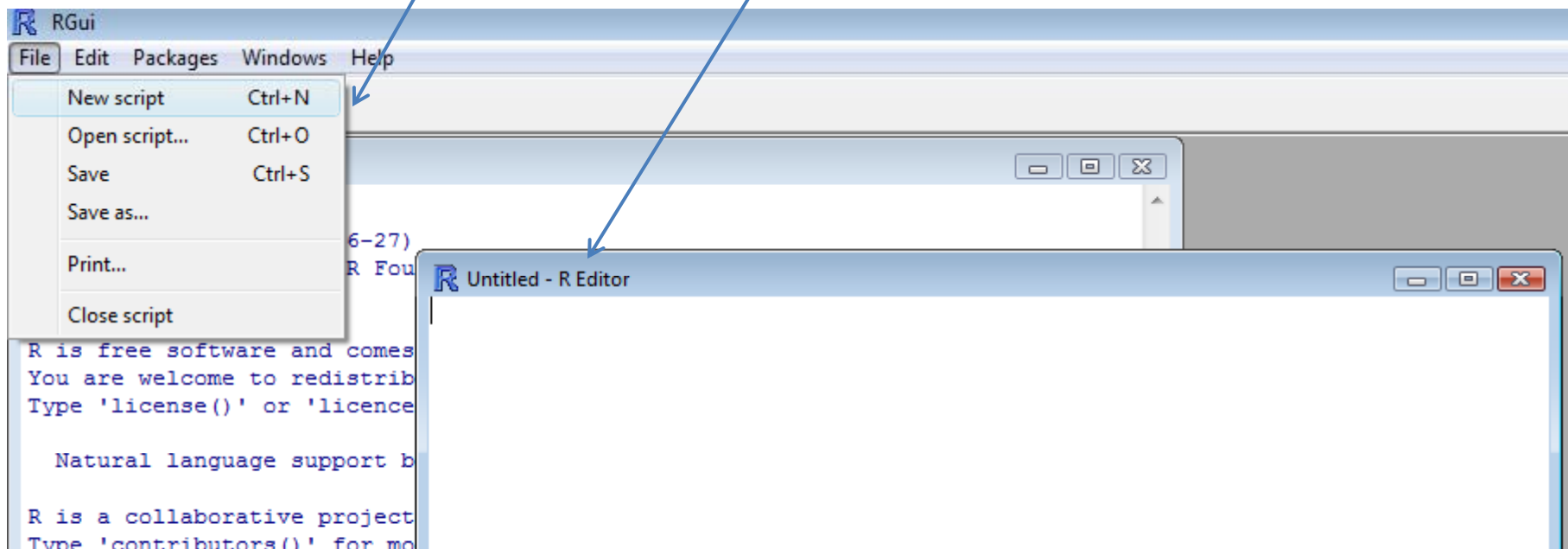
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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.
```

At the bottom of the console, the command `> help ("lm")` has been entered, and a red cursor is visible on the next line. To the right of the console, a small window titled "R Help for package stats" is open, showing a help page with a toolbar.

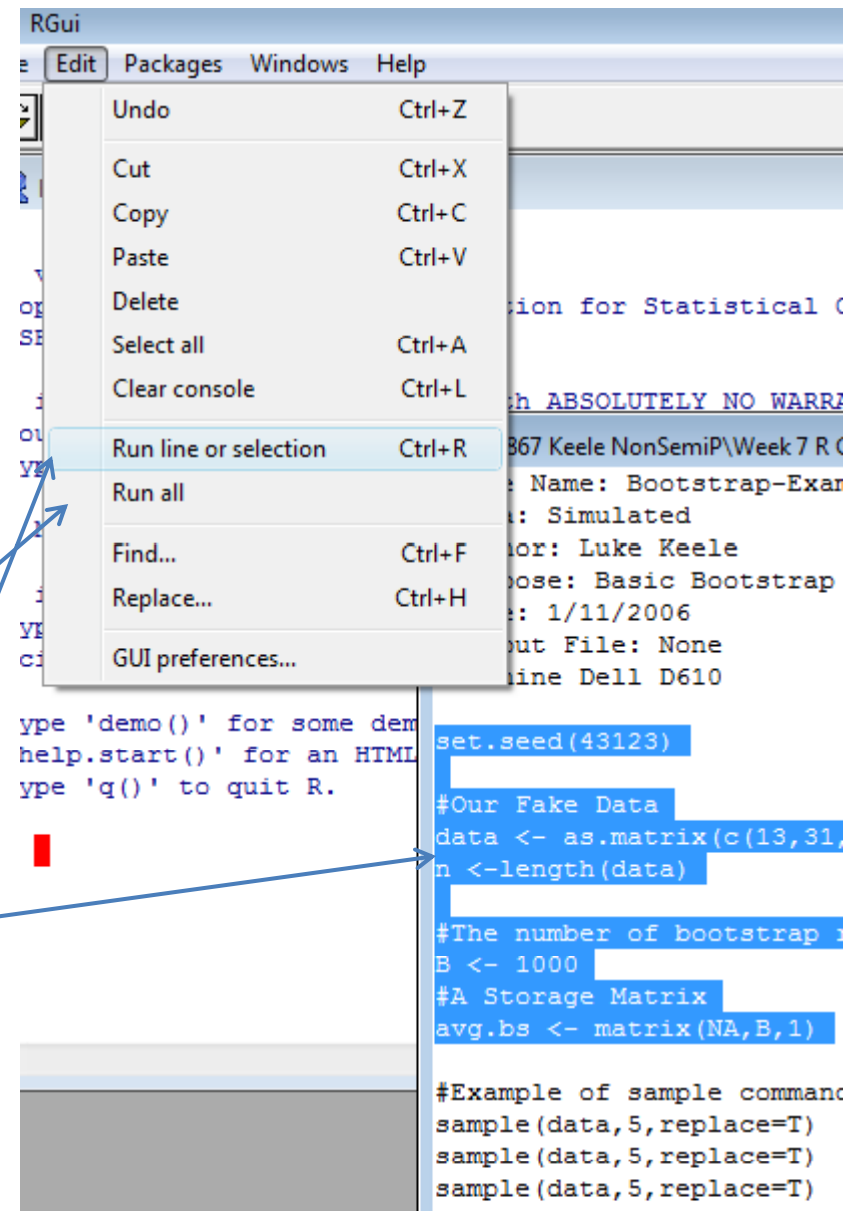
R Script

- Beyond typing directly to the console, R allows you to keep track of all your commands in a text document called a “script”
- Starting a new script is easy: File → New script
 - A new window opens: the “R Editor”



R Script

- Treat the editor like a txt editor
 - Save it periodically
 - Annotate with ‘#’
- After inputting your commands
 - You can run all or select some of the commands to run from the script



R Script

- You can find all the examples from this presentation on the aforementioned script
 - I:\PRISM\Brownbags\Intro to R\introR_V07.txt
- If you are working along
 - Copy the script onto your personal drive
 - Go to File → Open script
 - Browse in your folders for the script
 - Select it
 - It opens in a new window

Working Directories in R

- R may write over previous R output if you do not specify appropriate working directories
 - So we need to establish a particular folder in which to work from and save our output to each time
- Syntax procedure: in the console or the editor
 - `setwd("K:\PRISM\Brownbags")`
- GUI procedure: drop-down menus
 - For PCs
 - Go to File → Change working directory
 - Browse for the folder of your choosing
 - For Mac Users (who are super cool, btw)
 - Go to Misc → Change working directory
 - Select/create the folder for this project
- Thus this new directory will have your data as well any output created from R

Objects in R

- R is based on objects: vectors & matrices
- When entering commands
 - Expressions and commands are case-sensitive
 - Anything following the pound symbol (#) is treated as a comment and ignored by R
 - An object name must start with an alphabetical character but may contain numbers and periods thereafter
 - Arrow keys allow you to scroll through previous commands at the prompt
- Note: for this presentation all R syntax will be in Courier New font

Objects in R

- The basic R format for commands
 - `object.name <- command(options)`
 - `object.name = command(options)`
 - Note: `=` and `<-` equivalent after R1.4.0#
 - Pick one and stick with it
- So
 - The arrow function defines the object (call it `any.name`)
 - Canned operations identified by the parentheses
 - Command options identified by what's within the parentheses
 - Results are returned with a numeric indicator of the data frame, eg `[1]` if it is a vector

Objects in R

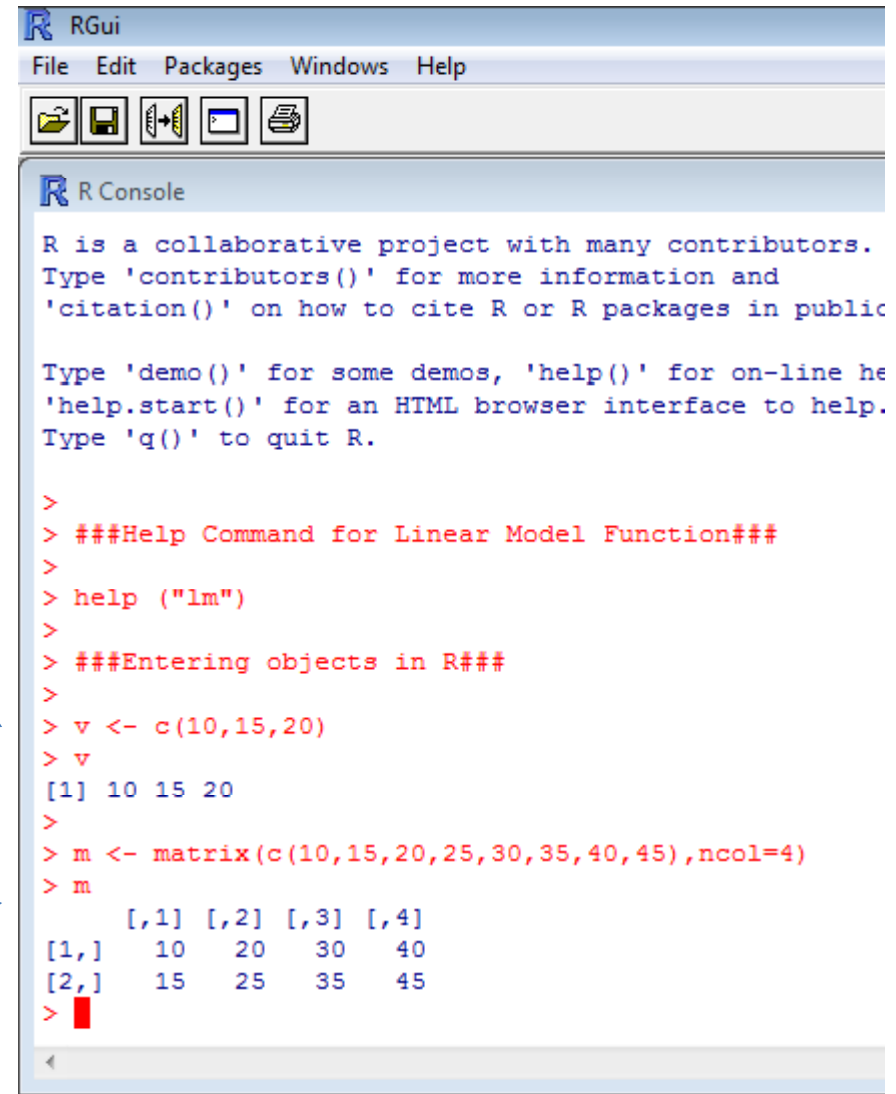
- Before jumping into a large dataset, let's create some simple objects in R

- Vector

- `v <- c(10, 15, 20)`
- `v`

- Matrices

- `m <- matrix(c(10, 15, 20, 25, 30, 35, 40, 45), ncol=4)`
- `m`



The screenshot shows the RGui window with the R Console. The console displays the following text:

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in public

Type 'demo()' for some demos, 'help()' for on-line help
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
> ###Help Command for Linear Model Function###
>
> help("lm")
>
> ###Entering objects in R###
>
> v <- c(10,15,20)
> v
[1] 10 15 20
>
> m <- matrix(c(10,15,20,25,30,35,40,45),ncol=4)
> m
      [,1] [,2] [,3] [,4]
[1,]  10   20   30   40
[2,]  15   25   35   45
> █
```

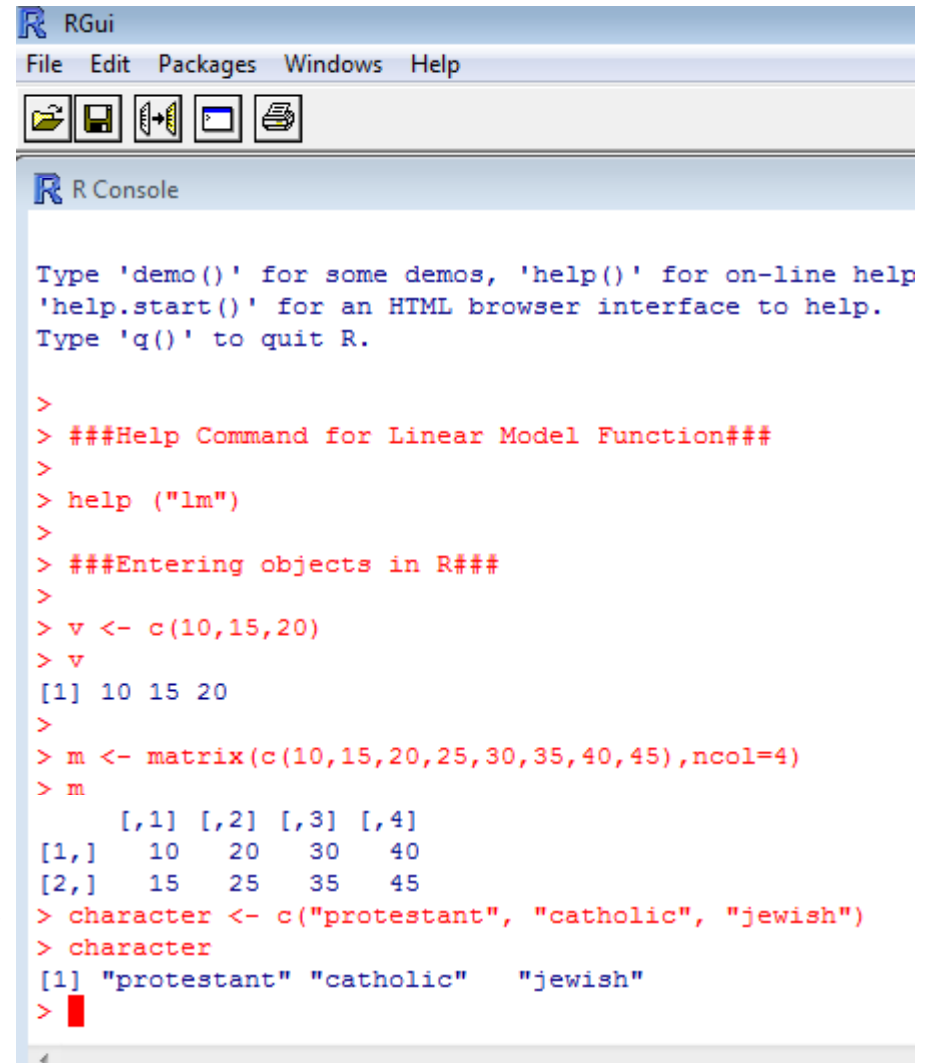
Two blue arrows point from the code in the text to the corresponding lines in the console output. One arrow points from `v` to the output of `v`, and another points from `matrix(...)` to the output of `m`.

Objects in R

- Beyond numerical vectors, we can also do character or logic vectors

– A character vector

- `character <- c("protestant", "catholic", "jewish")`
- `character`



```
RGui
File Edit Packages Windows Help

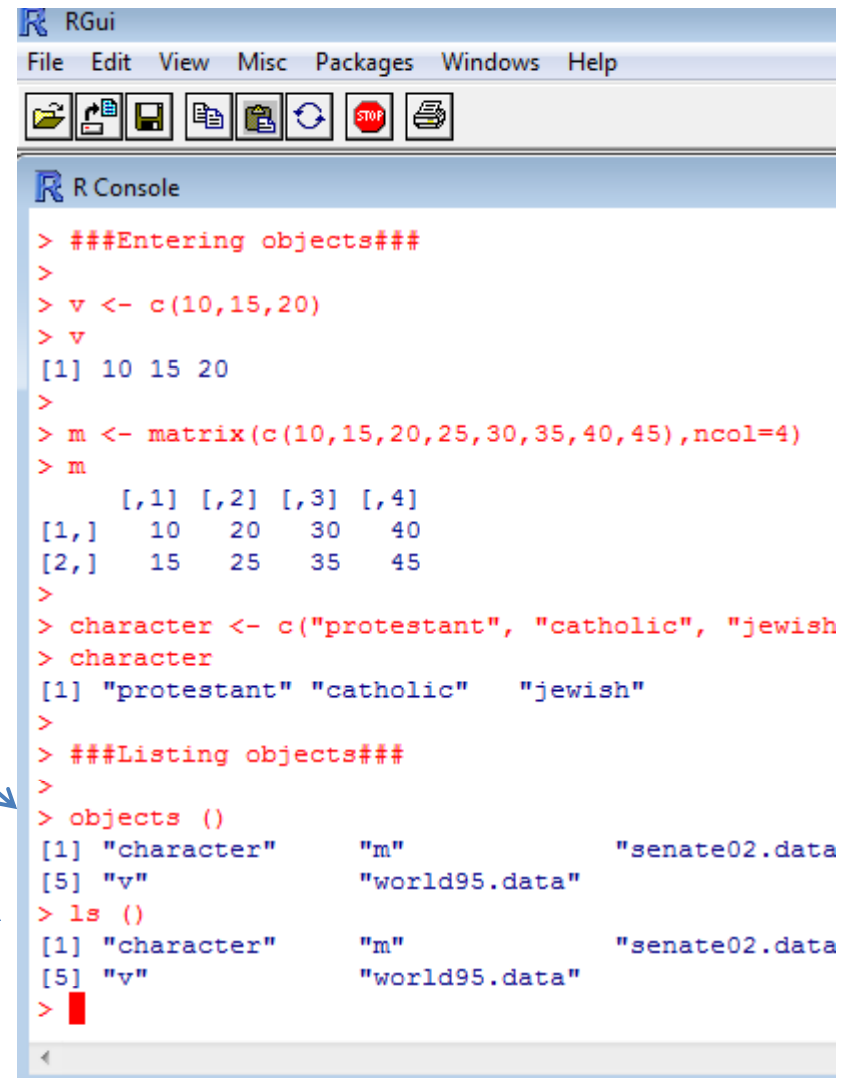
R Console

Type 'demo()' for some demos, 'help()' for on-line help
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
> ###Help Command for Linear Model Function###
>
> help ("lm")
>
> ###Entering objects in R###
>
> v <- c(10,15,20)
> v
[1] 10 15 20
>
> m <- matrix(c(10,15,20,25,30,35,40,45),ncol=4)
> m
      [,1] [,2] [,3] [,4]
[1,]  10  20  30  40
[2,]  15  25  35  45
> character <- c("protestant", "catholic", "jewish")
> character
[1] "protestant" "catholic"  "jewish"
> █
```

Objects in R

- So you've created a couple of objects
- How do you see what objects you have?
 - `objects()`
 - `ls()`
- Objects will remain until they are removed
- To remove an object
 - `rm(object_name)`



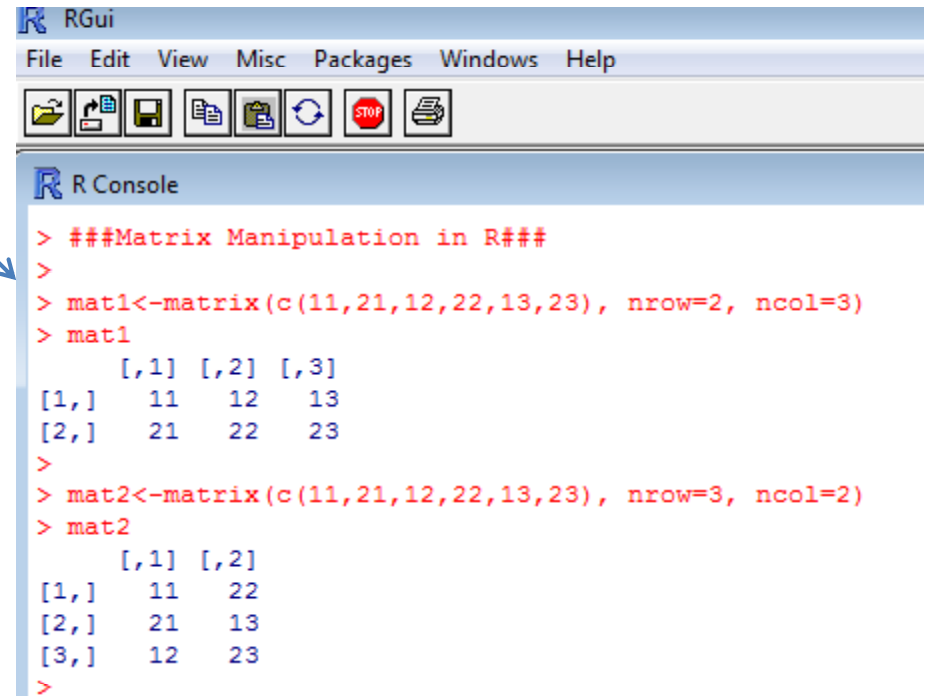
```
RGui
File Edit View Misc Packages Windows Help
[Icons]
R Console
> ###Entering objects###
>
> v <- c(10,15,20)
> v
[1] 10 15 20
>
> m <- matrix(c(10,15,20,25,30,35,40,45),ncol=4)
> m
      [,1] [,2] [,3] [,4]
[1,]  10   15  20   25
[2,]  15   20  25   30
[3,]  20   25  30   35
[4,]  25   30  35   40
[5,]  30   35  40   45
[6,]  35   40  45   45
[7,]  40   45  45   45
[8,]  45   45  45   45
>
> character <- c("protestant", "catholic", "jewish")
> character
[1] "protestant" "catholic"   "jewish"
>
> ###Listing objects###
>
> objects ()
[1] "character"      "m"           "senate02.data"
[5] "v"              "world95.data"
> ls ()
[1] "character"      "m"           "senate02.data"
[5] "v"              "world95.data"
>
<
```

Matrices in R

- Thus our objects are really vectors and matrices in R
 - How R handles matrices is key to understanding how R can work for you
 - Allows us to calculate coefficients, std errors and t scores...etc.
- So let's try creating a few more matrices for practice
 - As we saw above, `matrix` turns a distribution of values into a matrix of n rows and k columns

Matrices in R

- `mat1<-matrix(c(11,21,12,22,13,23), nrow=2, ncol=3)`
- `mat1`
 - This gives you a 2x3 dimensional array of the numbers and placements you specified above
 - R reads by row first taking the first two numbers as row 1 and row 2 then starting a new column with the next two and so on...
- What happens when you reverse the row and column dimensions?



```
RGui
File Edit View Misc Packages Windows Help
[Icons]
R Console
> ###Matrix Manipulation in R###
>
> mat1<-matrix(c(11,21,12,22,13,23), nrow=2, ncol=3)
> mat1
      [,1] [,2] [,3]
[1,]  11  12  13
[2,]  21  22  23
>
> mat2<-matrix(c(11,21,12,22,13,23), nrow=3, ncol=2)
> mat2
      [,1] [,2]
[1,]  11  22
[2,]  21  13
[3,]  12  23
>
```

Matrices in R

- With larger datasets we may want to know the dimensions of the data
 - `dim(mat1)` gives you the $n \times k$ dimensions
 - `ncol(mat1)` the columns
 - `nrow(mat1)` the rows

```
R Console
      [,1] [,2] [,3]
[1,]  11  12  13
[2,]  21  22  23
> ###Matrix Manipulation in R###
>
> mat1<-matrix(c(11,21,12,22,13,23), n
> mat1
      [,1] [,2] [,3]
[1,]  11  12  13
[2,]  21  22  23
>
> mat2<-matrix(c(11,21,12,22,13,23), n
> mat2
      [,1] [,2]
[1,]  11  22
[2,]  21  13
[3,]  12  23
>
> dim (mat1)
[1] 2 3
> ncol (mat1)
[1] 3
> nrow (mat1)
[1] 2
. ■
```

Matrices in R

- We can also input data from a sequence of numbers

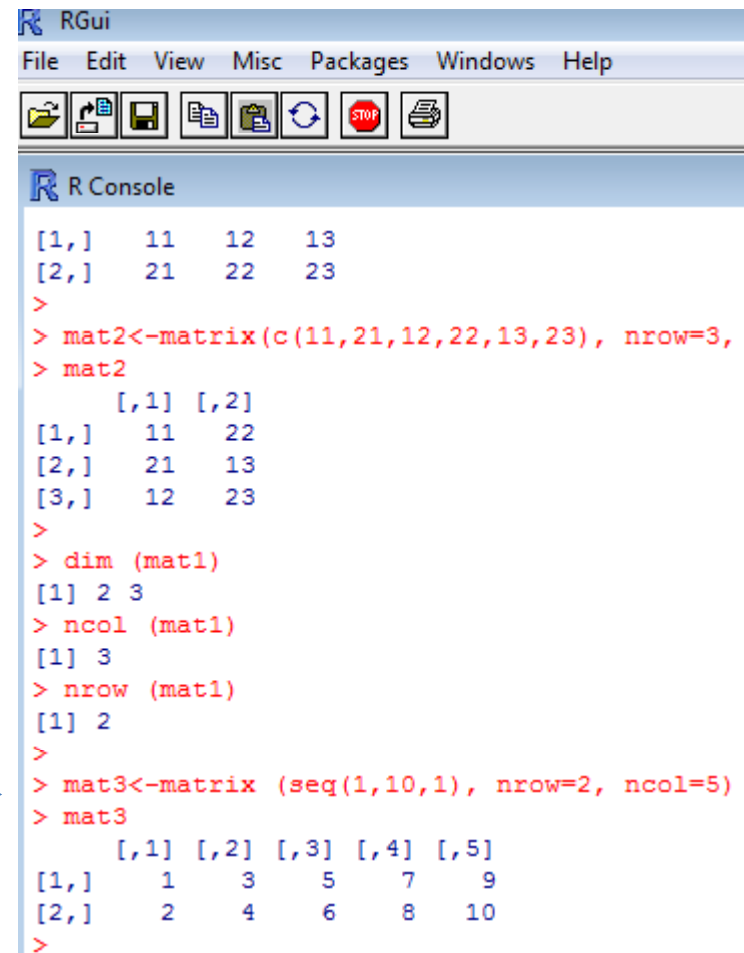
- `seq(from, to, by)`

- Where

- from is the beginning value of the sequence
- to is the ending value of the sequence
- by is the difference between consecutive values

- `mat3<-matrix(seq(1,10,1), nrow=2, ncol=5)` →

- `mat3`



```
RGui
File Edit View Misc Packages Windows Help

R Console
[1,] 11 12 13
[2,] 21 22 23
>
> mat2<-matrix(c(11,21,12,22,13,23), nrow=3,
> mat2
      [,1] [,2]
[1,] 11 22
[2,] 21 13
[3,] 12 23
>
> dim(mat1)
[1] 2 3
> ncol(mat1)
[1] 3
> nrow(mat1)
[1] 2
>
> mat3<-matrix(seq(1,10,1), nrow=2, ncol=5)
> mat3
      [,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
>
```

Matrices in R

- Addition
 - To add matrices we just use the summation sign
 - `mat1+mat4`
 - To subtract two matrices use the negative sign
 - `mat1-mat4`

```
File Edit Packages Windows Help
[Icons]
R Console
>
> mat4<-matrix (seq(0,5,1), nrow=
> mat4
      [,1] [,2] [,3]
[1,]    0    2    4
[2,]    1    3    5
>
> mat1
      [,1] [,2] [,3]
[1,]   11   12   13
[2,]   21   22   23
> mat4
      [,1] [,2] [,3]
[1,]    0    2    4
[2,]    1    3    5
> mat1+mat4
      [,1] [,2] [,3]
[1,]   11   14   17
[2,]   22   25   28
>
> mat1-mat4
      [,1] [,2] [,3]
[1,]   11   10    9
[2,]   20   19   18
> █
```


Matrices in R

- Multiplication of matrices is performed by `%*%`
 - `mat1*mat2`
 - 2x2 matrix results
- Kronecker product is performed by `%x%`
 - `mat1*mat3`
 - 4x15 matrix results

```
>
> mat1
      [,1] [,2] [,3]
[1,]  11  12  13
[2,]  21  22  23
> mat2
      [,1] [,2]
[1,]  11  22
[2,]  21  13
[3,]  12  23
> mat1%*%mat2
      [,1] [,2]
[1,]  529  697
[2,]  969 1277
>
> mat1
      [,1] [,2] [,3]
[1,]  11  12  13
[2,]  21  22  23
> mat3
      [,1] [,2] [,3] [,4] [,5]
[1,]   1   3   5   7   9
[2,]   2   4   6   8  10
> mat1%x%mat3
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15]
[1,]  11  33  55  77  99  12  36  60  84  107 131 155 179 203 227
[2,]  22  44  66  88 110  24  48  72  96 120 144 168 192 216 240
[3,]  21  63 105 147 189  22  66 110 154 196 238 280 322 364 406
[4,]  42  84 126 168 210  44  88 132 176 220 264 308 352 396 440
      [,15]
[1,]  117
[2,]  130
[3,]  207
```

Matrices in R

- For regression and beyond a few more commands are especially helpful
- Extracting the determinant of a square matrix
 - `det(mat6)`
- Inverting matrices
 - `solve(mat6)`

```
R Console
[1,] 117
[2,] 130
[3,] 207
[4,] 230
>
>
> mat5<-matrix (seq(0,8,1), nrow=3, ncol=3)
> mat5
      [,1] [,2] [,3]
[1,]  0    3    6
[2,]  1    4    7
[3,]  2    5    8
>
> det(mat5)
[1] 0
> solve(mat5) #will not solve bc mat5 is singular -
Error in solve.default(mat5) : Lapack routine dgesv
>
> mat6<-matrix (c(11,21,12,22), nrow=2, ncol=2)
> mat6
      [,1] [,2]
[1,]  11   12
[2,]  21   22
> det(mat6)
[1] -10
> solve(mat6)
      [,1] [,2]
[1,] -2.2  1.2
[2,]  2.1 -1.1
>
```

Matrices in R

- Transposing a matrix
 - `t(matrix)`
- Create a matrix with a particular diagonal
 - `diag(value, nrow=x, ncol=y)`
- Extracting eigenvalues and eigenvectors
 - `eigen(matrix)`

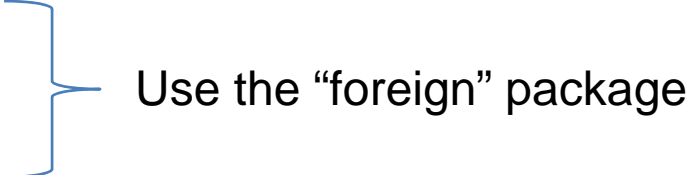
```
R Console
> mat6
      [,1] [,2]
[1,]  11  12
[2,]  21  22
> det(mat6)
[1] -10
> solve(mat6)
      [,1] [,2]
[1,] -2.2  1.2
[2,]  2.1 -1.1
>
> t(mat1)
      [,1] [,2]
[1,]  11  21
[2,]  12  22
[3,]  13  23
> diag(1, nrow=5, ncol=5)
      [,1] [,2] [,3] [,4] [,5]
[1,]  1  0  0  0  0
[2,]  0  1  0  0  0
[3,]  0  0  1  0  0
[4,]  0  0  0  1  0
[5,]  0  0  0  0  1
> eigen(mat6)
$values
[1] 33.3002976 -0.3002976

$vectors
      [,1] [,2]
[1,] -0.4738594 -0.7280128
[2,] -0.8806005  0.6855635
> █
```

Matrices in R

- We now have the basic understanding of the R language to “hand-roll” an ordinary least squares (OLS) regression and calculate the std. errors
 - $y_i = \alpha + \beta x_i + \varepsilon_i$
 - In matrix form: $(X'X)^{-1} X'Y$
- We can
 - Bind values into a vector
 - Invert matrices
 - Transpose matrices
- To do so with much larger datasets is where we move next...

Datasets in R

- We can create simple datasets by simply naming the rows and columns of an object
- However, we will often be looking at much larger datasets than those we just created
 - Unless of course you're a comparativist 😊
 - Typically we collect or store the data as other file types
- Fortunately R reads all kinds of datasets
 - ASCII or .txt files
 - SPSS or .sav files
 - STATA or .dta files

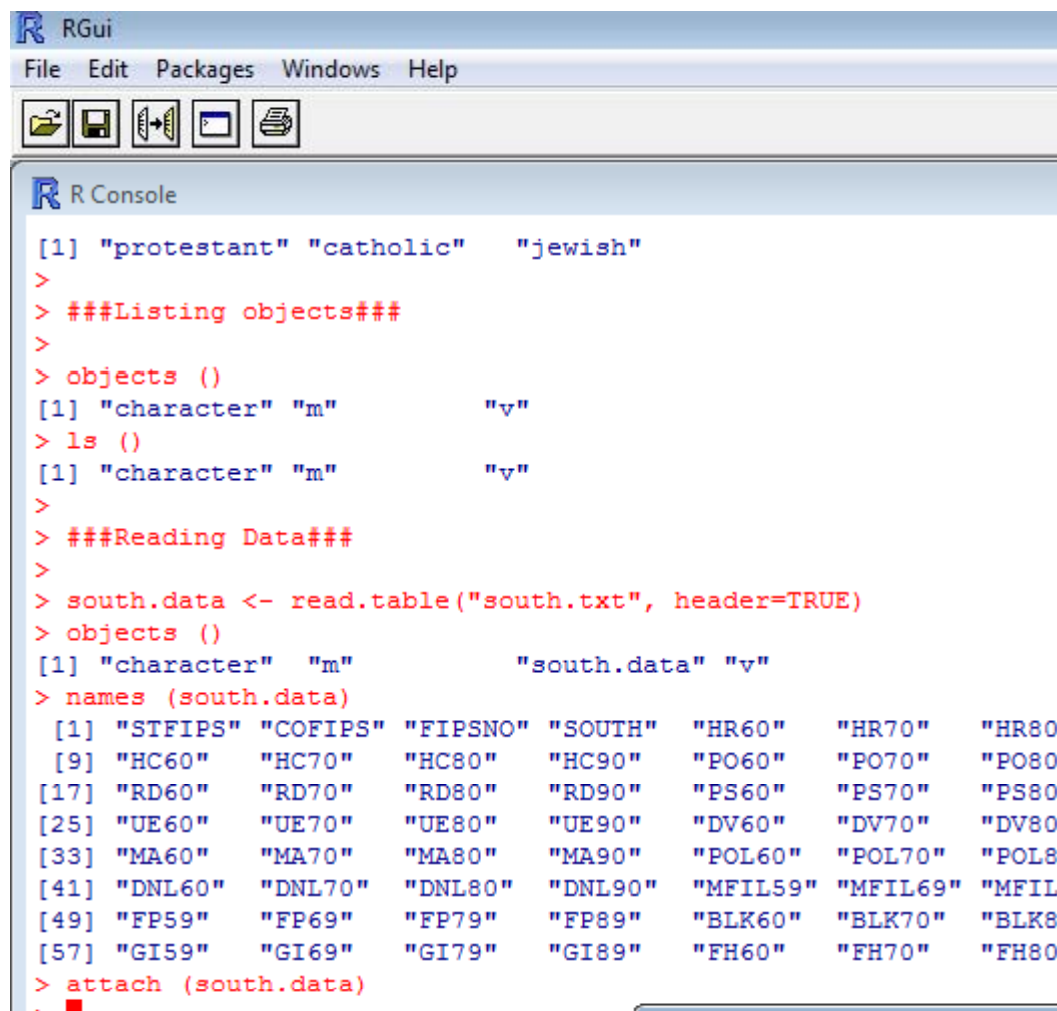
Use the “foreign” package

Datasets in R: ASCII

- ASCII files are common among political science data (.txt or .dat)
- Let's read into R the "south.txt" data using the `read.table` function
 - `south.data<-read.table("south.txt", header=TRUE)`
 - Note: we are assuming this data is already in our working directory

Datasets in R: ASCII

- Let's check and see what new objects we have
 - `objects()`
- What are the names of our variables
 - `names(south.dta)`
- How can we use the variables as vectors in our subsequent analyses?
 - Attach the data
 - `attach(south.dta)`



```
RGui
File Edit Packages Windows Help

R Console

[1] "protestant" "catholic" "jewish"
>
> ###Listing objects###
>
> objects ()
[1] "character" "m" "v"
> ls ()
[1] "character" "m" "v"
>
> ###Reading Data###
>
> south.data <- read.table("south.txt", header=TRUE)
> objects ()
[1] "character" "m" "south.data" "v"
> names (south.data)
 [1] "STFIPS" "COFIPS" "FIPSNO" "SOUTH" "HR60" "HR70" "HR80"
 [9] "HC60" "HC70" "HC80" "HC90" "PO60" "PO70" "PO80"
[17] "RD60" "RD70" "RD80" "RD90" "PS60" "PS70" "PS80"
[25] "UE60" "UE70" "UE80" "UE90" "DV60" "DV70" "DV80"
[33] "MA60" "MA70" "MA80" "MA90" "POL60" "POL70" "POL8"
[41] "DNL60" "DNL70" "DNL80" "DNL90" "MFIL59" "MFIL69" "MFIL"
[49] "FP59" "FP69" "FP79" "FP89" "BLK60" "BLK70" "BLK8"
[57] "GI59" "GI69" "GI79" "GI89" "FH60" "FH70" "FH80"
> attach (south.data)
> █
```

Datasets in R: Foreign

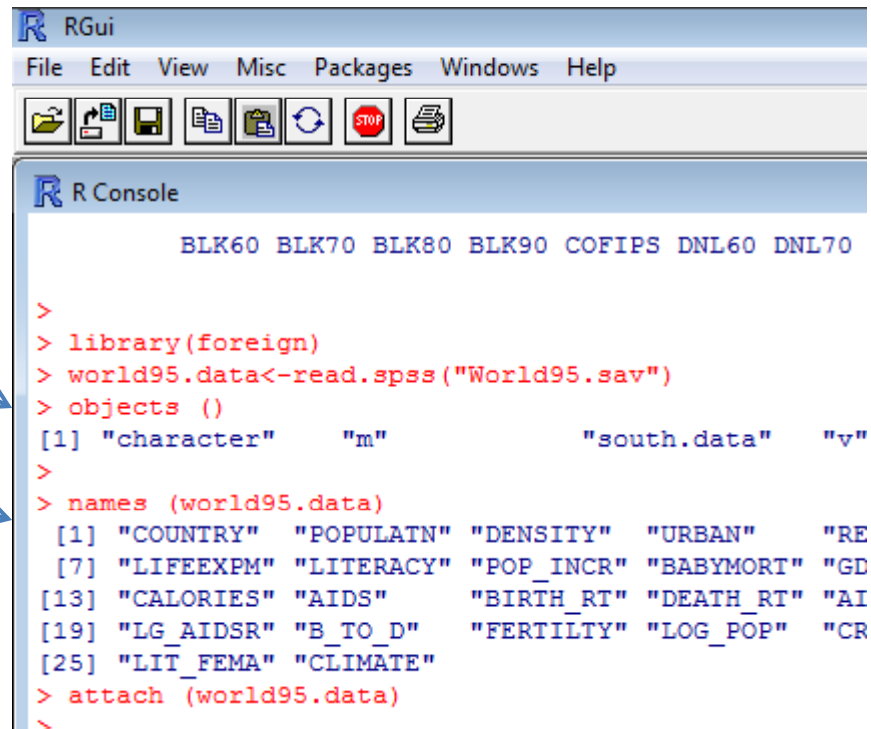
- To read in other types of data, we must load the `foreign` package
 - `library(foreign)`
- Quick digression on packages
 - R has a host of packages – over a thousand of them, actually
 - Open source
 - Dangers to using a package without knowing what it really does
 - Use at your own risk
 - STATA and SPSS better about testing the merits of canned commands
 - Packages have “canned” routines for some of the most frequently used statistical commands
 - Recall the `lm` command which comes from the linear models package
 - Getting packages
 - Go to Packages
 - Select an appropriate mirror (go Blue?)
 - Download the package of your choosing

Datasets in R: SPSS

- Let's read into R the "World95.sav" data using the `foreign` package
 - `world95.data <- read.spss("World95.sav")`
 - Note: we are again assuming this data is already in our working directory
- Cute feature
 - If the data is not in the working directory, we can browse for the data file with the `file.choose` option
 - `world95.data.2 <- read.spss(file.choose())`

Datasets in R: SPSS

- Let's check and see what new objects we have
 - `objects()`
- What are the names of our variables
 - `names(world95.dta)`
- How can we use the variables as vectors in our subsequent analyses?
 - Attach the data
 - `attach(world95.dta)`



```
RGui
File Edit View Misc Packages Windows Help

R Console
BLK60 BLK70 BLK80 BLK90 COFIPS DNL60 DNL70

>
> library(foreign)
> world95.data<-read.spss("World95.sav")
> objects ()
[1] "character"      "m"              "south.data"     "v"
>
> names (world95.data)
[1] "COUNTRY"  "POPULATN"  "DENSITY"  "URBAN"  "RE
[7] "LIFEEXPM" "LITERACY"  "POP_INCR" "BABYMORT" "GD
[13] "CALORIES" "AIDS"      "BIRTH_RT" "DEATH_RT" "AI
[19] "LG_AIDSR" "B_TO_D"    "FERTILTY" "LOG_POP"  "CR
[25] "LIT_FEMA" "CLIMATE"
> attach (world95.data)
>
```

Datasets in R: STATA

- How about a .dta file from STATA?
 - I prefer to do all my data recoding in STATA and then use R for analyses and graphs
- Let's read into R the "senate.dta" data using the `foreign` package
 - This package is already loaded so we don't need to do so again
 - `senate02.data <- read.dta("Senate2002.dta")`
 - As always

Datasets in R: STATA

- Again...
- Let's check and see what new objects we have
 - `objects()`
- What are the names of our variables
 - `names(senate02.dta)`
- How can we use the variables as vectors in our subsequent analyses?
 - Attach the data
 - `attach(senate02.dta)`
- And now we are ready for data analysis!
- See the next set of slides...

```
[13] "CALORIES" "AIDS"      "BIRTH_RT" "DEATH_R
[19] "LG_AIDSR" "B_TO_D"    "FERTILTY" "LOG_POP
[25] "LIT_FEMA" "CLIMATE"
> attach(world95.data)
>
> senate02.dta<-read.dta("Senate2002.dta")
> objects()
[1] "character"      "m"           "senate02.
[5] "v"              "world95.data"
>
> names(senate02.dta)
[1] "repvshr" "income" "presvote" "pressup"
> attach(senate02.dta)
>
>
>
>
> █
```

Our Favorite R Resources

- Invaluable Resources online
 - The R manuals
<http://cran.r-project.org/manuals.html>
 - Fox's slides <http://socserv.mcmaster.ca/jfox/Courses/R-course/index.html>
 - Faraway's book
<http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf>
 - Anderson's ICPSR lectures using R
<http://socserv.mcmaster.ca/andersen/icpsr.html>
 - Arai's guide http://people.su.se/~ma/R_intro/
 - UCLA notes <http://www.ats.ucla.edu/stat/SPLUS/default.htm>
 - Keele's intro guide <http://www.polisci.ohio-state.edu/faculty/lkeele/RIntro.pdf>
- Great R books
 - Verzani's book
<http://www.amazon.com/Using-Introductory-Statistics-John-Verzani/dp/1584884509>
 - Maindonald and Braun's book
<http://www.amazon.com/Data-Analysis-Graphics-Using-R/dp/0521813360>