JOHNS HOPKINS
BLOOMBERG
SCHOOL of PUBLIC HEALTH

A Brief Introduction to R

Causal Inference in Medicine and Public Health (140.664)

Department of Biostatistics

What is R?

- “A language and environment for statistical computing and graphics”
- Comprehensive R Archive Network (CRAN): http://www.r-project.org/
- Latest version is v3.1.2 (“Pumpkin Helmet”)

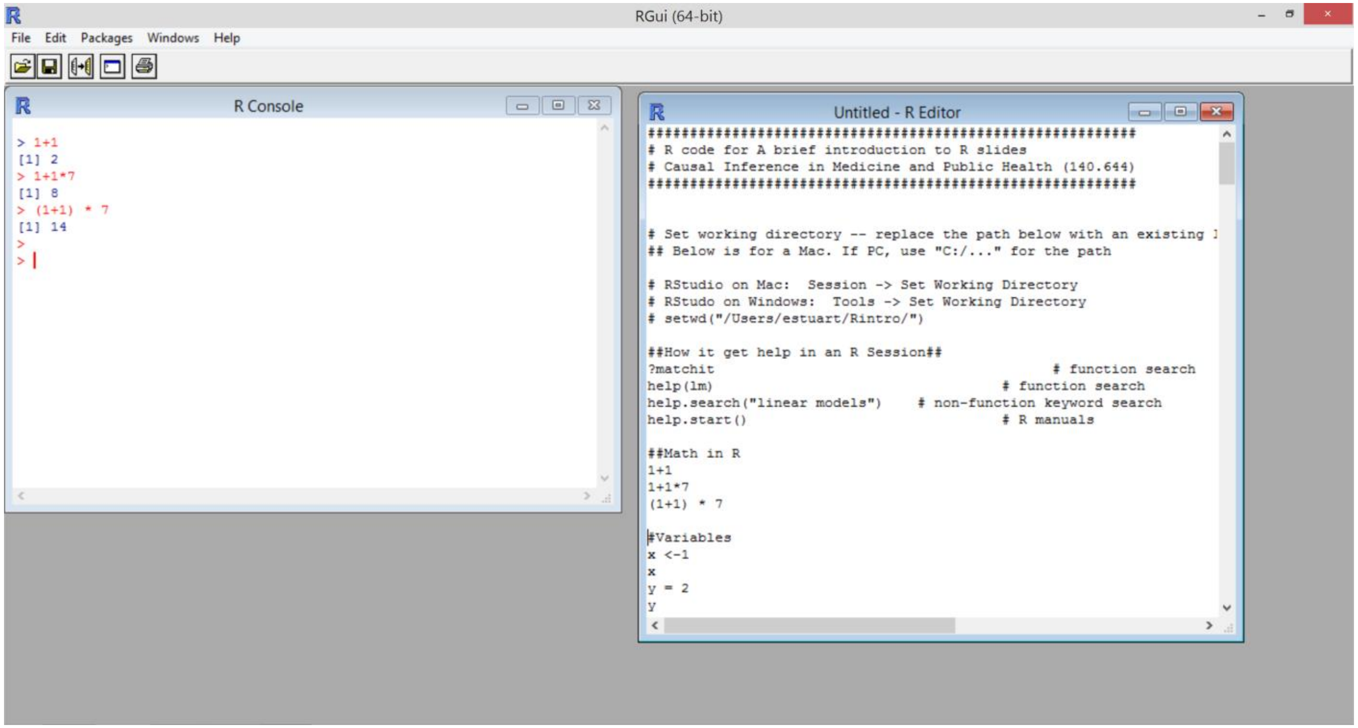
Installation

- R can be downloaded from

  http://cran.r-project.org

Getting Started

- Workspace ~ current R working environment and includes all user-defined objects
- Console ~ type your commands and see the text results
  - The > symbol is the prompt to type commands
- Source files ~ a sequence of commands that can be run/re-run anytime
  - Use the native script editor to create source files



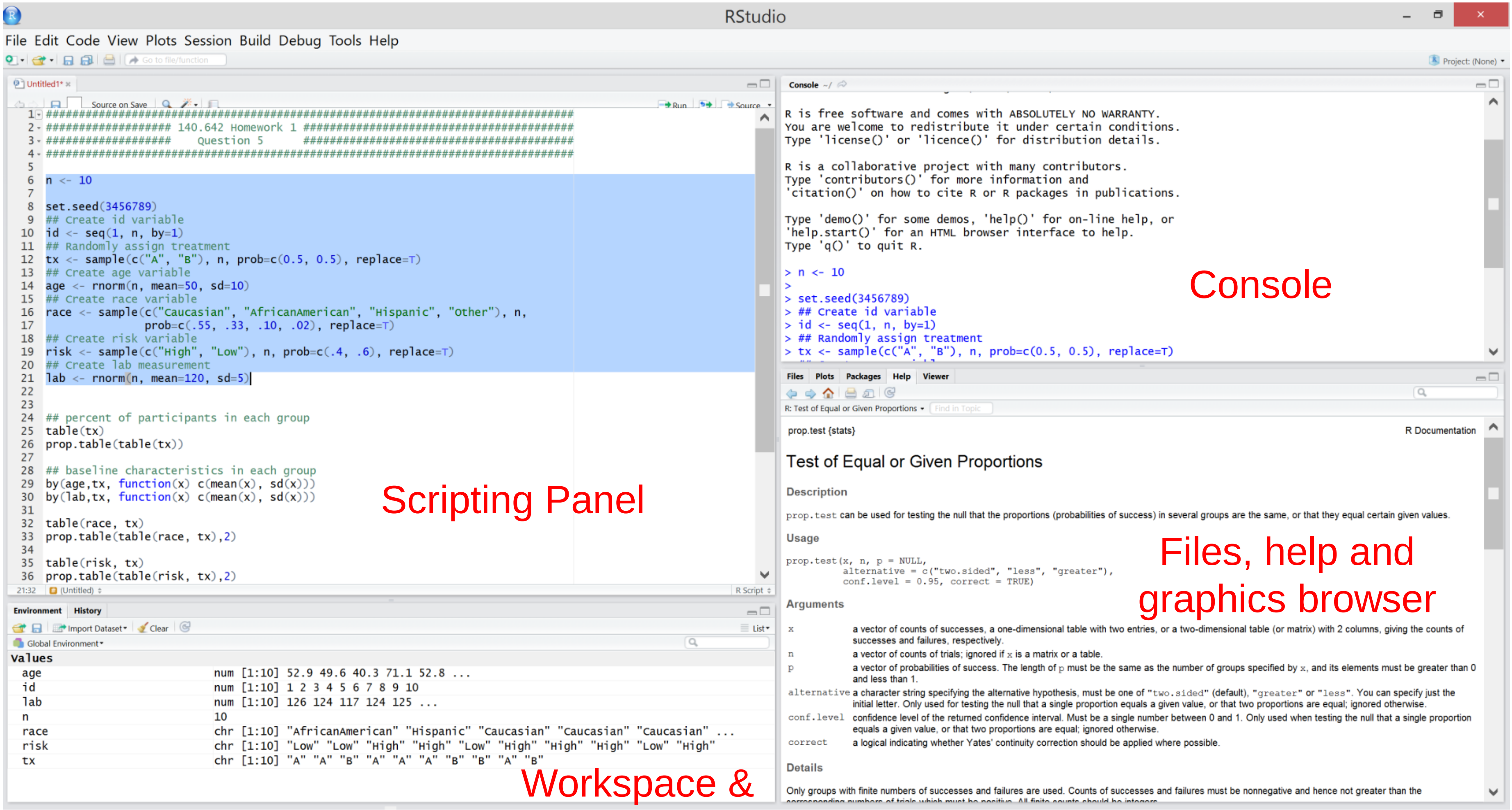
```
> 1+1
[1] 2
> 1+1*7
[1] 8
> (1+1) * 7
[1] 14
>
> |
```

```
##########################################################
# R code for A brief introduction to R slides
# Causal Inference in Medicine and Public Health (140.644)
##########################################################


# Set working directory -- replace the path below with an existing 1
## Below is for a Mac. If PC, use "C:/..." for the path

# RStudio on Mac:  Session -> Set Working Directory
# RStudo on Windows:  Tools -> Set Working Directory
# setwd("/Users/estuart/Rintro/")

##How it get help in an R Session##
?matchit                                        # function search
help(lm)                                  # function search
help.search("linear models")    # non-function keyword search
help.start()                              # R manuals

##Math in R
1+1
1+1*7
(1+1) * 7

#Variables
x <-1
x
y = 2
y
```

RStudio

- An Integrated Development Environment (IRE) for R
  - R is the programming language
  - RStudio is a convenient interface
- After installing R you also need to install RStudio
  - http://rstudio.org

RStudio



```
####################################################################################
################## 140.642 Homework 1 ##############################################
##################    Question 5      ##############################################
####################################################################################

n <- 10

set.seed(3456789)
## Create id variable
id <- seq(1, n, by=1)
## Randomly assign treatment
tx <- sample(c("A", "B"), n, prob=c(0.5, 0.5), replace=T)
## Create age variable
age <- rnorm(n, mean=50, sd=10)
## Create race variable
race <- sample(c("Caucasian", "AfricanAmerican", "Hispanic", "Other"), n,
               prob=c(.55, .33, .10, .02), replace=T)
## Create risk variable
risk <- sample(c("High", "Low"), n, prob=c(.4, .6), replace=T)
## Create lab measurement
lab <- rnorm(n, mean=120, sd=5)


## percent of participants in each group
table(tx)
prop.table(table(tx))

## baseline characteristics in each group
by(age,tx, function(x) c(mean(x), sd(x)))
by(lab,tx, function(x) c(mean(x), sd(x)))

table(race, tx)
prop.table(table(race, tx),2)

table(risk, tx)
prop.table(table(risk, tx),2)
```

```
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.

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.

> n <- 10
>
> set.seed(3456789)
> ## Create id variable
> id <- seq(1, n, by=1)
> ## Randomly assign treatment
> tx <- sample(c("A", "B"), n, prob=c(0.5, 0.5), replace=T)
```

prop.test {stats} R Documentation

Test of Equal or Given Proportions

Description

prop.test can be used for testing the null that the proportions (probabilities of success) in several groups are the same, or that they equal certain given values.

Usage

```
prop.test(x, n, p = NULL,
          alternative = c("two.sided", "less", "greater"),
          conf.level = 0.95, correct = TRUE)
```

Arguments

x	a vector of counts of successes, a one-dimensional table with two entries, or a two-dimensional table (or matrix) with 2 columns, giving the counts of successes and failures, respectively.
n	a vector of counts of trials; ignored if x is a matrix or a table.
p	a vector of probabilities of success. The length of p must be the same as the number of groups specified by x, and its elements must be greater than 0 and less than 1.
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. Only used for testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
conf.level	confidence level of the returned confidence interval. Must be a single number between 0 and 1. Only used when testing the null that a single proportion equals a given value, or that two proportions are equal; ignored otherwise.
correct	a logical indicating whether Yates' continuity correction should be applied where possible.

Details

Only groups with finite numbers of successes and failures are used. Counts of successes and failures must be nonnegative and hence not greater than the

```
Values
  age        num [1:10] 52.9 49.6 40.3 71.1 52.8 ...
  id         num [1:10] 1 2 3 4 5 6 7 8 9 10
  lab        num [1:10] 126 124 117 124 125 ...
  n          10
  race       chr [1:10] "AfricanAmerican" "Hispanic" "Caucasian" "Caucasian" "Caucasian" ...
  risk       chr [1:10] "Low" "Low" "High" "High" "Low" "High" "High" "High" "Low" "High"
  tx         chr [1:10] "A" "A" "B" "A" "A" "A" "B" "B" "A" "B"
```

Console

Scripting Panel

Files, help and graphics browser

Workspace & History

RStudio

- To customize Rstudio layout:
  - Tools >> Global Options
  - Code Editing: Allows you to select the number of spaces for tab (Rstudio will auto indent when writing loops, functions, etc.)
  - Appearance: Can change the background for stealth night coding!
  - Pane layout: Allows you to move around the console, script, workspace/history, etc.

Notes about RStudio

- The R *command prompt* looks the same
- Saves R sessions correctly
- The Workspace Panel displays variables, functions, data frames, and other objects in the current workspace
- The Scripting Panel acts as a high-level text editor

Let’s get started

Open up R…

(or RStudio)

R syntax

- The R interface is often called a “session”

- How to input the syntax:
  - Type directly into the session
  - Create “documents”, select and run the commands using Ctrl+R (Windows) or Cmd+Enter (Mac)
  - In Rstudio, use the Script Editor

- Can use the arrow keys on keyboard to scroll through previous commands in session

R syntax

- ***R is case-sensitive***
- Command lines do not need to end with a character (like a semicolon in SAS)
- Anything following the pound character (#) on one line is commented out

“Base” version and add-ons

- What you initially download from CRAN is the “base” version
  - A core set of R features that does most basic functions
  - The “base” version is a fully functional statistical environment
- Add-on packages
  - perform many additional statistical and graphical procedures
  - can download through R itself after it is installed
  - 1000s available from the CRAN repository, even more in other web repositories

Add-on packages

- To use an add-on package, you must first install it (once), by one of two ways:

  1a. Install in R by navigating to the `Packages and Data | Package Installer` toolbar

  1b. Choose a close CRAN mirror (e.g., USA (MD))

  1c. Select a package (here, select “MatchIt”) from the menu and click on “Install Selected”

  2. Or type: `install.packages(“MatchIt”)`

- ***During each session you want to use a package, you must load it in by typing***:

```
> library(MatchIt)
> library(foreign)
```

Add-on packages

- **Foreign**
  - Reads data stored created by Minitab, S, SAS, SPSS, Stata, Systat, dBase
- **MatchIt**
  - Implements non-parametric matching methods
  - Also contains the dataset (“lalonde”) we will use in this tutorial

Working directory

- The best way to work in R is by setting a working directory for different data projects
- Set working directory as an (existing) folder on our computer using the `setwd` function:

In Windows --

```
> setwd("C:/Files/RIntro/")
```

On a Mac --

```
> setwd("/Users/estuart/Rintro/")
```

In R Studio --

Use the drop down menu from the Sessions Tab

Sample program for today

- Open the R program "R_tutorial_slides_2014_code.R" from CoursePlus to follow along
    - Navigate to

      `File | Open Document` (Mac)

      `File | Open Script` (Windows)
    - Find where you saved the file `"R_tutorial_slides_2014.R"`

Working in R

- R is an *object-oriented* programming language
- Objects have data elements and are member of classes.
- *Functions* operate on objects
  - Functions are invoked by their name followed by the parenthesis
  - `>summary(x)`
  - The class of an object determines how the function operates

Objects

- R has 5 basic classes of objects:
  - numeric (real numbers)
  - Integer
  - character
  - complex
  - logical (True/False)

- “Things” are assigned to and stored in objects using the **<-** or **=** operator.

Object Classes - Numeric

- Decimal vales are called numerics
  - Numerics are the default computational data type

```
> x = 10.5     # assign a decimal value
> x            # print the value of x
[1] 10.5
> class(x)     # print the class name of x
[1] "numeric"
```

Object Classes - Numeric

- Numbers in R a generally treated as numeric objects (i.e. double precision real numbers)
- If you explicitly want an integer, you need to specify the L suffix

Object Classes - Integers

- Whole numbers (stored without double precision)
  - Integer objects exist so that data can be passed to C or Fortran code that expects them

```
> number = 1L
> class(number)
[1] "integer"
> number
[1] 1
```

Vectors

- The most basic object is a vector
  - an ordered collection of data of the same type
  - A vector can only contain objects of the same class

Defining vectors

- The ***c() function*** can be used to create vectors of objects.

```
> a <- c(0.5, 0.6)                  ## numeric
> b <- c(TRUE, FALSE)               ## logical
> c <- c(T, F)                      ## logical
> d <- c("a", "b", "c")             ## character
> e <- 9:29                         ## integer
> f <- c(1+0i, 2+4i)                ## complex
```

- Using the vector() function

```
> g <- vector("numeric", length = 10)
> g
[1] 0 0 0 0 0 0 0 0 0 0
```

Defining matrices

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)

Define 2 vectors, x and y

```
> x <- c(10,20,30,40)
> y <- c(50,60,70,80)
```

Create 2 matrices from these vectors

Matrices can be created by column-binding or row binding with cbind() and rbind().

```
> matrix1 <- cbind(x,y)
> matrix2 <- rbind(x,y)
> matrix1
```

Factors

- Factors are used to represent categorical data.
- Factors can be *unordered* or *ordered*
- Think of a factors as an integer vector where each integer has a label.
- Factors can be used in statistical modeling where they will be implemented correctly,
  - they will then be assigned the correct number of degrees of freedom.
- Using factors with labels is better than using integers because factors are
  - self-describing; having a variable that has values “Male" and “Female" is better than a variable that has values 1 and 2.

Factors

```
gender <- c("male", "male", "male", "male", "male", "male", "male",
"male", "male", "male", "male", "male", "male”, "female”, "female”,
"male", "male",  "male", "male", "female")

is.factor(gender)
gender.f <- factor(gender, levels = c(“male”, “female”))
```

- When the variable is ordinal, we need to use *ordered factors*
- This is important in linear modeling because the 1st level is the baseline level (i.e. the reference group).
- There are two options to create ordered factors:
  - Use the factor() function with the argument ordered=TRUE.
  - Use the ordered() function.

```
g.order <- ordered(gender, levels = c("male", "female"))
```

Data Frames

- ***Data frames*** are typical data tables that researchers come up with – like a spreadsheet.
- It is a rectangular table with rows and columns; data within each column has the same type (e.g. number, text, logical), but different columns may have different types.
- Example:

```
> A
      localisation tumorsize progress
XX348     proximal       6.3    FALSE
XX234       distal       8.0     TRUE
XX987     proximal      10.0    FALSE
```

- Or with `View(A)` which will display the data frame in a spreadsheet-like view in the script window

Reading in a Data File

- Function: `read.csv()`

  ```
  read.csv(file, header = TRUE,  sep = ",“)
  ```

  - `sep = “,”` tells R that data delimited by comma
  - `header = T` tells R that first line of data contains variable names
- Can specify optional arguments in any order:

  ```
  > data <- read.csv(“lalonde.txt”,
    sep=“;”, header = T)
  ```

  or

  ```
  > data <- read.csv(“lalonde.txt”, header
    = TRUE, sep = “;”)
  ```

Importing Data from other Programs

```
library (foreign)
```

From SPSS

- `spss.data <- read.spss("C:/temp/spssfile.sav")`

From Stata

- `statadata <- read.dta("C:/temp/statafile.dta")`

From SAS

- `sasdata <- read.xport("C:/temp/sasfile.xpt")`

**R can now (as of 4/2014) import SAS permanent data sets (.sas7bdat) files – use `read.sas7bdat` in the `sas7bdat` package

Sample dataset: “lalonde”

- “lalonde” data included in the `MatchIt` package – use “`data`” function to load in:

```
> data(lalonde)
```

- Subset of data from an analysis conducted in the mid-1970s comparing men participating in a national federally-funded job training program to similar men not in the program
  - N = 614 in dataset (185 “treated” individuals, 429 controls)
  - 10 variables

Exploring the data: Summaries

- To get a list of the variable names in the dataset:

```
> names(lalonde)
```

- To get a summary of each of the variables in the dataset called “lalonde”:

```
> summary(lalonde)
```

Variables and matrices

- Matrices and data frames can be referenced by variable names (columns) and by row numbers

- Refer to an individual variable with `$VARIABLENAME`

```
> lalonde$married
```

Subsetting

- Square brackets `[n,m]` are used to subset a dataset
  - Elements to the left of the comma subset **rows** (observations)
  - Elements to the right of the comma subset **columns** (variables)

Ex) Print out data for first 6 individuals (rows)

```
> lalonde[1:6,]
```

Alternatively: `> head(lalonde)`

Ex) Extract columns 2-5

```
> lalonde.4vars <- lalonde[,2:5]
```

Ex) Matrix of all variables for individuals with `nodegree` = 1

```
> lalonde.nodegree <-lalonde[lalonde$nodegree==1,]
```

Missing Data

- Missing values are represented by the symbol **NA** (not available)
- Impossible values (e.g., dividing by zero) are represented by the symbol **NaN** (not a number).

Missing Data

- **Recoding Values to NA**
  - Recode 99 to missing for variable v1
  - Select rows where v1 is 99 and recode column v1
  - `mydata$nomissing[mydata$nomissing==99 ] = NA`

- **Exclude Missing Values**
  - `x <- c(1,2,NA,3)`
  - `Mean(x)`
  - `mean(x, na.rm=TRUE)`

- **Omit Missing Values**
  - `newdata <- na.omit(mydata)`

Logical statements

Cmd	Meaning		Cmd	Meaning
<	Is less than		==	Is equal to
<=	Is less than or equal to		!=	Is not equal to
>	Is greater than		&	And
>=	Is greater than or equal to		|	Or

- Use logical statements to evaluate conditions
  - E.g., Which observations have more than 5 years and an `re74` value below $5,000?

```
> educ>5 & re74<5000
> table(educ>5 & re74<5000)
```

Recoding

- Creating a dummy variable:
  Ex) Want indicator variable to identify those with low income (`re74 < 5000`) and greater than 5 yrs education

educre74 is equal to: 1 if **educ** > 5 AND **re74** < 5000
0 otherwise (i.e., **educ** <=5 OR **re74** >=5000)

```
> lalonde$v2 <- as.numeric(educ > 5 & re74 < 5000)
```

or

```
> lalonde$v3 <- ifelse(educ > 5 & re74 < 5000,1,0)
```

- Check your work!

```
> table(lalonde$v2)
```

Descriptive statistics

Average: function `mean()`

- If missing data, specify an additional option, `na.rm=TRUE`, to get the mean after removing missing values (`NA`)

```
> mean(age, na.rm=TRUE)
```

Frequency Tables:

- Built-in function: `table(educ)`
- Alternatively,

```
install.packages(“gmodels”)
library(gmodels)
```

1-way table: `CrossTable(educ)`

2-way table: `CrossTable(educ, treat)`

Basic test statistics

- `Chi-Squared Test`

```
> chisq.test(lalonde$treat,lalonde$married)
```

OR

```
> CrossTable(lalonde$treat,lalonde$married,chisq =
     T)
```

- `t.test(vector1,vector2)`

```
> t.test(lalonde$educ[treat==1],
    lalonde$educ[treat==0])
```

- `wilcox.test(vector1,vector2)`

```
> wilcox.test(lalonde$educ[treat==1],
lalonde$educ[treat==0])
```

Remember: Type `?commandname` or `help(commandname)` for more details on any of these commands!

Basic models

Generalized linear models

- A wide range of generalized linear models can be generated using the `glm` command, by specifying your formula, the family, and the link function:

  Example: **logistic regression**
  (binomial family and logistic link)

  ```
  > married.model <- glm(married~educ+age,
  family=binomial(link=logit), data=lalonde)
  ```

- The regression outputs can be extracted from the object created by the `glm` command:

  ```
  > summary(married.model)
  > married.model$coefficients
  > exp(married.model$coefficients)
  ```

Objects and clearing memory

- R keeps all objects you've created in the current session in memory

- To reveal objects currently in memory:

```
> objects()
```

or

```
> ls()
```

- To remove an object, use the `rm` function
  - To remove an object called "a", use:

    ```
    > rm(a)
    ```

  - To remove all objects, use:

    ```
    > rm(list = ls())
    ```

Comparisons between R and Stata

Feature	R	Stata
Changing/specifying a working directory	`setwd("C:/folder")`	`cd "c:\..."`
Writing a program	creating an R document, or using `source()` to read in a text file	creating a ".do" file
Saving your work	`sink()` output	log files
Comments in a program	#	*
Name of the data	an object	data

To get additional packages	`install.packages("xxx")`
	`library(xxx)`
		`findit xxx`
		(.ado files)
		`ssc install xxx`

Missing values	`NA` or `NaN`	.

Where to get help

- From the prompt, you can type `?` followed by the name of any function to return the documentation for that function:

  ```
  > ?lm
  ```

- The command `help` does the same thing:

  ```
  > help(lm)
  ```

- For non-command key words, type the string in quotation marks after the command `help.search`:

  ```
  > help.search(“linear models”)
  ```

- You can also browse the html manuals with:

  ```
  > help.start()
  ```

Where to get more help

- A Google search is often helpful!
- Even better: an www.rseek.com search
- The R website has a lot of good information
  - www.r-project.org
  - cran.r-project.org/doc/contrib/Short-refcard.pdf (short reference card of key commands)
  - www.r-project.org/search.html (to search R forums and mailing lists)
- Quick-R for SAS/SPSS/Stata users:
  - www.stat-methods.net

Where to get even more help

- JHSPH resources
  - Brian Caffo's website: www.biostat.jhsph.edu/~bcaffo/651/resources.html
  - Andrew Jaffe's R seminar in the Epi department: www.biostat.jhsph.edu/~ajaffe/rseminar.html
- UCLA's Stat Computing website
  - www.ats.ucla.edu/stat/r/
  - www.ats.ucla.edu/stat/r/sk ("starter kit")
- Tutorial document lists other resources

Where to get even more help

- Alyssa Frazee’s 1-hour R introduction:

http://alyssafrazee.com/introducing-R.html

- And her references:

https://www.codeschool.com/courses/try-r

http://www.cookbook-r.com/

https://www.datacamp.com/courses/introduction-to-r