While We Are Waiting…

- If you want to work along with the presentation
- All the materials are on the department share drive
  - Go to: I → PRISM → Brownbags → Intro to R
  - I:\PRISM\Brownbags\Intro to R
- Copy the following files into your K personal drive
  - The datasets are labeled
    - "World95.sav"
    - "south.txt"
    - "Senate2002.dta"
  - The command script is labeled "Rscript_V07_FINAL.txt"
  - The presentation is labeled "IntrotoR_V07FINAL.pdf"

PRISM Brownbag:
An Introduction to

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
VIII. Data Analysis
    i. Descriptives
    ii. Command functions and hand-rolling
    iii. Diagnostics
    iv. Graphing
    v. Functions and loops
IX. Moving forward

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

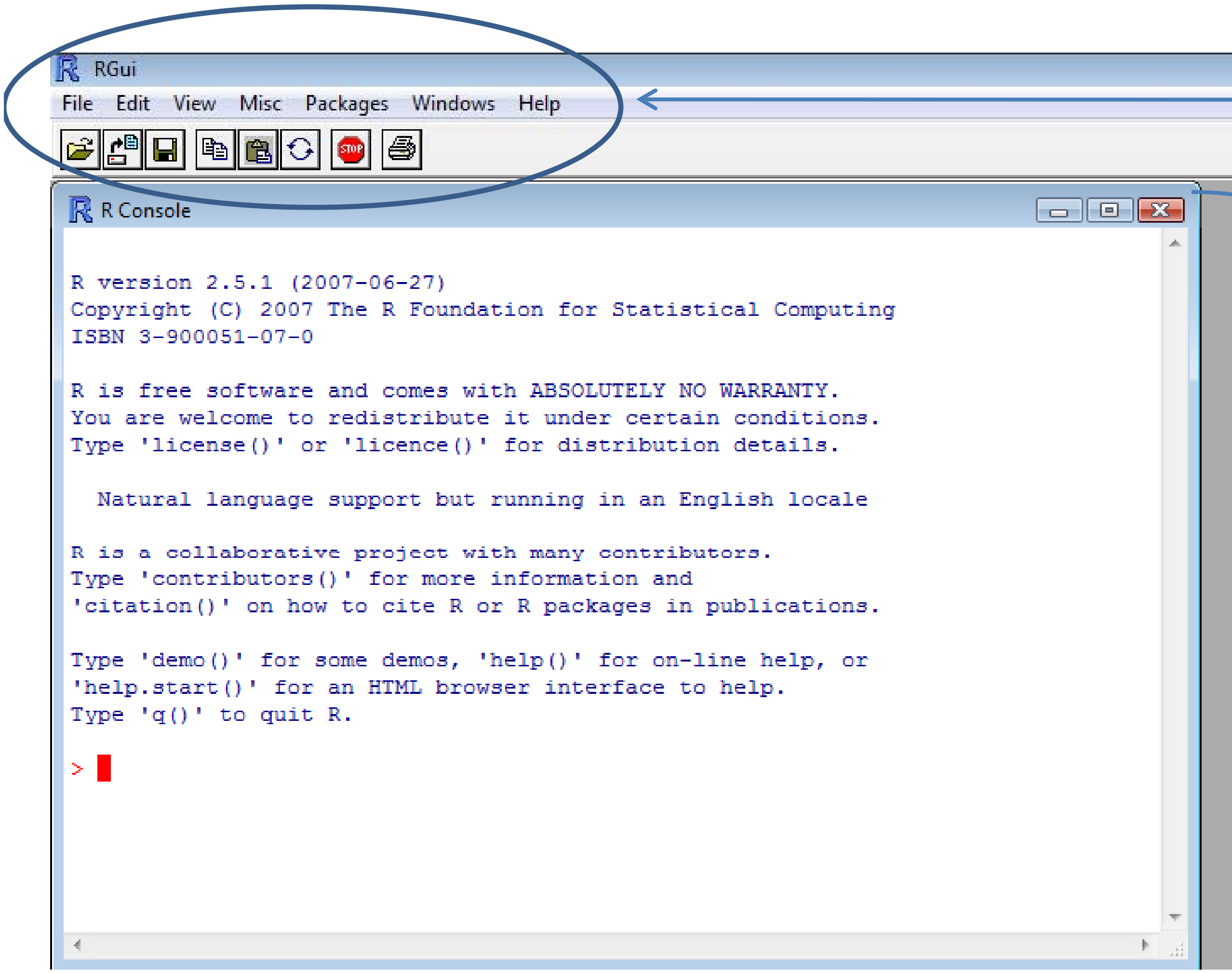
Why NOT use R?

- 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



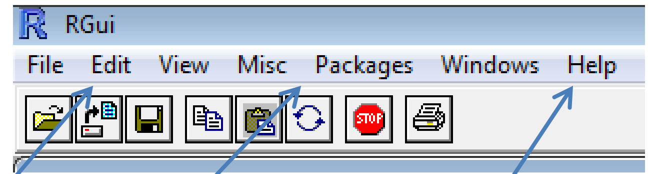
R's GUI

- 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

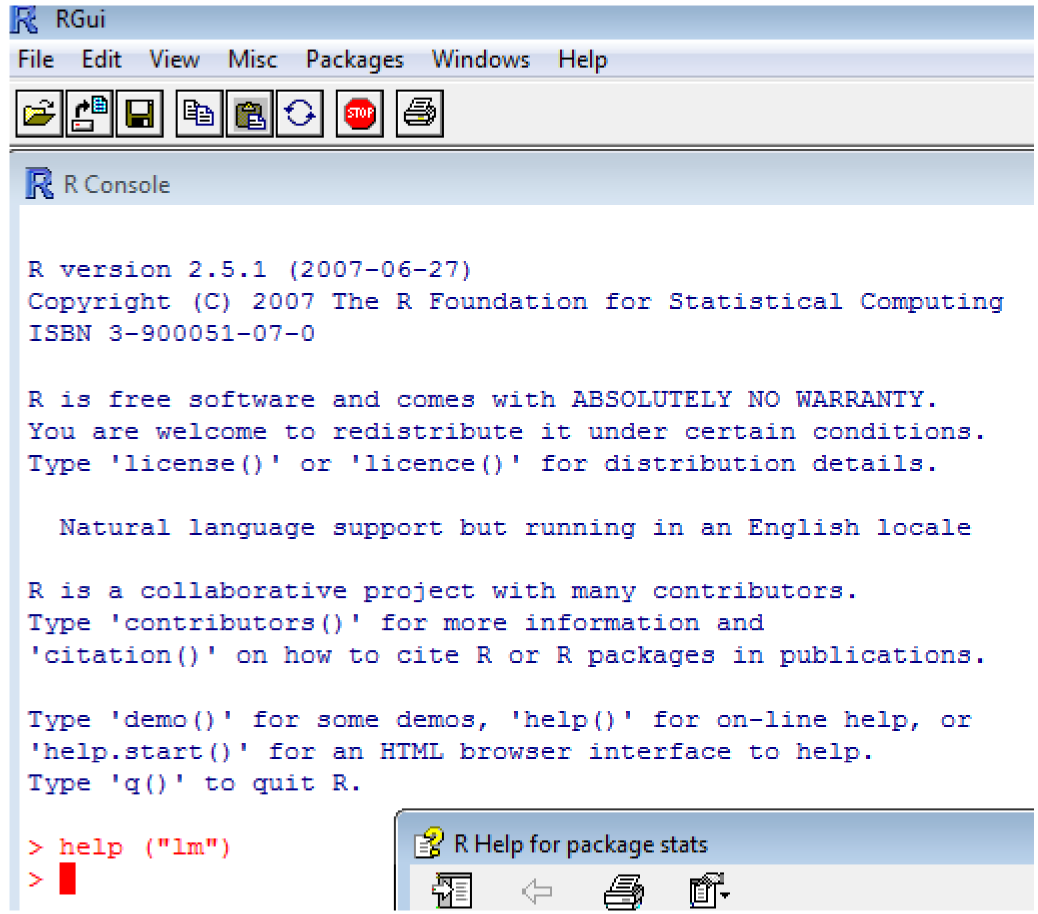
R’s GUI



- 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

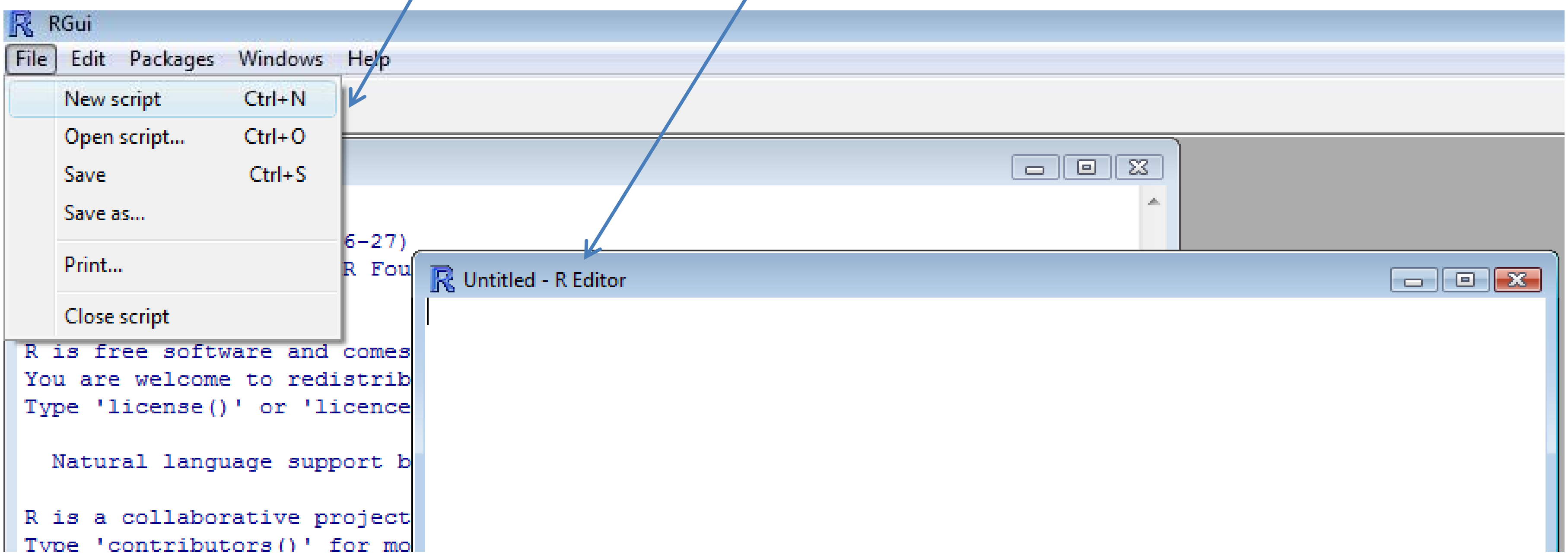
A Note on GUI

- 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



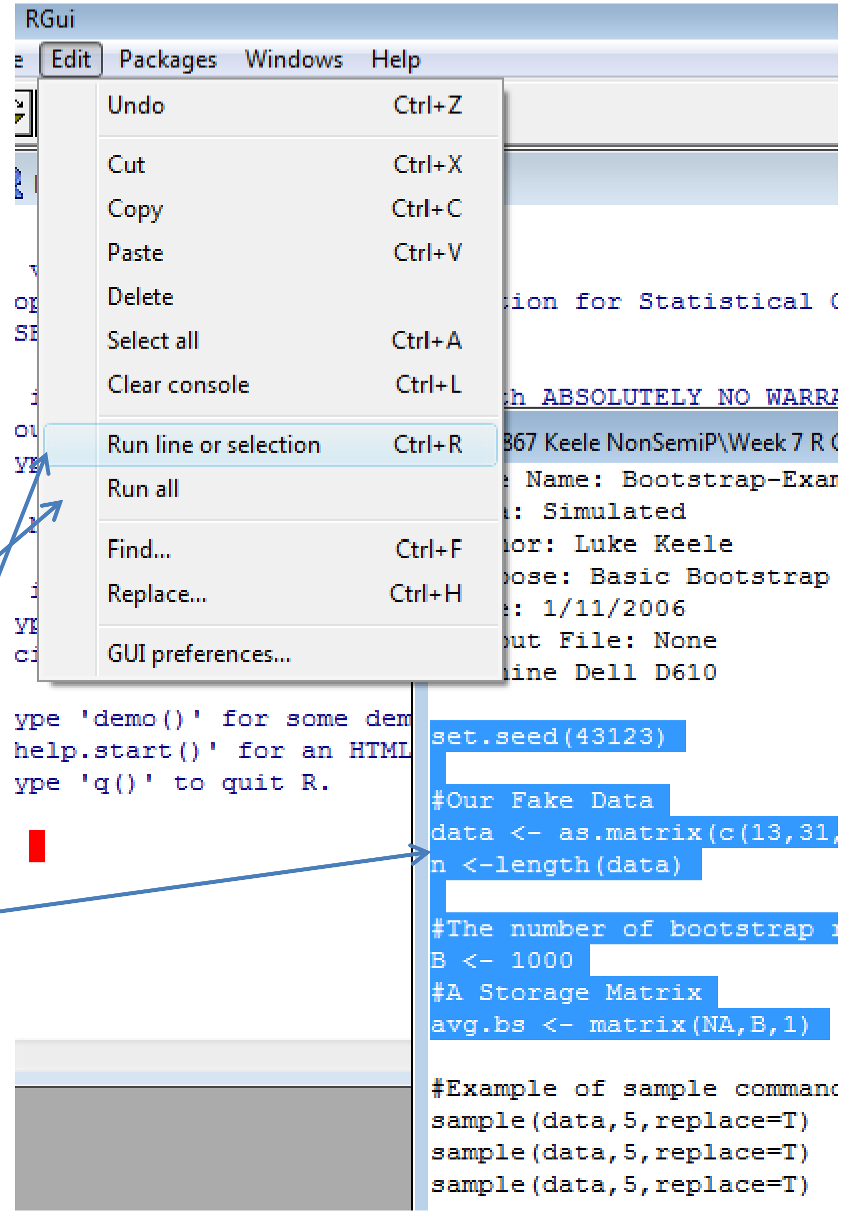
R Script

- 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

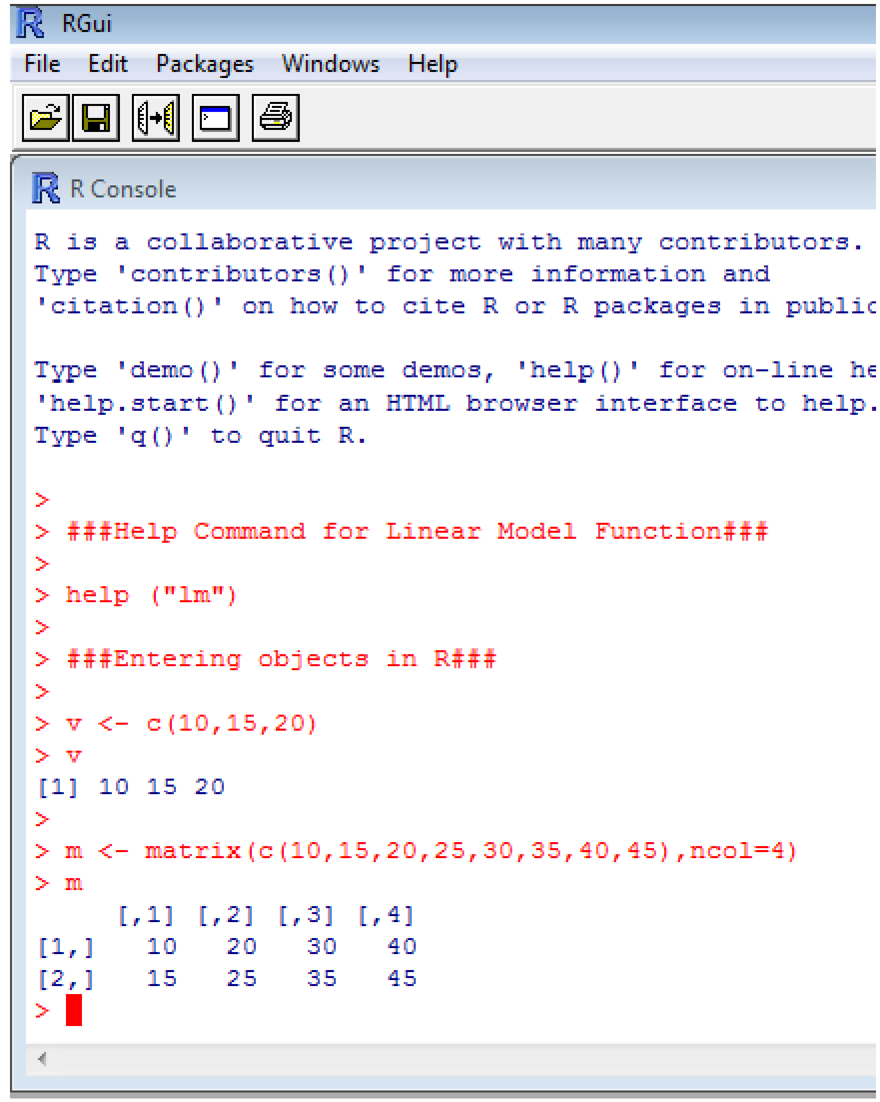
- R is based on objects: vectors & matrices
- When entering commands
  - Expressions and commands are case-sensitive
  - Anything following the pound symbol (#) is treated as a comment and ignored by R
  - An object name must start with an alphabetical character but may contain numbers and periods thereafter
  - 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(command options)`
  - `object.name = command(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

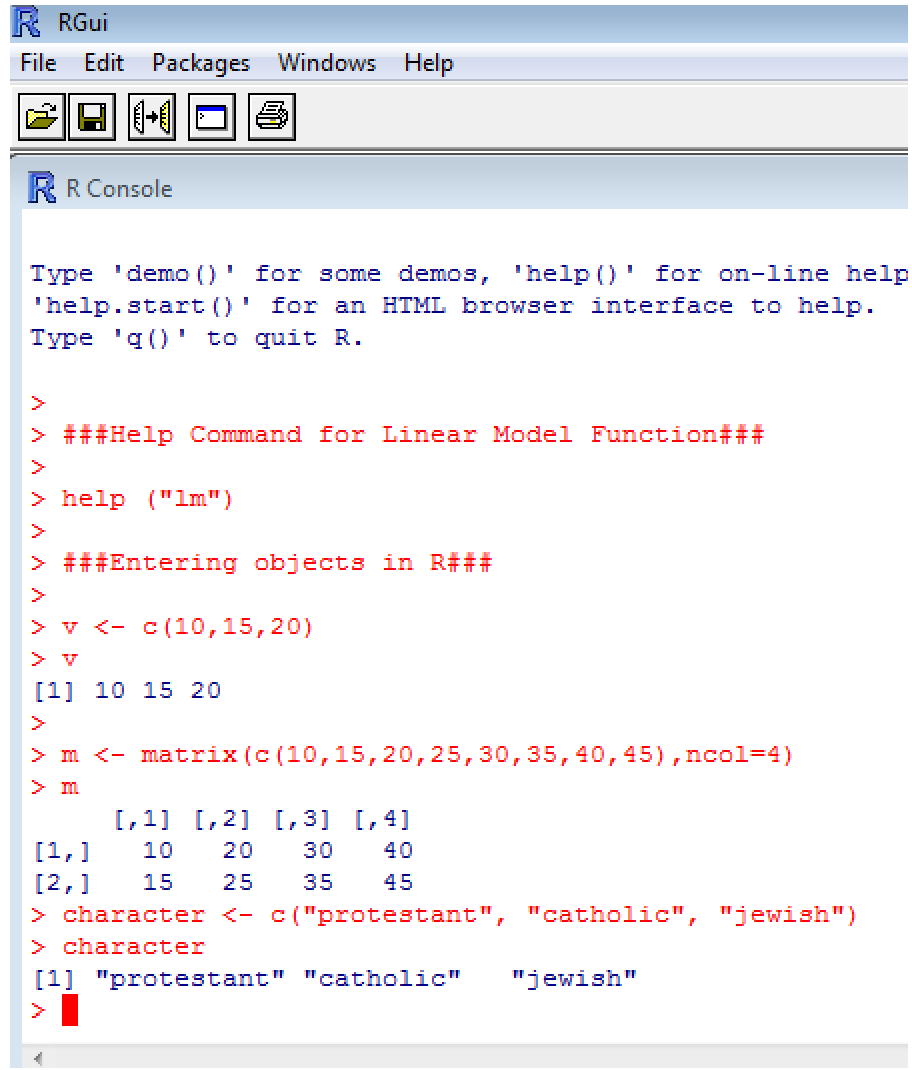
Objects in R

- 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`



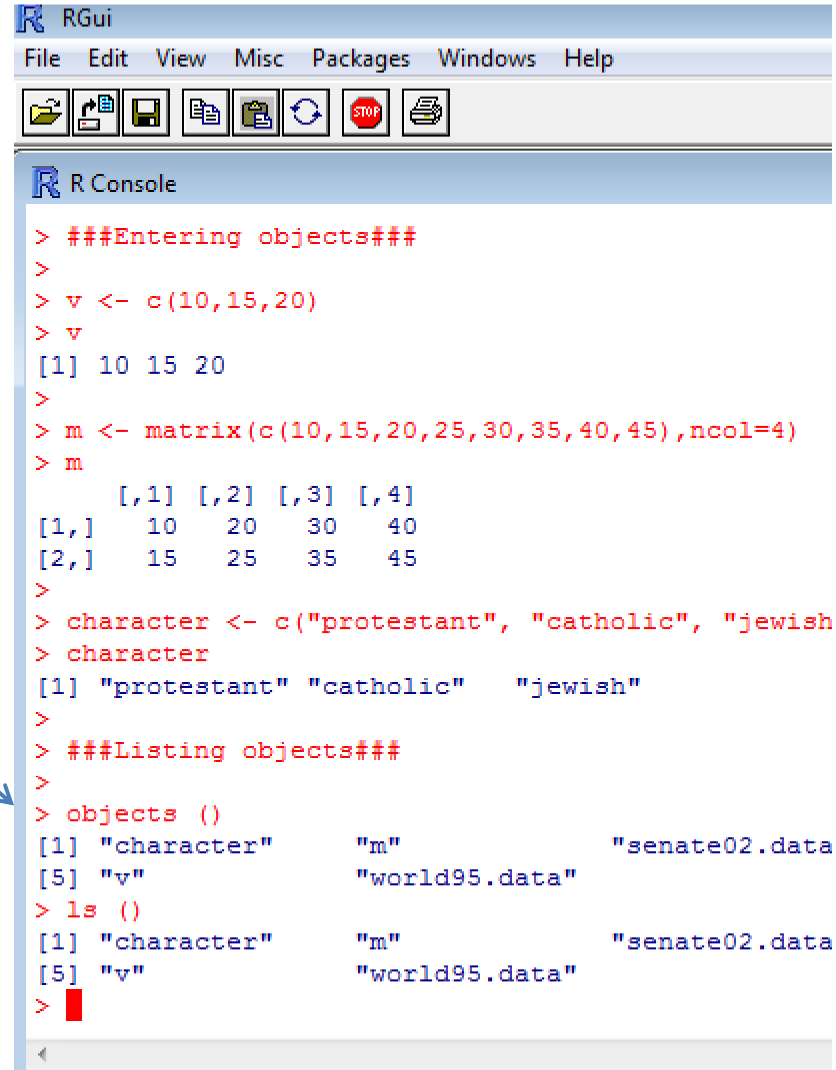
Objects in R

- Beyond numerical vectors, we can also do character or logic vectors
  - A character vector
    - `character <- c(“protestant”, “catholic”, ”jewish”)`
    - `character`



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)`

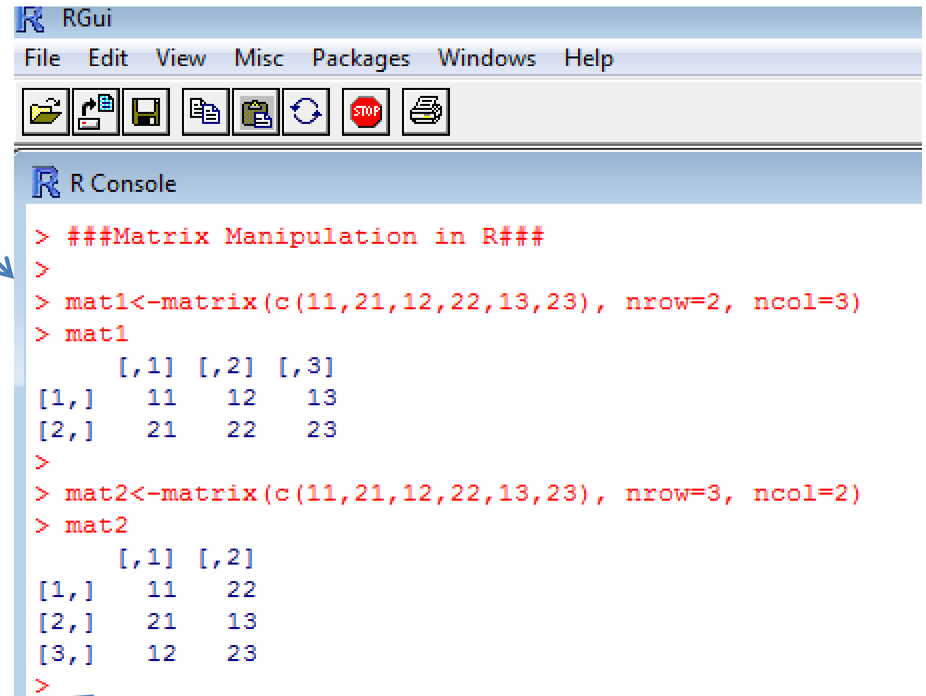


Matrices in R

- 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

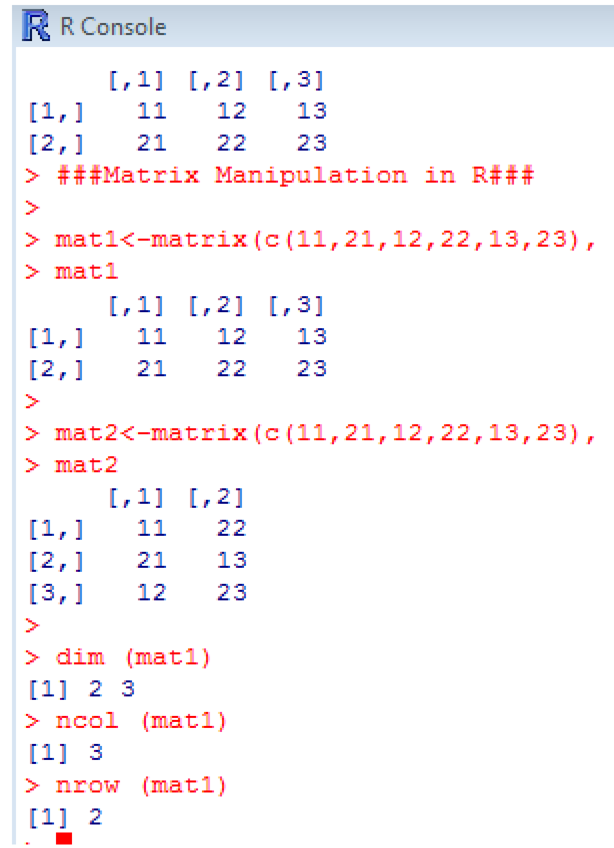
Matrices in R

- `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?



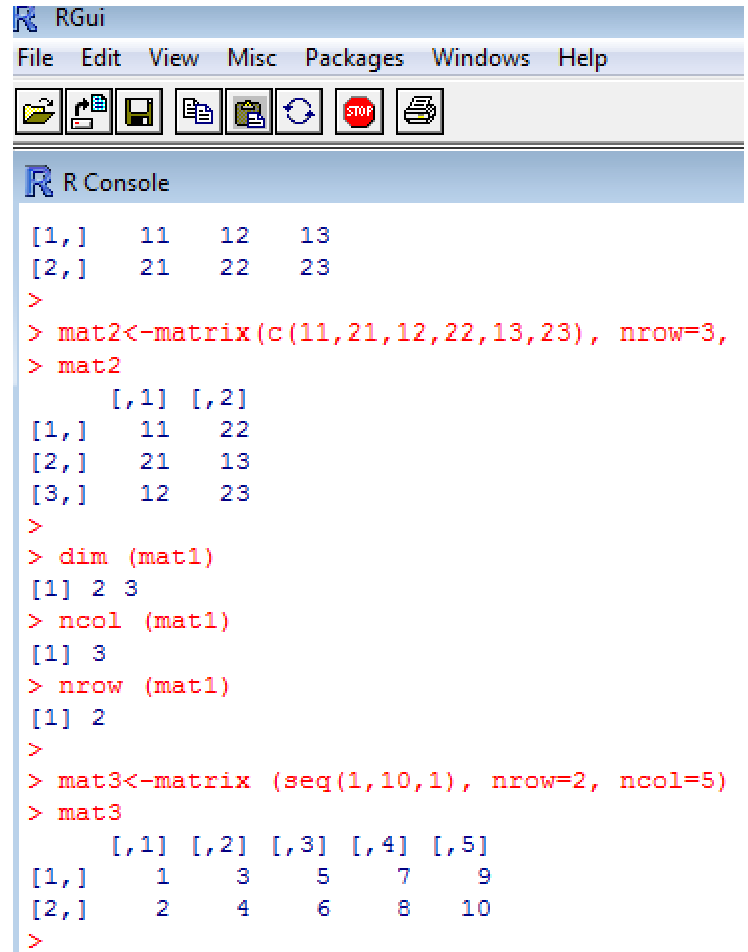
Matrices in R

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



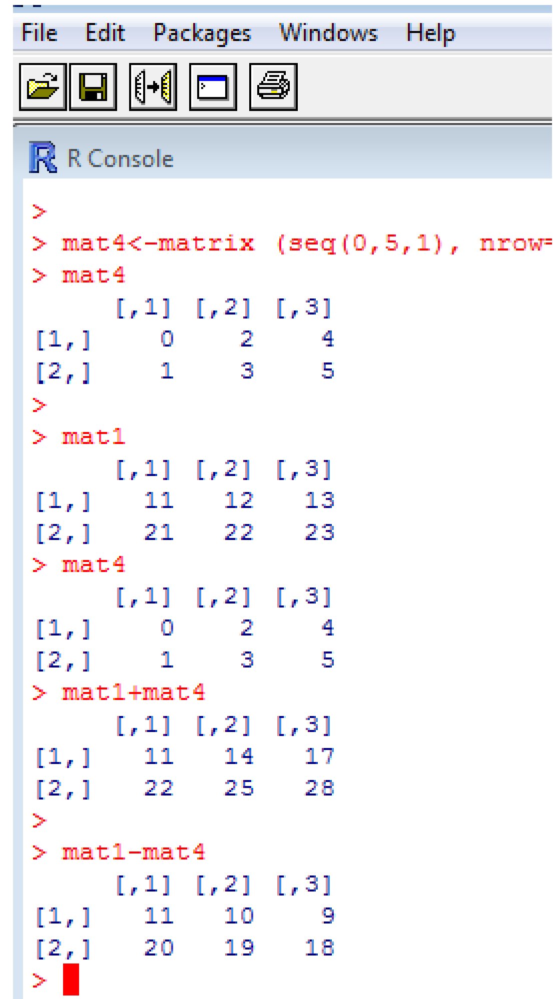
Matrices in R

- 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`



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`



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] [
[1,]   11   33   55   77   99   12   36   60   84
[2,]   22   44   66   88  110   24   48   72   96
[3,]   21   63  105  147  189   22   66  110  154
[4,]   42   84  126  168  210   44   88  132  176
     [,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 dges
>
> 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

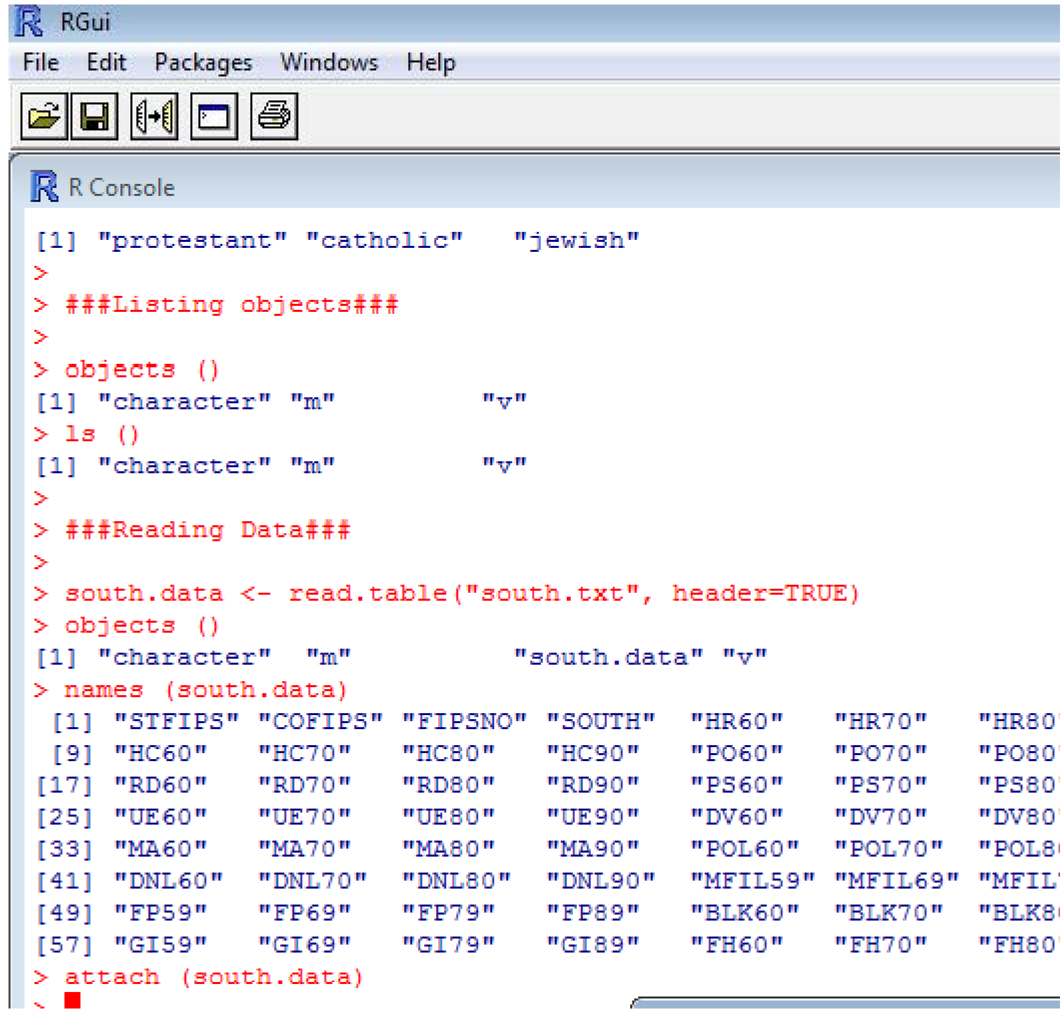
Use the "foreign" package

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)`



Datasets in R: Foreign

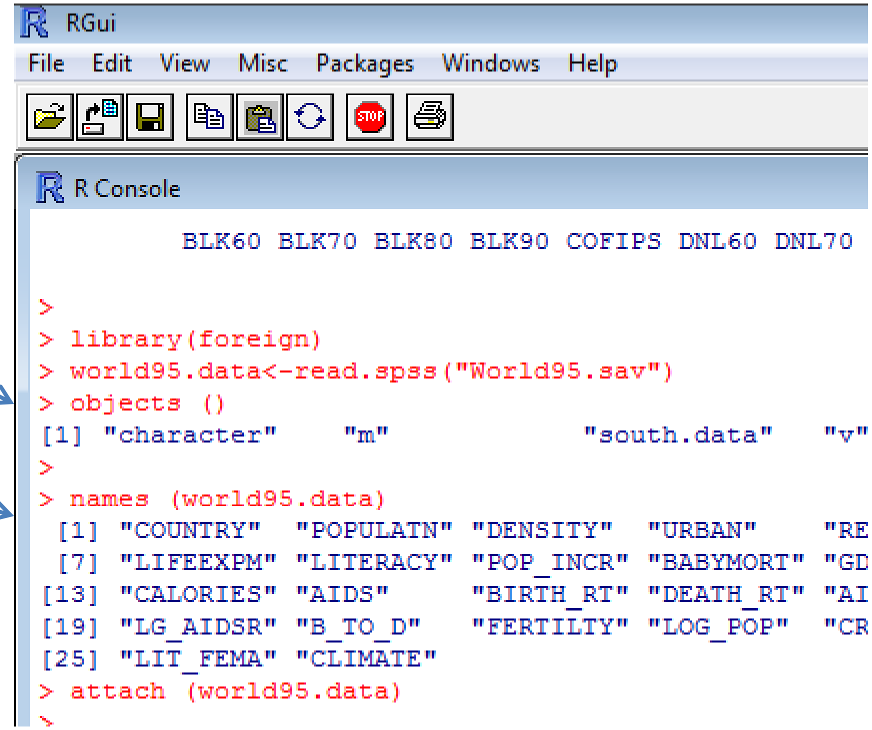
- To read in other types of data, we must load the `foreign` package
  - `library(foreign)`
- Quick digression on packages
  - R has a host of packages – over a thousand of them, actually
    - Open source
      - Dangers to using a package without knowing what it really does
      - Use at your own 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)`

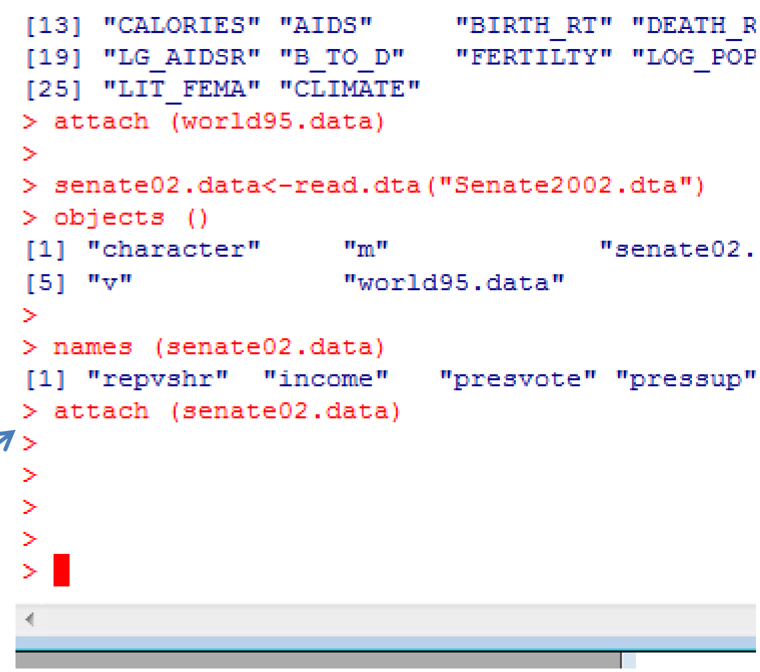


Datasets in R: STATA

- 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!



Data Analysis: Descriptive Stats

- R has several built-in commands for describing data
- The `list()` command can output all elements of an object

```
R Console
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> #Load Libraries
> library(foreign)
>
>
> #Read in the Data
>
> data <- read.dta("Senate2002.dta")
> attach(data)
> #Descriptive Statistics - Examples
>
> list(data)
[[1]]
    repvshr income presvote pressup
1  59.52665  34135     56.0      88
2  88.14516  51571     59.0      95
3  52.55351  47203     51.0      96
4  41.20611  47381     42.0      77
5  53.46671  42433     55.0      90
6  66.68880  37572     67.0      95
7  38.65622  46590     43.0      67
8  44.69222  39469     48.0      69
9  64.67520  33672     57.0      96
10 48.29952  32566     53.0      84
11 58.43565  37240     44.0      88
12 38.46476  44667     46.0      66
13 33.58933  33024     58.0      88
14 84.97204  39250     62.0      98
15 65.04277  34133     47.8      96
16 61.21235  33400     60.0      96
17 58.66380  40916     46.5      91
18 21.57351  42090     32.0      66
19 55.13115  37082     57.0      82
20 49.92046  35282     60.0      68
21 36.88865  29696     52.0      71
22 56.77559  37892     68.0      93

>
```

Data Analysis: Descriptive Stats

- The `summary()` command can be used to describe all variables contained within a dataframe
- The `summary()` command can also be used with individual variables

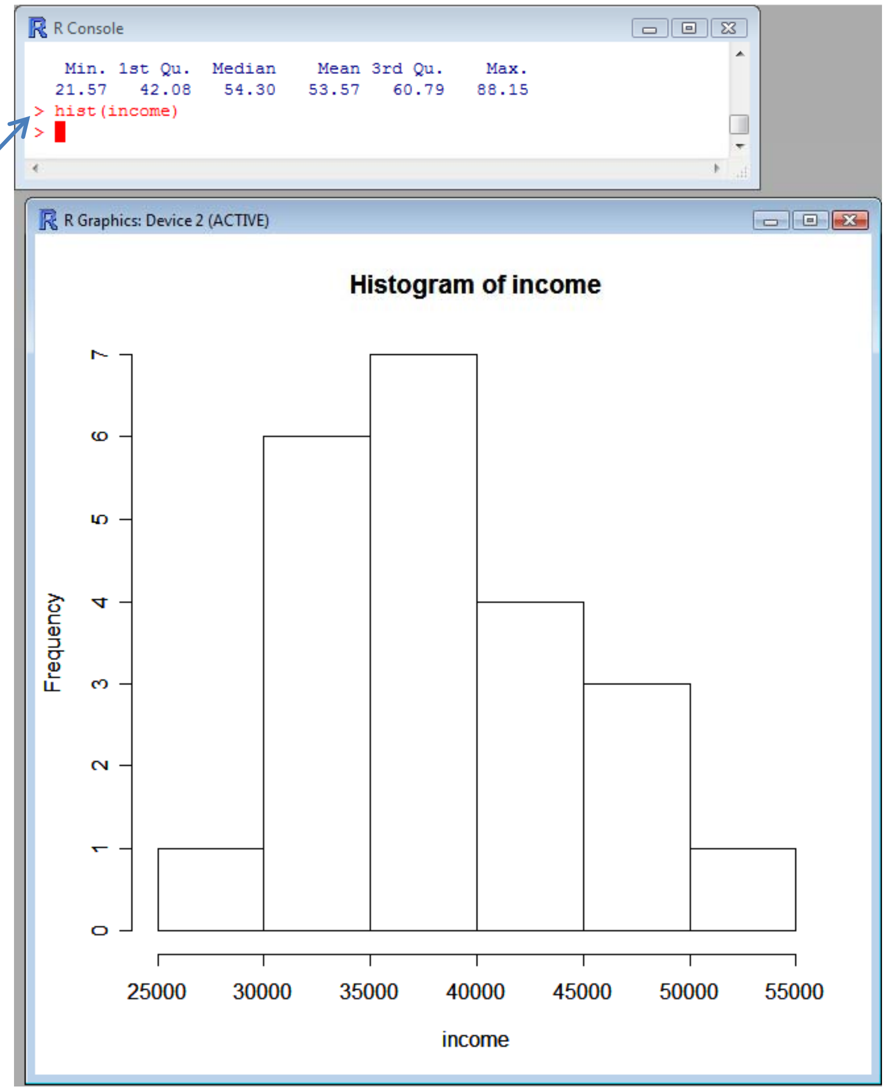
R Console

```
> #Descriptive Statistics - Examples
>
> list(data)
[[1]]
     repvshr income presvote pressup
1  59.52665  34135     56.0      88
2  88.14516  51571     59.0      95
3  52.55351  47203     51.0      96
4  41.20611  47381     42.0      77
5  53.46671  42433     55.0      90
6  66.68880  37572     67.0      95
7  38.65622  46590     43.0      67
8  44.69222  39469     48.0      69
9  64.67520  33672     57.0      96
10 48.29952  32566     53.0      84
11 58.43565  37240     44.0      88
12 38.46476  44667     46.0      66
13 33.58933  33024     58.0      88
14 84.97204  39250     62.0      98
15 65.04277  34133     47.8      96
16 61.21235  33400     60.0      96
17 58.66380  40916     46.5      91
18 21.57351  42090     32.0      66
19 55.13115  37082     57.0      82
20 49.92046  35282     60.0      68
21 36.88865  29696     52.0      71
22 56.77559  37892     68.0      93

> summary(data)
    repvshr          income          presvote         pressup
 Min.   :21.57   Min.   :29696   Min.   :32.00   Min.   :66.00
 1st Qu.:42.08   1st Qu.:34134   1st Qu.:46.82   1st Qu.:72.50
 Median :54.30   Median :37732   Median :54.00   Median :88.00
 Mean   :53.57   Mean   :38967   Mean   :52.92   Mean   :84.55
 3rd Qu.:60.79   3rd Qu.:42347   3rd Qu.:58.75   3rd Qu.:95.00
 Max.   :88.15   Max.   :51571   Max.   :68.00   Max.   :98.00
> summary(repvshr)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  21.57   42.08   54.30   53.57   60.79   88.15
>
```

Data Analysis: Descriptive Stats

- Simple plots can also provide familiarity with the data

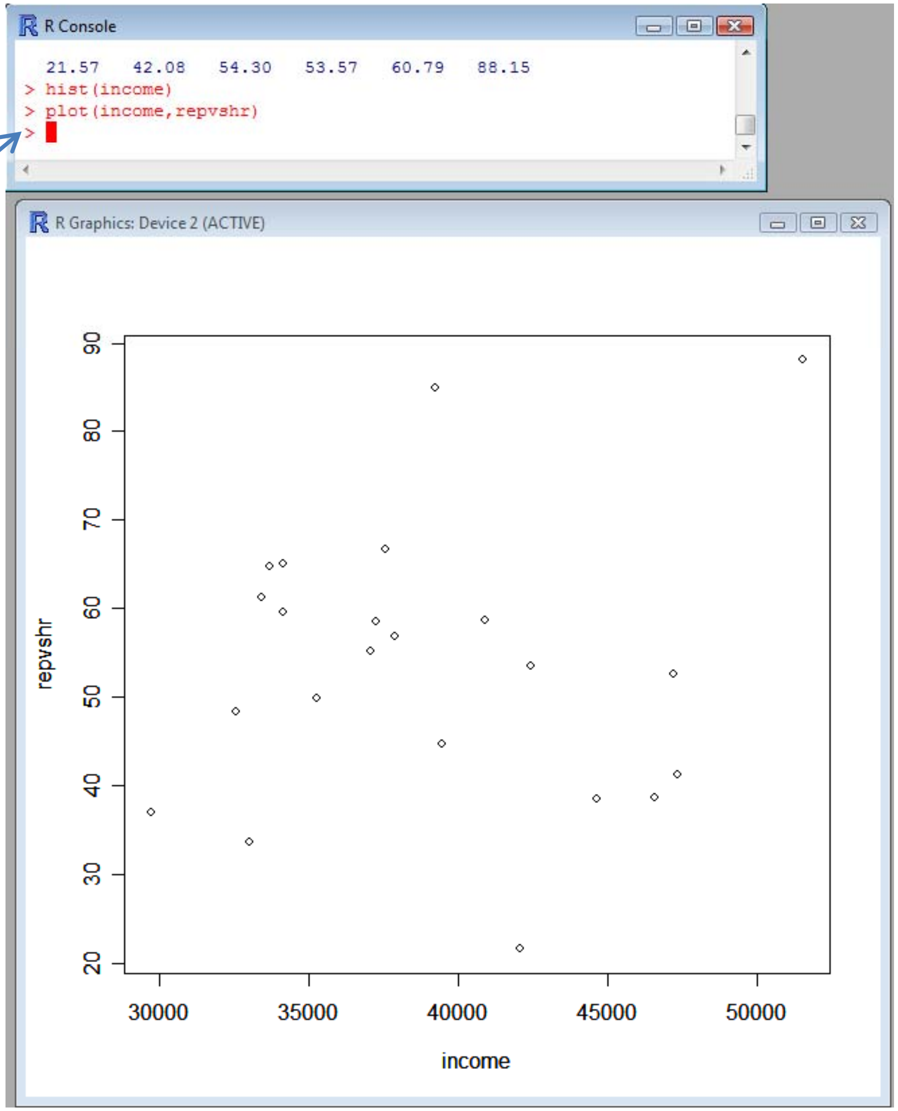
- The `hist()` command produces a histogram for any given data values



Data Analysis: Descriptive Stats

- Simple plots can also provide familiarity with the data

- The `plot()` command can produce both univariate and bivariate plots for any given objects



Data Analysis: Descriptive Stats

Other Useful Commands

- `sum`
- `mean`
- `var`
- `sd`
- `range`
- `min`
- `max`
- `median`
- `cor`
- `summary`

Data Analysis: Regression

- As mentioned above, one of the big perks of using R is flexibility.

- R comes with its own canned linear regression command: `lm(y ~ x)`

- However, we're going to use R to make our own OLS estimator. Then we will compare with the canned procedure, as well as Stata.

Data Analysis: Regression

- First, let's take a look at our code for the hand-rolled OLS estimator
- The Holy Grail: $(X'X)^{-1}X'Y$
- We need a single matrix of independent variables
- The `cbind()` command takes the individual variable vectors and combines them into one x-variable matrix
- A "1" is included as the first element to account for the constant.

Rintro - Notepad

File Edit Format View Help

```
#Hand-rolled OLS

    x<-as.matrix(cbind(int=1,income,presvote,pressup))
    y<-as.vector(repvshr)
    i<-diag(1,nrow=nrow(x),ncol=ncol(x))

    n<-length(y)
    p<-ncol(x)-1

    xy<-t(x)%*%y                  # X'Y
    xxi<-solve(t(x)%*%x)          #(X'X)^(-1)
    h<-x%*%xxi%*%t(x)             #hat matrix of x
    i<-diag(1,nrow=n,ncol=n)
    b<-as.vector(xxi%*%xy)        #estimated coefficients

    names(b)<-colnames(x)

    yhat<-as.vector(x%*%b)        #predicted values for y
    res<-y-yhat   # or (i-h)%*%y  #model residuals

    sst<-sum((y-mean(y))^2)       #Total sum of sqares
    sse<-t(res)%*%res             # or sum(res^2) which is also t(res)%*%res
    ssm<-sst-sse                  #sum of squares for model (regression)

    df.e<-(n-p-1)                 #degrees of freedom for error
    df.t<-(n-1)                   #total degrees of freedom
    df.m<-df.t-df.e               #degrees of freedom for model

    s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
    sigma2<-as.vector(sse/(n-p))
    r2<-1-(sse/sst)
    r2.adj<-1-((sse/df.e)/(sst/df.t))
    aic<-n*log(sse/n)+2*(p+1)
    cp<-(sse/s2)-(n-2*(p+1))
    f<-(ssm/df.m)/(sse/df.e)
    pvalue<-1-pf(f,df.m,df.e)

    b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
    b.t.statistic<-b/b.standard.errors           #t statistic for st. errors
    b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05
```

Data Analysis: Regression

- With the **x and y matrices complete**, we can now manipulate them to produce coefficients.
- After performing the **divine multiplication**, we can observe the estimates by entering the object name (in this case “b”).

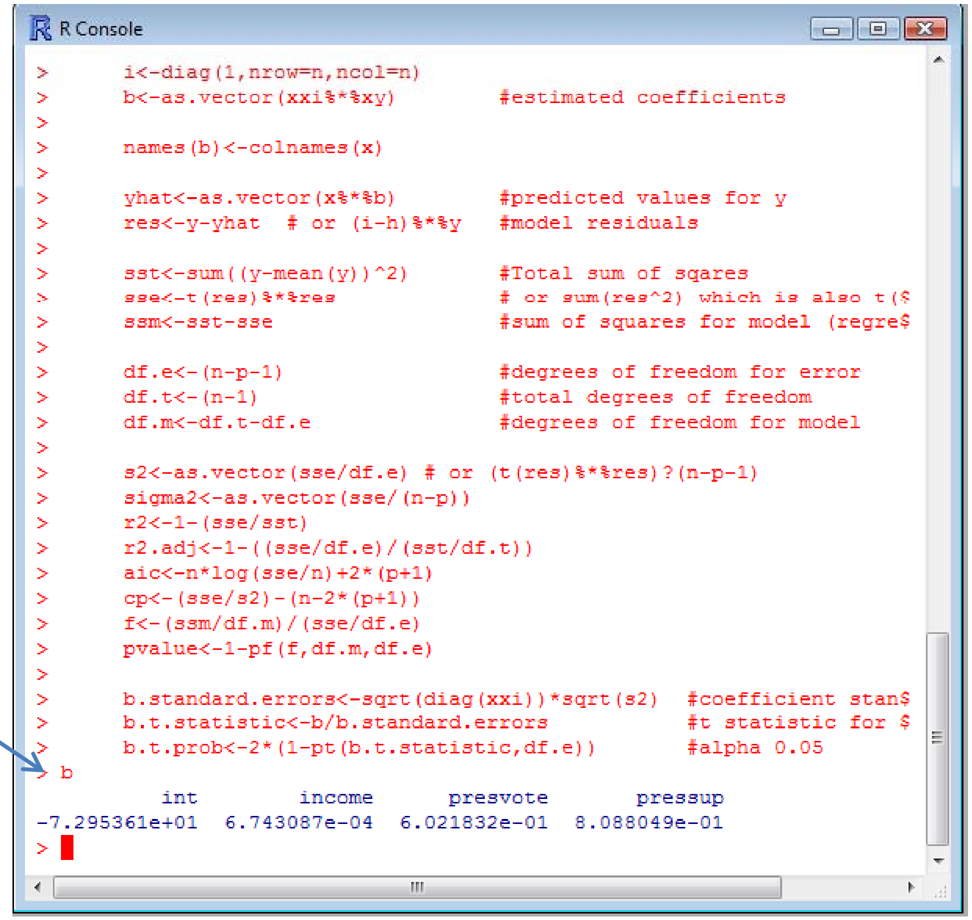
Rintro - Notepad

File Edit Format View Help

```
#Hand-rolled OLS


      x<-as.matrix(cbind(int=1,income,presvote,pressup))
      y<-as.vector(repvshr)
      i<-diag(1,nrow=nrow(x),ncol=ncol(x))

      n<-length(y)
      p<-ncol(x)-1


      xy<-t(x)%*%y                        # X'Y
      xxi<-solve(t(x)%*%x)                #(X'X)^(-1)
      h<-x%*%xxi%*%t(x)                   #hat matrix of x
      i<-diag(1,nrow=n,ncol=n)
      b<-as.vector(xxi%*%xy)              #estimated coefficients

      names(b)<-colnames(x)

      yhat<-as.vector(x%*%b)              #predicted values for y
      res<-y-yhat  # or (i-h)%*%y         #model residuals

      sst<-sum((y-mean(y))^2)             #Total sum of sqares
      sse<-t(res)%*%res                   # or sum(res^2) which is also t(res)%*%res
      ssm<-sst-sse                        #sum of squares for model (regression)

      df.e<-(n-p-1)                       #degrees of freedom for error
      df.t<-(n-1)                         #total degrees of freedom
      df.m<-df.t-df.e                     #degrees of freedom for model

      s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
      sigma2<-as.vector(sse/(n-p))
      r2<-1-(sse/sst)
      r2.adj<-1-((sse/df.e)/(sst/df.t))
      aic<-n*log(sse/n)+2*(p+1)
      cp<-(sse/s2)-(n-2*(p+1))
      f<-(ssm/df.m)/(sse/df.e)
      pvalue<-1-pf(f,df.m,df.e)

      b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
      b.t.statistic<-b/b.standard.errors           #t statistic for st. errors
      b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05
```

Data Analysis: Regression

- With the x and y matrices complete, we can now manipulate them to produce coefficients.
- After performing the divine multiplication, **we can observe the estimates** by entering the object name (in this case “b”).



Data Analysis: Regression

- To find the standard errors, we need to compute both the **variance of the residuals** and the cov matrix of the x's.
- The **sqrt of the diagonal elements of this var-cov matrix** will give us the standard errors.
- Other test statistics can be easily computed.
- View the standard errors.

Rintro - Notepad

File Edit Format View Help

```
#Hand-rolled OLS


     x<-as.matrix(cbind(int=1,income,presvote,pressup))
     y<-as.vector(repvshr)
     i<-diag(1,nrow=nrow(x),ncol=ncol(x))

     n<-length(y)
     p<-ncol(x)-1


     xy<-t(x)%*%y                    # X'Y
     xxi<-solve(t(x)%*%x)            #(X'X)^(-1)
     h<-x%*%xxi%*%t(x)               #hat matrix of x
     i<-diag(1,nrow=n,ncol=n)
     b<-as.vector(xxi%*%xy)          #estimated coefficients

     names(b)<-colnames(x)

     yhat<-as.vector(x%*%b)          #predicted values for y
     res<-y-yhat  # or (i-h)%*%y     #model residuals

     sst<-sum((y-mean(y))^2)         #Total sum of sqares
     sse<-t(res)%*%res               # or sum(res^2) which is also t(res)%*%res
     ssm<-sst-sse                    #sum of squares for model (regression)

     df.e<-(n-p-1)                   #degrees of freedom for error
     df.t<-(n-1)                     #total degrees of freedom
     df.m<-df.t-df.e                 #degrees of freedom for model

     s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
     sigma2<-as.vector(sse/(n-p))
     r2<-1-(sse/sst)
     r2.adj<-1-((sse/df.e)/(sst/df.t))
     aic<-n*log(sse/n)+2*(p+1)
     cp<-(sse/s2)-(n-2*(p+1))
     f<-(ssm/df.m)/(sse/df.e)
     pvalue<-1-pf(f,df.m,df.e)

     b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
     b.t.statistic<-b/b.standard.errors            #t statistic for st. errors
     b.t.prob<-2*(1-pt(b.t.statistic,df.e))        #alpha 0.05
```

Data Analysis: Regression

- To find the standard errors, we need to compute both the variance of the residuals and the cov matrix of the x's.
- The sqrt of the diagonal elements of this var-cov matrix will give us the standard errors.
- **Other test statistics can be easily computed.**
- View the standard errors.

Rintro - Notepad

File Edit Format View Help

```
#Hand-rolled OLS

     x<-as.matrix(cbind(int=1,income,presvote,pressup))
     y<-as.vector(repvshr)
     i<-diag(1,nrow=nrow(x),ncol=ncol(x))

     n<-length(y)
     p<-ncol(x)-1

     xy<-t(x)%*%y                    # X'Y
     xxi<-solve(t(x)%*%x)            #(X'X)^(-1)
     h<-x%*%xxi%*%t(x)               #hat matrix of x
     i<-diag(1,nrow=n,ncol=n)
     b<-as.vector(xxi%*%xy)          #estimated coefficients

     names(b)<-colnames(x)

     yhat<-as.vector(x%*%b)          #predicted values for y
     res<-y-yhat  # or (i-h)%*%y     #model residuals

     sst<-sum((y-mean(y))^2)         #Total sum of sqares
     sse<-t(res)%*%res               # or sum(res^2) which is also t(res)%*%res
     ssm<-sst-sse                    #sum of squares for model (regression)

     df.e<-(n-p-1)                   #degrees of freedom for error
     df.t<-(n-1)                     #total degrees of freedom
     df.m<-df.t-df.e                 #degrees of freedom for model

     s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
     sigma2<-as.vector(sse/(n-p))
     r2<-1-(sse/sst)
     r2.adj<-1-((sse/df.e)/(sst/df.t))
     aic<-n*log(sse/n)+2*(p+1)
     cp<-(sse/s2)-(n-2*(p+1))
     f<-(ssm/df.m)/(sse/df.e)
     pvalue<-1-pf(f,df.m,df.e)

     b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
     b.t.statistic<-b/b.standard.errors           #t statistic for st. errors
     b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05
```

Data Analysis: Regression

- To find the standard errors, we need to compute both the variance of the residuals and the cov matrix of the x's.
- The sqrt of the diagonal elements of this var-cov matrix will give us the standard errors.
- Other test statistics can be easily computed.
- **View the standard errors.**

R Console

```
>       names(b)<-colnames(x)
>
>       yhat<-as.vector(x%*%b)          #predicted values for y
>       res<-y-yhat  # or (i-h)%*%y     #model residuals
>
>       sst<-sum((y-mean(y))^2)         #Total sum of sqares
>       sse<-t(res)%*%res               # or sum(res^2) which is also t($
>       ssm<-sst-sse                    #sum of squares for model (regre$
>
>       df.e<-(n-p-1)                   #degrees of freedom for error
>       df.t<-(n-1)                     #total degrees of freedom
>       df.m<-df.t-df.e                 #degrees of freedom for model
>
>       s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
>       sigma2<-as.vector(sse/(n-p))
>       r2<-1-(sse/sst)
>       r2.adj<-1-((sse/df.e)/(sst/df.t))
>       aic<-n*log(sse/n)+2*(p+1)
>       cp<-(sse/s2)-(n-2*(p+1))
>       f<-(ssm/df.m)/(sse/df.e)
>       pvalue<-1-pf(f,df.m,df.e)
>
>       b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient stan$
>       b.t.statistic<-b/b.standard.errors           #t statistic for $
>       b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05
> b
          int        income      presvote       pressup 
-7.295361e+01  6.743087e-04  6.021832e-01  8.088049e-01 
> b.standard.errors
         int       income     presvote      pressup 
2.474573e+01 3.878401e-04 3.105179e-01 2.208367e-01 
> 
```

Data Analysis: Regression

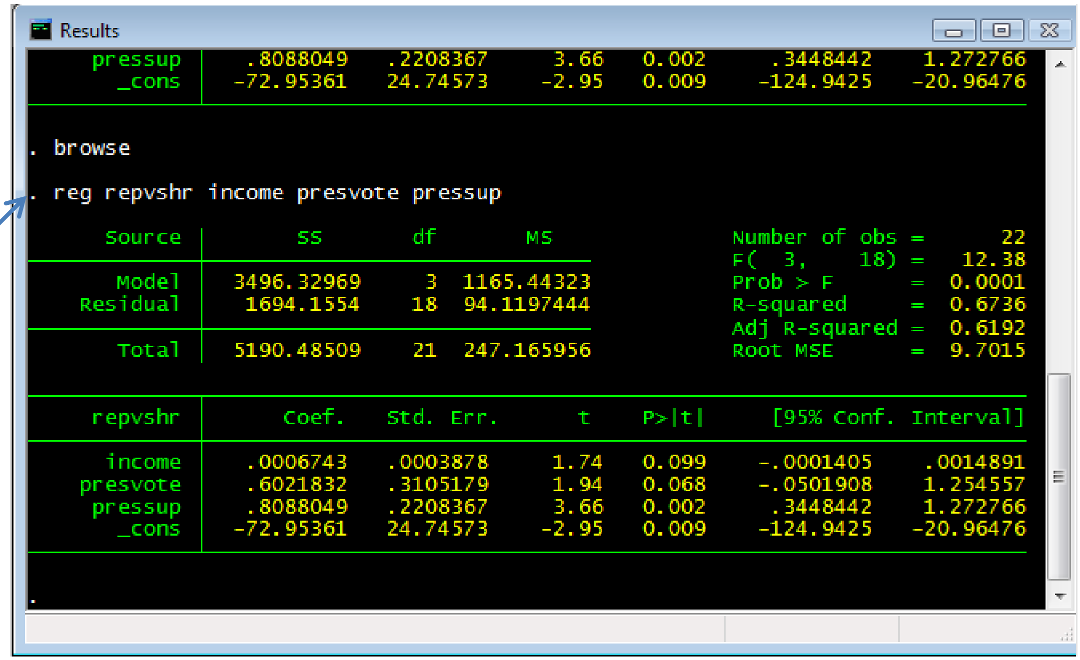
- Time to Compare
- **Use the `lm()` command** to estimate the model using R's canned procedure
- As we can see, the estimates are very similar

R Console

```
> b
          int        income      presvote       pressup
-7.295361e+01  6.743087e-04  6.021832e-01  8.088049e-01
> b.standard.errors
         int        income      presvote       pressup
2.474573e+01 3.878401e-04 3.105179e-01 2.208367e-01
> #OLS using the canned R procedure (i.e. the 'lm' command)
>
> canned.ols <- lm(repvshr ~ income + presvote + pressup)
> summary(canned.ols)

Call:
lm(formula = repvshr ~ income + presvote + pressup)

Residuals:
     Min       1Q   Median       3Q      Max
-21.8269  -4.7384   0.6484   5.8808  14.8608

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -7.295e+01  2.475e+01  -2.948  0.00860 **
income       6.743e-04  3.878e-04   1.739  0.09918 .
presvote     6.022e-01  3.105e-01   1.939  0.06830 .
pressup      8.088e-01  2.208e-01   3.662  0.00178 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.702 on 18 degrees of freedom
Multiple R-Squared: 0.6736,     Adjusted R-squared: 0.6192
F-statistic: 12.38 on 3 and 18 DF,  p-value: 0.0001245

>
```

Data Analysis: Regression

- Time to Compare
- We can also see how both the hand-rolled and canned OLS procedures stack up to Stata
- **Use the `reg` command** to estimate the model
- As we can see, the estimates are once again very similar



```
Results
     pressup |   .8088049   .2208367     3.66   0.002     .3448442    1.272766
       _cons |  -72.95361   24.74573    -2.95   0.009    -124.9425   -20.96476
------------------------------------------------------------------------------

. browse

. reg repvshr income presvote pressup

      Source |       SS       df       MS              Number of obs =      22
-------------+------------------------------           F(  3,    18) =   12.38
       Model |  3496.32969     3  1165.44323           Prob > F      =  0.0001
    Residual |   1694.1554    18  94.1197444           R-squared     =  0.6736
-------------+------------------------------           Adj R-squared =  0.6192
       Total |  5190.48509    21  247.165956           Root MSE      =  9.7015

------------------------------------------------------------------------------
     repvshr |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      income |   .0006743   .0003878     1.74   0.099    -.0001405    .0014891
    presvote |   .6021832   .3105179     1.94   0.068    -.0501908    1.254557
     pressup |   .8088049   .2208367     3.66   0.002     .3448442    1.272766
       _cons |  -72.95361   24.74573    -2.95   0.009    -124.9425   -20.96476
------------------------------------------------------------------------------

.
```

Data Analysis: Regression

R Console

```
> b
          int        income      presvote       pressup
-7.295361e+01  6.743087e-04  6.021832e-01  8.088049e-01
> b.standard.errors
         int       income      presvote       pressup
2.474573e+01 3.878401e-04 3.105179e-01 2.208367e-01
> #OLS using the canned R procedure (i.e. the 'lm' command)
>
> canned.ols <- lm(repvshr ~ income + presvote + pressup)
> summary(canned.ols)

Call:
lm(formula = repvshr ~ income + presvote + pressup)

Residuals:
     Min       1Q   Median       3Q      Max
-21.8269  -4.7384   0.6484   5.8808  14.8608

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -7.295e+01  2.475e+01  -2.948  0.00860 **
income       6.743e-04  3.878e-04   1.739  0.09918 .
presvote     6.022e-01  3.105e-01   1.939  0.06830 .
pressup      8.088e-01  2.208e-01   3.662  0.00178 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.702 on 18 degrees of freedom
Multiple R-Squared: 0.6736,     Adjusted R-squared: 0.6192
F-statistic: 12.38 on 3 and 18 DF,  p-value: 0.0001245

>
```

Results

```
     pressup     .8088049   .2208367     3.66   0.002     .3448442    1.272766
       _cons    -72.95361   24.74573    -2.95   0.009    -124.9425   -20.96476
------------------------------------------------------------------------------

. browse

. reg repvshr income presvote pressup

      Source |       SS       df       MS              Number of obs =      22
-------------+------------------------------           F(  3,    18) =   12.38
       Model |  3496.32969     3  1165.44323           Prob > F      =  0.0001
    Residual |  1694.1554     18  94.1197444           R-squared     =  0.6736
-------------+------------------------------           Adj R-squared =  0.6192
       Total |  5190.48509    21  247.165956           Root MSE      =  9.7015

------------------------------------------------------------------------------
     repvshr |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      income |   .0006743   .0003878     1.74   0.099    -.0001405    .0014891
    presvote |   .6021832   .3105179     1.94   0.068    -.0501908    1.254557
     pressup |   .8088049   .2208367     3.66   0.002     .3448442    1.272766
       _cons |  -72.95361   24.74573    -2.95   0.009    -124.9425   -20.96476
------------------------------------------------------------------------------

.
```

Data Analysis: Regression

Other Useful Commands

- **lm**
  – Linear Model
- **lme**
  – Mixed Effects
- **anova**
- **glm**
  - General lm
- **multinom**
  - Multinomial Logit
- **optim**
  - General Optimizer

OLS Diagnostics in R

- Post-estimation diagnostics are key to data analysis
  - We want to make sure we estimated the proper model
  - Besides, Irfan will hurt you if you neglect to do this
- Furthermore, diagnostics allow us the opportunity to show off some of R's graphs
  - R's real strength is that it has virtually unlimited graphing capabilities
  - Of course, such strengths on R's part is dependent on your knowledge of both R and statistics
    - Still, with just some basics we can do some cool graphs

OLS Diagnostics in R

- What could be *unjustifiably* driving our data?
  - Outlier: unusual observation
  - Leverage: ability to change the slope of the regression line
  - Influence: the combined impact of strong leverage and outlier status
    - According to John Fox, influence=leverage*outliers

OLS Diagnostics: Leverage

- Recall our ols model
  - `ols.model1<-lm(formula = repvshr~income+presvote+pressup)`
- Our measure of leverage: is the h_i or "hat value"
  - It's just the predicted values written in terms of h_i
  - Where, H_{ij} is the contribution of observation Y_i to the fitted value Y_j
  - If h_{ij} is large, then the i^{th} observation has a significant impact on the jth fitted value
  - So, skipping the formulas, we know that the larger the hat value the greater the leverage of that observation

OLS Diagnostics: Leverage

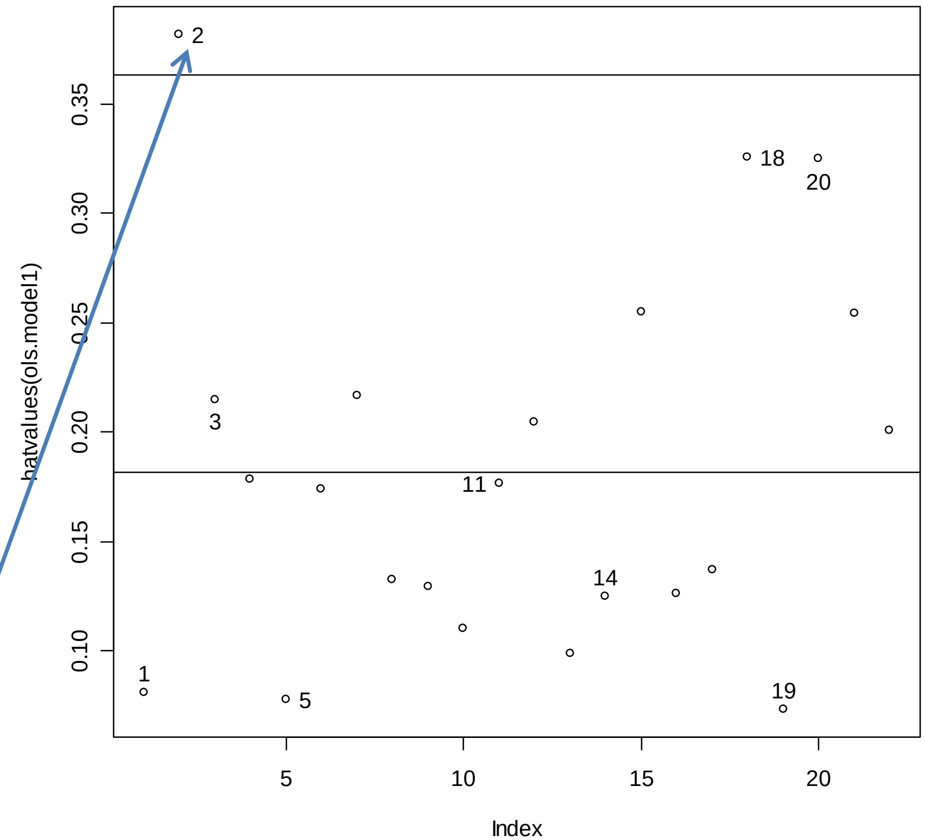
- Find the hat values
  - `hatvalues(ols.model1)`

```
> ##Leverage
> hatvalues(ols.model1)
         1          2          3          4          5          6          7 
0.08058958 0.38217510 0.21508254 0.17839298 0.07791739 0.17390212 0.21652515 
         8          9         10         11         12         13         14 
0.13240657 0.12946990 0.11013685 0.17680240 0.20482571 0.09892587 0.12505991 
        15         16         17         18         19         20         21 
0.25521188 0.12628592 0.13708349 0.32578291 0.07297085 0.32496207 0.25453795 
        22 
0.20095287 
> 
> avg.mod1<-ncol(x)/nrow(x)
> avg.mod1
[1] 0.1818182
```

- Calculate the average hat value
  - `avg.mod1<-ncol(x)/nrow(x)`

OLS Diagnostics: Leverage

- But a picture is worth a hundred numbers?
- Graph the hat values with lines for the average, twice the avg (large samples) and three times the avg (small samples) hat values
  - `plot(hatvalues(ols.model1))`
  - `abline(h=1*(ncol(x))/nrow(x))`
  - `abline(h=2*(ncol(x))/nrow(x))`
  - `abline(h=3*(ncol(x))/nrow(x))`
  - `identify(hatvalues(ols.model1))`
    - `identify` lets us select the data points in the new graph
- State #2 is over twice the avg
- Nothing above three times



OLS Diagnostics: Outliers

- Can we find any data points that are unusual for Y given the Xs?
- Use studentized residuals

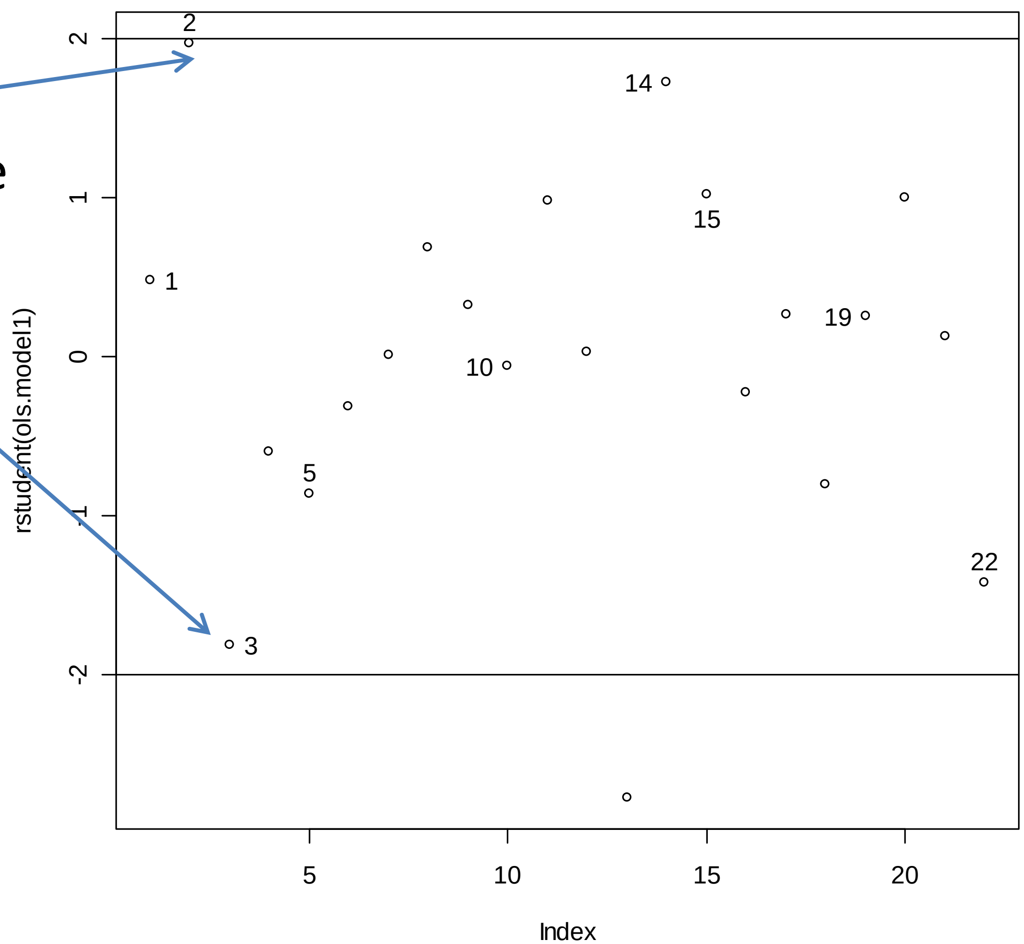
$$u_i^* = \frac{u_i}{\sigma_{u(-1)}\sqrt{1-h_i}}$$

  - We can see whether there is a significant change in the model
  - If their absolute values are larger than 2, then the corresponding observations are likely to be outliers)
  - `rstudent(ols.model1)`

```
> rstudent(ols.model1)
          1           2           3           4           5           6 
 0.48019795  1.97192270 -1.81307635 -0.59849094 -0.86387841 -0.31785263 
          7           8           9          10          11          12 
 0.01244686  0.68902256  0.31806953 -0.05965655  0.97657494  0.02443043 
         13          14          15          16          17          18 
-2.77709792  1.72517421  1.02255885 -0.22885529  0.26198911 -0.80877619 
         19          20          21          22 
 0.25367148  0.99768167  0.12528015 -1.42108584 
> 
```

OLS Diagnostics: Outliers

- Again, let's plot them with lines for 2 & -2
- States 2 and 3 appear to be outliers, or darn close
- We should definitely take a look at what makes these states unusual…
  – Perhaps there is a mistake in data entry
  – Perhaps the model is misspecified in terms of functional form (forthcoming) or omitted vars
  – Maybe you can throw out your bad observation
  – If you must include the bad observation, try robust regression

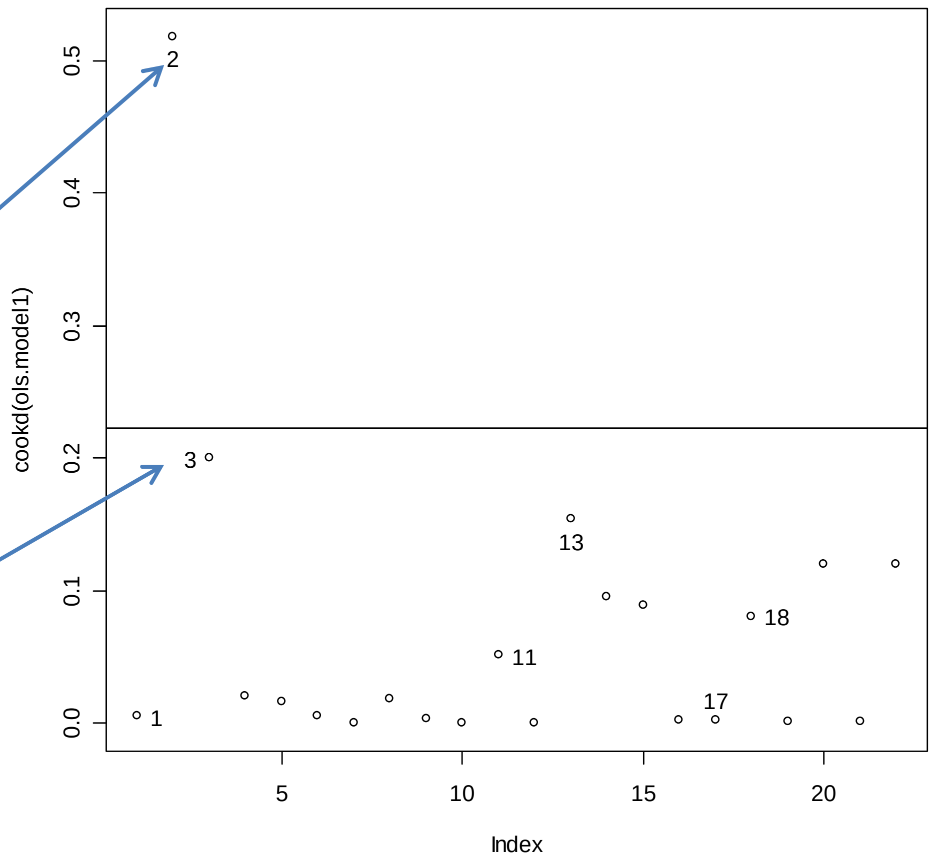


OLS Diagnostics: Influence

- Cook's D gives a kind of summary for each observation's influence

$$D_i = \frac{u_i'^2}{k+1} * \frac{h}{1-h_i}$$

- If Cook's D is greater than 4/(n-k-1), then the observation is said to exert undue influence
- Let's just plot it
  - `plot(cookd(ols.model1))`
  - `abline(h=4/(nrow(x)-ncol(x)))`
  - `Identify(cookd(ols.model1))`
- States 2 and (maybe) 3 are in the trouble zone



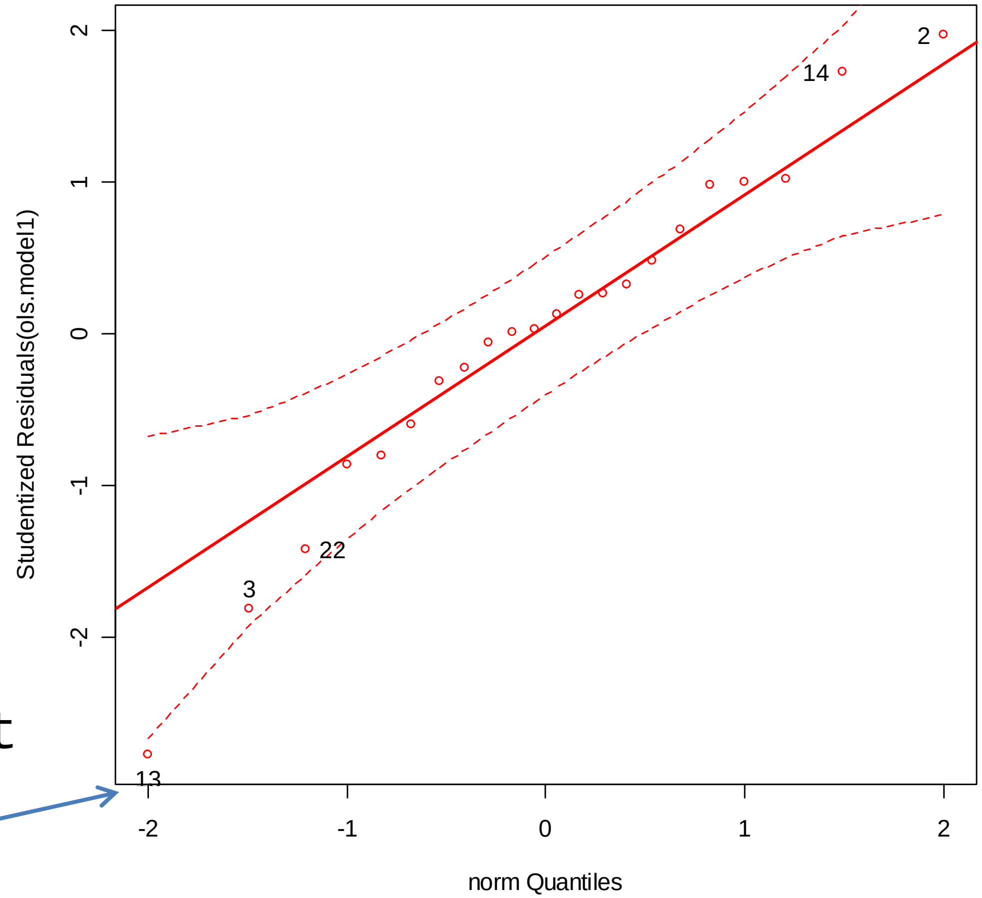
OLS Diagnostics: Influence

- For a host of measures of influence, including df betas and df fits
  - `influence.measures(ols.model1)`
- dfbeta gives the influence of an observation on the coefficients – or the change in iv's coefficient caused by deleting a single observation
- Simple commands for partial regression plots can be found on Fox's website…

```
> ##Other measures of influence, including df-beta and df-fit
> influence.measures(ols.model1)
Influence measures of
         lm(formula = repvshr ~ income + presvote + pressup) :

     dfb.1_ dfb.incm  dfb.prsv  dfb.prss    dffit cov.r   cook.d    hat inf
1  0.047591 -0.08463 -3.65e-03  0.020468  0.14217 1.295 5.28e-03 0.0806
2 -1.312887  1.36488  5.54e-01  0.206512  1.55092 0.892 5.18e-01 0.3822   *
3  0.589635 -0.58486  2.12e-01 -0.542793 -0.94909 0.790 2.00e-01 0.2151
4  0.018551 -0.15898  9.62e-02  0.003785 -0.27888 1.407 2.02e-02 0.1784
5  0.140077 -0.13349 -3.66e-02 -0.066607 -0.25112 1.148 1.60e-02 0.0779
6  0.074461 -0.02307 -1.05e-01  0.003705 -0.14584 1.486 5.60e-03 0.1739
7  0.000784  0.00338  6.08e-05 -0.003494  0.00654 1.604 1.13e-05 0.2165
8  0.137543  0.00171  3.56e-02 -0.197503  0.26917 1.298 1.87e-02 0.1324
9  0.012258 -0.06613 -2.85e-02  0.070448  0.12266 1.410 3.96e-03 0.1295
10 -0.013501  0.01606  4.56e-03 -0.000609 -0.02099 1.411 1.17e-04 0.1101
11  0.181001 -0.17994 -3.79e-01  0.264511  0.45258 1.227 5.13e-02 0.1768
12  0.002175  0.00537  2.43e-03 -0.008720  0.01240 1.580 4.07e-05 0.2048
13 -0.294634  0.56467 -1.25e-01 -0.026032 -0.92017 0.312 1.54e-01 0.0989   *
14 -0.398730  0.13983  2.24e-01  0.261259  0.65223 0.753 9.58e-02 0.1251
15  0.177517 -0.31912 -4.41e-01  0.439292  0.59858 1.329 8.93e-02 0.2552
16 -0.000611  0.04267 -2.17e-03 -0.037603 -0.08701 1.421 2.00e-03 0.1263
17  0.000169 -0.00028 -7.43e-02  0.072067  0.10442 1.433 2.87e-03 0.1371
18 -0.346076  0.04901  3.86e-01  0.055702 -0.56220 1.603 8.06e-02 0.3258
19  0.009685 -0.00750  3.67e-02 -0.032806  0.07117 1.335 1.34e-03 0.0730
20  0.185511 -0.04938  4.76e-01 -0.595824  0.69222 1.483 1.20e-01 0.3250
21  0.059471 -0.05104  3.78e-03 -0.037402  0.07321 1.680 1.42e-03 0.2545   *
22  0.352385 -0.14803 -5.73e-01  0.119730 -0.71266 1.004 1.20e-01 0.2010
>
```

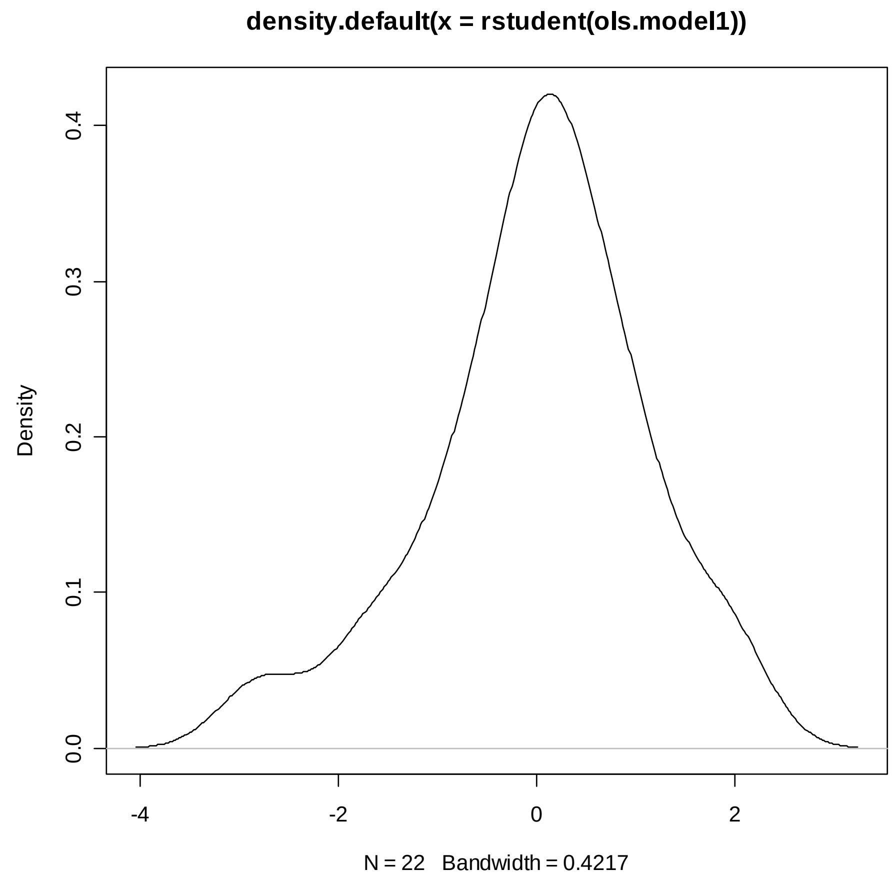
OLS Diagnostics: Normality

- Is our data distributed normally?
- Was it correct to use a linear model?
- Use a quantile plot (qq plot) to check
  - Plots empirical quantiles of a variable against studentized residuals
  - Looking for obs on a straight line
  - In R it is simple to plot the error bands as well
  - Deviation requires us to transform our variables
- `qq.plot(ols.model1,distribution="norm")`
- The problems are again 2 and 13, with 3, 22 and 14 bordering on trouble this time around



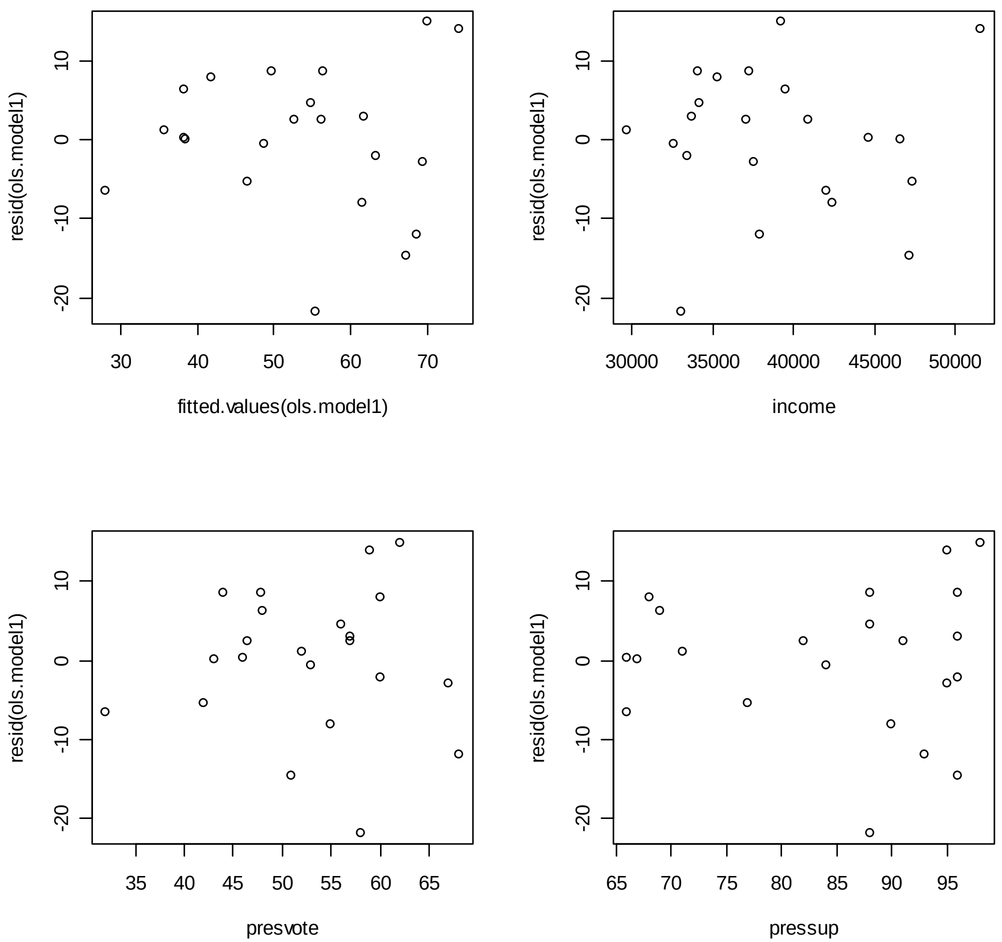
OLS Diagnostics: Normality

- A simple density plot of the studentized residuals helps to determine the nature of our data
- The apparent deviation from the normal curve is not severe, but there certainly seems to be a slight negative skew



OLS Diagnostics: Error Variance

- We can also easily look for heteroskedasticity
- Plotting the residuals against the fitted values and the continuous independent variables let's us examine our statistical model for the presence of unbalanced error variance
  - `par(mfrow=c(2,2))`
  - `plot(resid(ols.model1) ~fitted.values(ols.model1))`
  - `plot(resid(ols.model1) ~income)`
  - `plot(resid(ols.model1) ~presvote)`
  - `plot(resid(ols.model1) ~pressup)`



OLS Diagnostics: Error Variance

- Formal tests for heteroskedasticity are available from the `lmtest` library
  - `library(lmtest)`
  - `bptest(ols.model1)` will give you the Breusch-Pagan test stat
  - `gqtest(ols.model1)` will give you the Goldfeld-Quandttest stat
  - `hmctest(ols.model1)` will give you the Harrison-McCabe test stat

```
> ##Breusch-Pagan, Goldfeld-Quandt, and Harrison-McCabe tests
> bptest(ols.model1)

        studentized Breusch-Pagan test

data:  ols.model1
BP = 3.2325, df = 3, p-value = 0.3571

> gqtest(ols.model1)

        Goldfeld-Quandt test

data:  ols.model1
GQ = 1.6338, df1 = 7, df2 = 7, p-value = 0.2664

> hmctest(ols.model1)

        Harrison-McCabe test

data:  ols.model1
HMC = 0.3878, p-value = 0.235
```

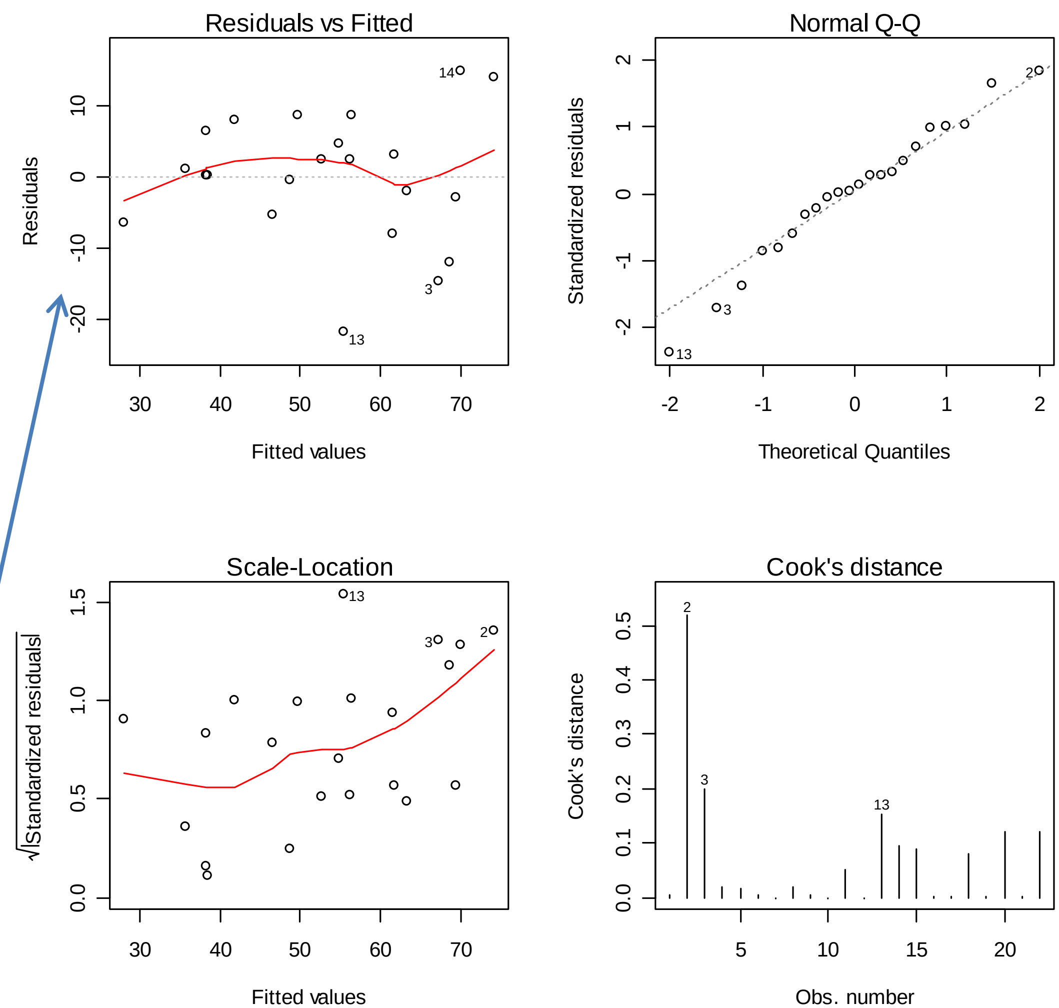
OLS Diagnostics: Collinearity

- Finally, let's look out for collinearity
- To get the variance inflation factors
  - `vif(ols.model1)`
- Let's look at the condition index from the `perturb` library
  - `library(perturb)`
  - `colldiag(ols.model1)`
- Issues here is the largest condition index
- If it is larger than 30, *Houston we have…*

```
>
> ##Variance inflation factors
> vif(ols.model1)
  income presvote  pressup
1.127017 1.636216 1.482685
>
>
> ##Obtain the condition index
> colldiag(ols.model1)
Condition
Index    Variance Decomposition Proportions
           intercept income presvote pressup
1   1.000  0.000     0.001  0.001    0.001
2  10.920  0.004     0.307  0.162    0.030
3  21.626  0.012     0.030  0.588    0.926
4  27.883  0.983     0.662  0.250    0.044
>
```

OLS Diagnostics: Shortcut

- My favorite shortcut command to get you four essential diagnostic plots after you run your model
  - `plot(ols.model1, which=1:4)`
- Now you have no excuse not to run some diagnostics!
- Btw, look at the high residuals in the rvf plot for 14, 13 and 3 – suggesting outliers



The Final Act: Loops and Functions

- As was mentioned above, R's biggest asset is its flexibility. Loops and functions directly utilize this asset.

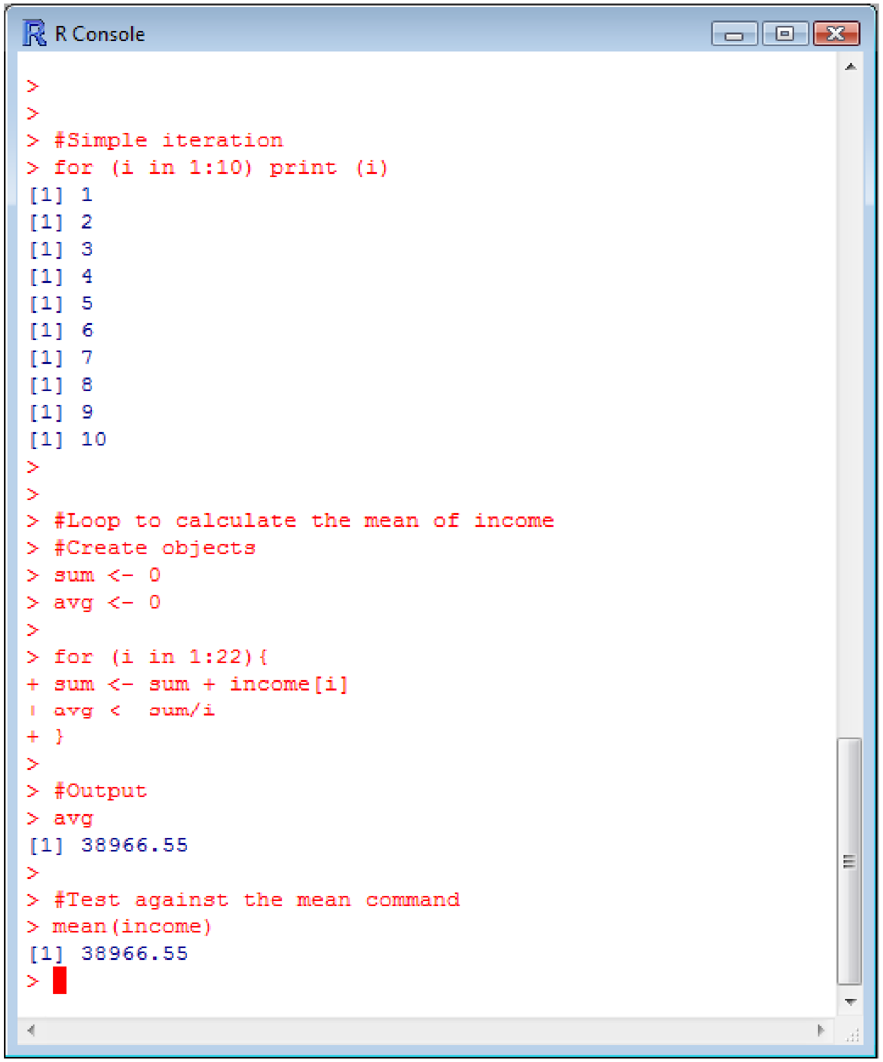
- Loops can be implemented for a number of purposes, essentially when repeated actions are needed (i.e. simulations).

- Functions allow us to create our own commands. This is especially useful when a canned procedure does not exist. We will create our own OLS function with the hand-rolled code used earlier.

Loops

- `for` loops are the most common and the only type of loop we will look at today.

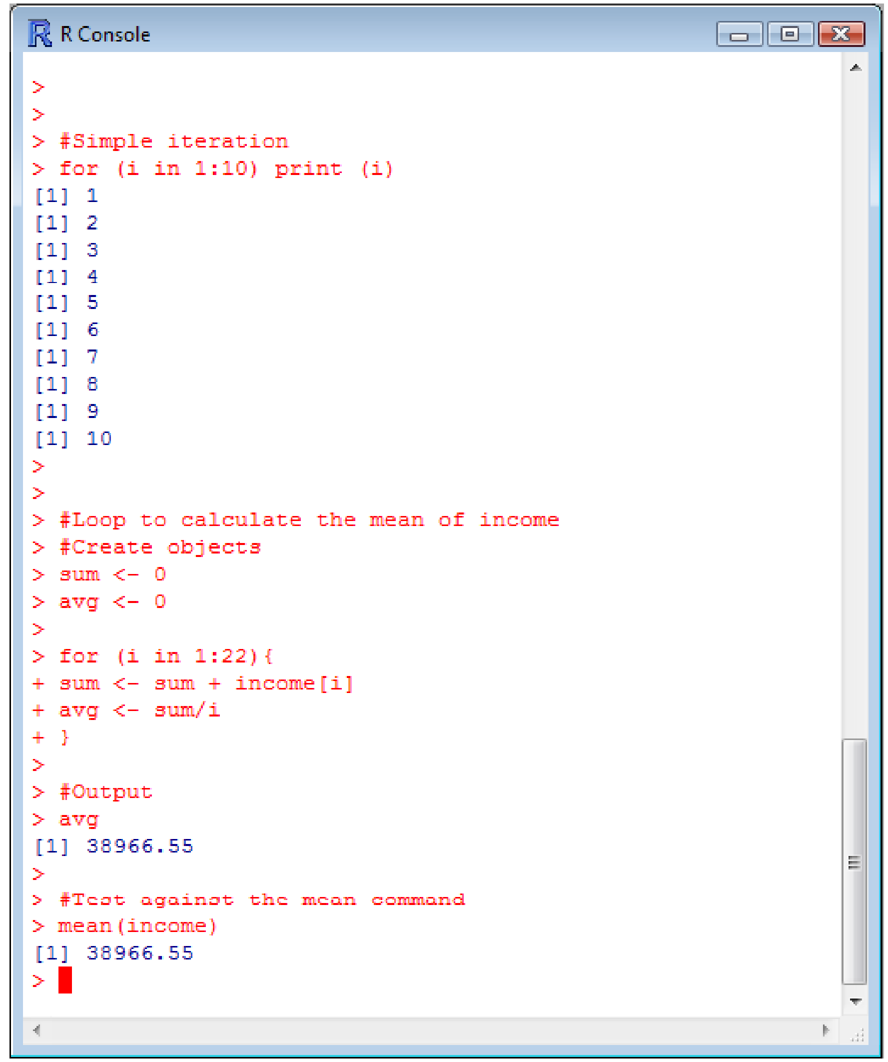
- The first loop command at the right shows simple loop iteration.



```
R Console
>
>
> #Simple iteration
> for (i in 1:10) print (i)
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
>
>
> #Loop to calculate the mean of income
> #Create objects
> sum <- 0
> avg <- 0
>
> for (i in 1:22){
+ sum <- sum + income[i]
+ avg <- sum/i
+ }
>
> #Output
> avg
[1] 38966.55
>
> #Test against the mean command
> mean(income)
[1] 38966.55
>
```

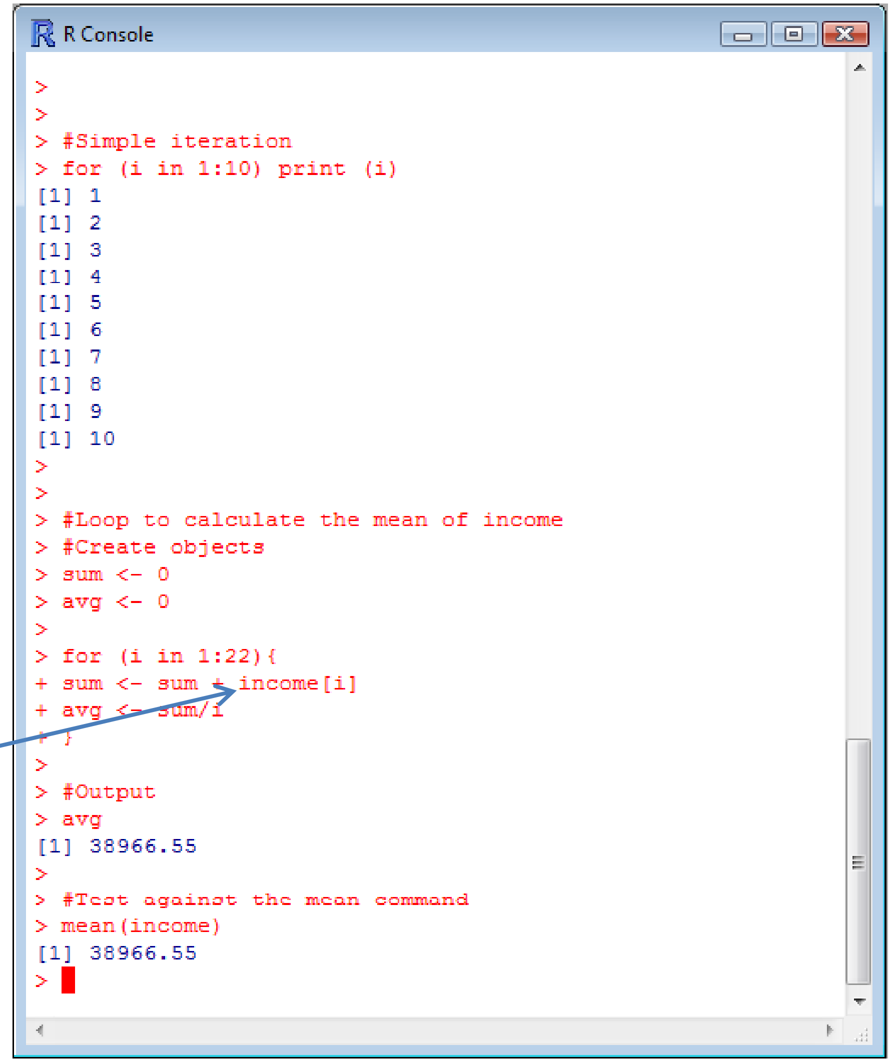
Loops

- However, we can also see how loops can be a little more useful.
- **The second example at right (although inefficient) calculates the mean of income**
- Note how the index accesses elements of the "income" vector.
- Loops and Monte Carlo



Loops

- However, we can also see how loops can be a little more useful.
- The second example at right (although inefficient) calculates the mean of income
- **Note how the index accesses elements of the “income” vector.**
- Loops and Monte Carlo



Functions

- Now we will make our own linear regression function using our hand-rolled OLS code
- Functions require **inputs** (which are the objects to be utilized) and **arguments** (which are the commands that the function performs)
- The actual estimation procedure does not change. However, some changes are made.

Rintro - Notepad

File Edit Format View Help

```
#Designing an OLS function (based on hand-rolled ols commands above)

ols<-function(y,x){



    x<-as.matrix(cbind(int=1,x))
    y<-as.vector(y)
    i<-diag(1,nrow=nrow(x),ncol=ncol(x))

    n<-length(y)
    p<-ncol(x)-1

    xy<-t(x)%*%y                  # X'Y
    xxi<-solve(t(x)%*%x)          #(X'X)^(-1)
    h<-x%*%xxi%*%t(x)             #hat matrix of x
    i<-diag(1,nrow=n,ncol=n)
    b<-as.vector(xxi%*%xy)        #estimated coefficients

    names(b)<-colnames(x)

    yhat<-as.vector(x%*%b)        #predicted values for y
    res<-y-yhat  # or (i-h)%*%y   #model residuals

    sst<-sum((y-mean(y))^2)       #Total sum of sqares
    sse<-t(res)%*%res               # or sum(res^2) which is also t(res)%*%res
    ssm<-sst-sse                  #sum of squares for model (regression)

    df.e<-(n-p-1)                 #degrees of freedom for error
    df.t<-(n-1)                   #total degrees of freedom
    df.m<-df.t-df.e               #degrees of freedom for model

    s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
    sigma2<-as.vector(sse/(n-p))
    r2<-1-(sse/sst)
    r2.adj<-1-((sse/df.e)/(sst/df.t))
    aic<-n*log(sse/n)+2*(p+1)
    cp<-(sse/s2)-(n-2*(p+1))
    f<-(ssm/df.m)/(sse/df.e)
    pvalue<-1-pf(f,df.m,df.e)

    b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
    b.t.statistic<-b/b.standard.errors           #t statistic for st. errors
    b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05


    b.table<-cbind(est=b,b.se=b.standard.errors,t.stat=b.t.statistic,p=b.t.prob)


  return(list(b=b.table
            ))
}
```

Functions

- **First, we have to tell R that we are creating a function. We'll name it `ols`.**

- This lets us generalize the procedure to multiple objects.

- Second, we have to tell the function what we want "returned" or what we want the output to look like.

Rintro - Notepad

File Edit Format View Help

```
#Designing an OLS function (based on hand-rolled ols commands above)

ols<-function(y,x){


    x<-as.matrix(cbind(int=1,x))
    y<-as.vector(y)
    i<-diag(1,nrow=nrow(x),ncol=ncol(x))

    n<-length(y)
    p<-ncol(x)-1

    xy<-t(x)%*%y                     # X'Y
    xxi<-solve(t(x)%*%x)             #(X'X)^(-1)
    h<-x%*%xxi%*%t(x)                #hat matrix of x
    i<-diag(1,nrow=n,ncol=n)
    b<-as.vector(xxi%*%xy)           #estimated coefficients

    names(b)<-colnames(x)

    yhat<-as.vector(x%*%b)           #predicted values for y
    res<-y-yhat  # or (i-h)%*%y      #model residuals

    sst<-sum((y-mean(y))^2)          #Total sum of sqares
    sse<-t(res)%*%res                  # or sum(res^2) which is also t(res)%*%res
    ssm<-sst-sse                     #sum of squares for model (regression)

    df.e<-(n-p-1)                    #degrees of freedom for error
    df.t<-(n-1)                      #total degrees of freedom
    df.m<-df.t-df.e                  #degrees of freedom for model

    s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
    sigma2<-as.vector(sse/(n-p))
    r2<-1-(sse/sst)
    r2.adj<-1-((sse/df.e)/(sst/df.t))
    aic<-n*log(sse/n)+2*(p+1)
    cp<-(sse/s2)-(n-2*(p+1))
    f<-(ssm/df.m)/(sse/df.e)
    pvalue<-1-pf(f,df.m,df.e)

    b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
    b.t.statistic<-b/b.standard.errors           #t statistic for st. errors
    b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05


    b.table<-cbind(est=b,b.se=b.standard.errors,t.stat=b.t.statistic,p=b.t.prob)


  return(list(b=b.table
              ))
}
```

Functions

- First, we have to tell R that we are creating a function. We'll name it `ols`.

- **This lets us generalize the procedure to multiple objects.**

- Second, we have to tell the function what we want "returned" or what we want the output to look like.

Rintro - Notepad

File Edit Format View Help

```
#Designing an OLS function (based on hand-rolled ols commands above)

ols<-function(y,x){


     x<-as.matrix(cbind(int=1,x))
     y<-as.vector(y)
     i<-diag(1,nrow=nrow(x),ncol=ncol(x))

     n<-length(y)
     p<-ncol(x)-1

     xy<-t(x)%*%y                  # X'Y
     xxi<-solve(t(x)%*%x)          #(X'X)^(-1)
     h<-x%*%xxi%*%t(x)             #hat matrix of x
     i<-diag(1,nrow=n,ncol=n)
     b<-as.vector(xxi%*%xy)        #estimated coefficients

     names(b)<-colnames(x)

     yhat<-as.vector(x%*%b)        #predicted values for y
     res<-y-yhat  # or (i-h)%*%y   #model residuals

     sst<-sum((y-mean(y))^2)       #Total sum of sqares
     sse<-t(res)%*%res               # or sum(res^2) which is also t(res)%*%res
     ssm<-sst-sse                  #sum of squares for model (regression)

     df.e<-(n-p-1)                 #degrees of freedom for error
     df.t<-(n-1)                   #total degrees of freedom
     df.m<-df.t-df.e               #degrees of freedom for model

     s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
     sigma2<-as.vector(sse/(n-p))
     r2<-1-(sse/sst)
     r2.adj<-1-((sse/df.e)/(sst/df.t))
     aic<-n*log(sse/n)+2*(p+1)
     cp<-(sse/s2)-(n-2*(p+1))
     f<-(ssm/df.m)/(sse/df.e)
     pvalue<-1-pf(f,df.m,df.e)

     b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
     b.t.statistic<-b/b.standard.errors           #t statistic for st. errors
     b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05


     b.table<-cbind(est=b,b.se=b.standard.errors,t.stat=b.t.statistic,p=b.t.prob)


  return(list(b=b.table
              ))
}
```

Functions

- First, we have to tell R that we are creating a function. We'll name it `ols`.
- This lets us generalize the procedure to multiple objects.
- **Second, we have to tell the function what we want "returned" or what we want the output to look like.**

Rintro - Notepad

File Edit Format View Help

```
#Designing an OLS function (based on hand-rolled ols commands above)

ols<-function(y,x){



    x<-as.matrix(cbind(int=1,x))
    y<-as.vector(y)
    i<-diag(1,nrow=nrow(x),ncol=ncol(x))

    n<-length(y)
    p<-ncol(x)-1

    xy<-t(x)%*%y                # X'Y
    xxi<-solve(t(x)%*%x)        #(X'X)^(-1)
    h<-x%*%xxi%*%t(x)           #hat matrix of x
    i<-diag(1,nrow=n,ncol=n)
    b<-as.vector(xxi%*%xy)      #estimated coefficients

    names(b)<-colnames(x)

    yhat<-as.vector(x%*%b)      #predicted values for y
    res<-y-yhat  # or (i-h)%*%y #model residuals

    sst<-sum((y-mean(y))^2)     #Total sum of sqares
    sse<-t(res)%*%res             # or sum(res^2) which is also t(res)%*%res
    ssm<-sst-sse                #sum of squares for model (regression)

    df.e<-(n-p-1)               #degrees of freedom for error
    df.t<-(n-1)                 #total degrees of freedom
    df.m<-df.t-df.e             #degrees of freedom for model

    s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
    sigma2<-as.vector(sse/(n-p))
    r2<-1-(sse/sst)
    r2.adj<-1-((sse/df.e)/(sst/df.t))
    aic<-n*log(sse/n)+2*(p+1)
    cp<-(sse/s2)-(n-2*(p+1))
    f<-(ssm/df.m)/(sse/df.e)
    pvalue<-1-pf(f,df.m,df.e)

    b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
    b.t.statistic<-b/b.standard.errors           #t statistic for st. errors
    b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05


    b.table<-cbind(est=b,b.se=b.standard.errors,t.stat=b.t.statistic,p=b.t.prob)


  return(list(b=b.table
             ))
}
```

Functions

OLS: Hand-rolled vs Function

```
#Hand-rolled OLS

     x<-as.matrix(cbind(int=1,income,presvote,pressup))
     y<-as.vector(repvshr)
     i<-diag(1,nrow=nrow(x),ncol=ncol(x))

     n<-length(y)
     p<-ncol(x)-1


     xy<-t(x)%*%y                   # X'Y
     xxi<-solve(t(x)%*%x)           #(X'X)^(-1)
     h<-x%*%xxi%*%t(x)              #hat matrix of x
     i<-diag(1,nrow=n,ncol=n)
     b<-as.vector(xxi%*%xy)         #estimated coefficients

     names(b)<-colnames(x)

     yhat<-as.vector(x%*%b)         #predicted values for y
     res<-y-yhat  # or (i-h)%*%y    #model residuals

     sst<-sum((y-mean(y))^2)        #Total sum of sqares
     sse<-t(res)%*%res              # or sum(res^2) which is also t(res)%*%res
     ssm<-sst-sse                   #sum of squares for model (regression)

     df.e<-(n-p-1)                  #degrees of freedom for error
     df.t<-(n-1)                    #total degrees of freedom
     df.m<-df.t-df.e                #degrees of freedom for model

     s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
     sigma2<-as.vector(sse/(n-p))
     r2<-1-(sse/sst)
     r2.adj<-1-((sse/df.e)/(sst/df.t))
     aic<-n*log(sse/n)+2*(p+1)
     cp<-(sse/s2)-(n-2*(p+1))
     f<-(ssm/df.m)/(sse/df.e)
     pvalue<-1-pf(f,df.m,df.e)

     b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
     b.t.statistic<-b/b.standard.errors           #t statistic for st. errors
     b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05
```

```
#Designing an OLS function (based on hand-rolled ols commands above)

ols<-function(y,x){


    x<-as.matrix(cbind(int=1,x))
    y<-as.vector(y)
    i<-diag(1,nrow=nrow(x),ncol=ncol(x))

    n<-length(y)
    p<-ncol(x)-1

    xy<-t(x)%*%y                 # X'Y
    xxi<-solve(t(x)%*%x)         #(X'X)^(-1)
    h<-x%*%xxi%*%t(x)            #hat matrix of x
    i<-diag(1,nrow=n,ncol=n)
    b<-as.vector(xxi%*%xy)       #estimated coefficients

    names(b)<-colnames(x)

    yhat<-as.vector(x%*%b)       #predicted values for y
    res<-y-yhat  # or (i-h)%*%y  #model residuals

    sst<-sum((y-mean(y))^2)      #Total sum of sqares
    sse<-t(res)%*%res              # or sum(res^2) which is also t(res)%*%res
    ssm<-sst-sse                 #sum of squares for model (regression)

    df.e<-(n-p-1)                #degrees of freedom for error
    df.t<-(n-1)                  #total degrees of freedom
    df.m<-df.t-df.e              #degrees of freedom for model

    s2<-as.vector(sse/df.e) # or (t(res)%*%res)?(n-p-1)
    sigma2<-as.vector(sse/(n-p))
    r2<-1-(sse/sst)
    r2.adj<-1-((sse/df.e)/(sst/df.t))
    aic<-n*log(sse/n)+2*(p+1)
    cp<-(sse/s2)-(n-2*(p+1))
    f<-(ssm/df.m)/(sse/df.e)
    pvalue<-1-pf(f,df.m,df.e)

    b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)  #coefficient standard errors
    b.t.statistic<-b/b.standard.errors           #t statistic for st. errors
    b.t.prob<-2*(1-pt(b.t.statistic,df.e))       #alpha 0.05


    b.table<-cbind(est=b,b.se=b.standard.errors,t.stat=b.t.statistic,p=b.t.prob)


  return(list(b=b.table
           ))
}
```

Functions

- **Implementing our new function `ols`, we get precisely the output that we asked for.**

- We can check this against the results produced by the standard `lm` function.

R Console

```
+     b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)    #coeffici$
+     b.t.statistic<-b/b.standard.errors             #t statis$
+     b.t.prob<-2*(1-pt(b.t.statistic,df.e))         #alpha 0.$
+
+
+     b.table<-cbind(est=b,b.se=b.standard.errors,t.stat=b.t$
+
+
+   return(list(b=b.table
+               ))
+ }
> #Output of new ols command
> ols(repvshr, income)
$b
             est         b.se    t.stat          p
int 4.585949e+01 2.382350e+01 1.9249684 0.06857514
x   1.979220e-04 6.050306e-04 0.3271273 0.74697069

> #Compare to lm command
> lm(repvshr ~ income)

Call:
lm(formula = repvshr ~ income)

Coefficients:
(Intercept)       income
  4.586e+01    1.979e-04

>
```

Functions

- Implementing our new function `ols`, we get precisely the output that we asked for.

- **We can check this against the results produced by the standard `lm` function.**

R Console

```
+      b.standard.errors<-sqrt(diag(xxi))*sqrt(s2)   #coeffici$
+      b.t.statistic<-b/b.standard.errors             #t statis$
+      b.t.prob<-2*(1-pt(b.t.statistic,df.e))         #alpha 0.$
+
+
+      b.table<-cbind(est=b,b.se=b.standard.errors,t.stat=b.t$
+
+
+   return(list(b=b.table
+               ))
+ }
> #Output of new ols command
> ols(repvshr, income)
$b
             est         b.se    t.stat          p
int 4.585949e+01 2.382350e+01 1.9249684 0.06857514
x   1.979220e-04 6.050306e-04 0.3271273 0.74697069

> #Compare to lm command
> lm(repvshr ~ income)

Call:
lm(formula = repvshr ~ income)

Coefficients:
(Intercept)       income
  4.586e+01    1.979e-04

>
```

Our Favorite 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

You're Now **R**eady to Go!

- PRISM fellows are available for help
- Contact us with your questions:
  - Dino Christenson, Senior Methods Fellow
    christenson.24@osu.edu
    Derby 2049Q; Phone: (614) 292-9661
    Office Hours: Mon - Thurs: 9:00am-11:30am
    & by appointment

  - Scott Powell, Junior Methods Fellow
    powell.413@polisci.osu.edu
    Derby 2049Q; Phone: (614) 292-9661
    Office Hours: Tues & Thurs: 9:30-11:30am 3:30-5:00pm
    Wed: 8:30-11:30am & by appointment

Upcoming PRISM Brownbags

- Please join us for our next brownbag
  - *An Introduction to STATA*
  - January 25, 2008
    10:30am-12:00pm
    Derby Hall 125
    - Should be very helpful if you are taking 686 next quarter!
- Spring quarter brownbag
  - *Bayesian Inference with WinBUGS*
    Date & Time TBA (Spring 2008)
- Additional information available on the PRISM website
  - http://polisci.osu.edu/prism/luncheons.htm