Stat 107: Introduction to Business and Financial Statistics
Class 3: Computing Discussion

Introduction to R

HOME PAGE | TODAY'S PAPER | VIDEO | MOST POPULAR | TIMES TOPICS

The New York Times
January 6, 2009

Business Computing

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS | OPINION

Search Technology Go

Inside Technology
Internet | Start-Ups | Business Computing | Companie

Data Analysts Captivated by R's Power

Stuart Isett for The New York Times

R first appeared in 1996, when the statistics professors Robert Gentleman, left, and Ross Ihaka released the code as a free

Why R?

- Free (open-source)
- Programming language (not point-and-click)
- Excellent graphics
- Offers broadest range of statistical tools
- Easy to generate reproducible reports
- Easy to integrate with other tools
- What the cool kids are using
- Used in 110,111,139,121, etc……

R is a tool for...

Data Manipulation

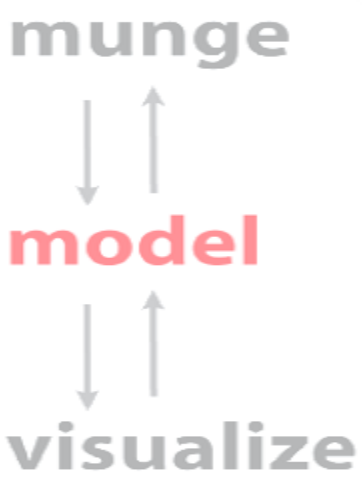
- connecting to data sources
- slicing & dicing data

Modeling & Computation

- statistical modeling
- numerical simulation

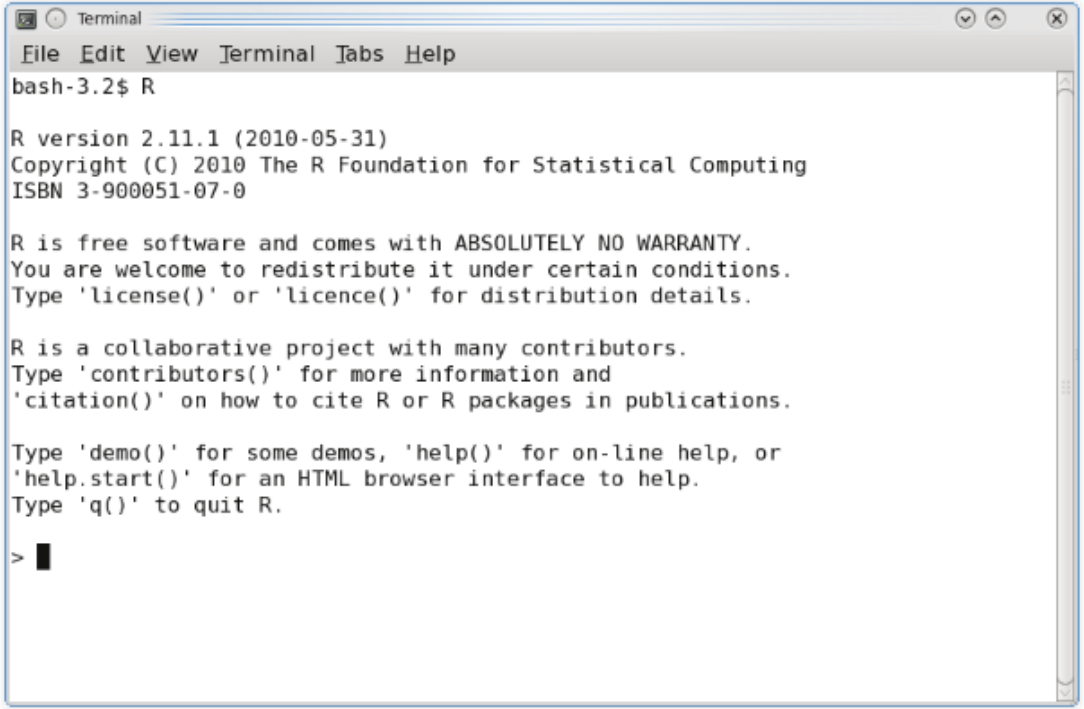
Data Visualization

- visualizing fit of models
- composing statistical graphics



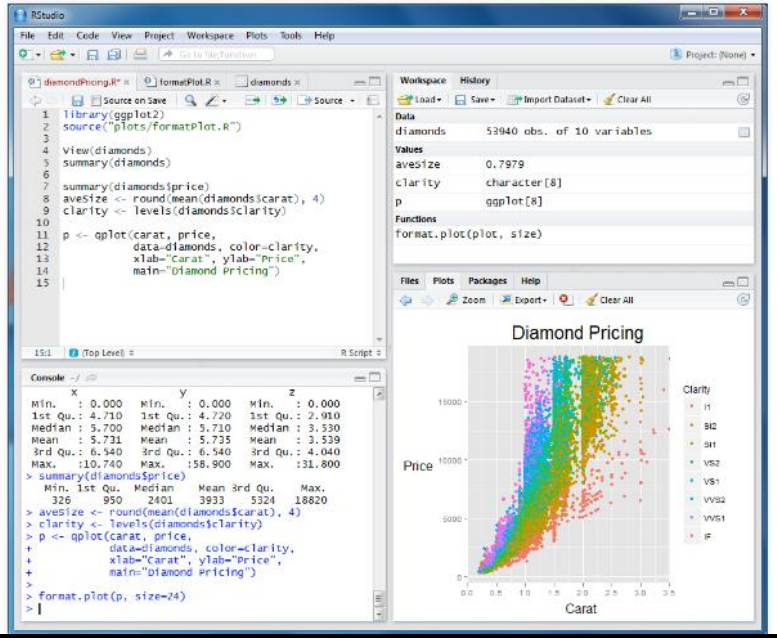
The R Console

- Basic interaction with R is through typing in the **console**
- This is the **terminal** or **command-line** interface



Rstudio is an IDE for R

- RStudio has 4 main windows ('panes'):
  - Source
  - Console
  - Workspace/History
  - Files/Plots/Packages/Help



History of R

- S: language for data analysis developed at Bell Labs circa 1976
- Licensed by *AT&T/Lucent* to *Insightful Corp*. Product name: *S-plus*.
- R: initially written & released as an open source software by Ross Ihaka and Robert Gentleman at U Auckland during 90s (R plays on name “S”)
- Since 1997: international R-core team ~15 people & 1000s of code writers and statisticians happy to share their libraries! AWESOME!

“Open source”... that just means I don’t have to pay for it, right?

- No. Much more:
  - –Provides full access to algorithms and their implementation
  - –Gives you the ability to fix bugs and extend software
  - –Provides a forum allowing researchers to explore and expand the methods used to analyze data
  - –Is the product of 1000s of leading experts in the fields they know best. It is CUTTING EDGE.
  - –Ensures that scientists around the world - and not just ones in rich countries - are the co-owners to the software tools needed to carry out research
  - –Promotes reproducible research by providing open and accessible tools
  - –Most of R is written in… R! This makes it quite easy to see what functions are actually doing.

There are over 800 add-on packages

- This is an enormous advantage - new techniques available without delay, and they can be performed using the R language you already know.
- Allows you to build a customized statistical program suited to your own needs.
- Downside = as the number of packages grows, it is becoming difficult to choose the best package for your needs, & QC is an issue.

(http://cran.r-project.org/src/contrib/PACKAGES.html)

http://cran.r-project.org/web/views/Finance.html

Finance

- The Rmetrics suite of packages comprises fArma, fAsianOption
  fNonlinear, fOptions, fPortfolio, fRegression, timeSeries (former
  and computational finance.
- The RQuantLib package provides several option-pricing functio
- The quantmod package offers a number of functions for quantita
- The portfolio package contains classes for equity portfolio mana
  market prices. The backtest offers tools to explore portfolio-ba
  multigroup models.
- The PerformanceAnalytics package contains a large number of
- The TTR contains functions to construct technical trading rules i
  analyse and use such trading rules.
- The financial package can compute present values, cash flows a
- The sde package provides simulation and inference functionality
- The termstrc and YieldCurve packages contain methods for the
  (1994) extension. The former package adds the McCulloch (19
- The vrtest package contains a number of variance ratio tests for
- The BLCOP package provides implementation of the Black-Lit
- The gmm package provides generalized method of moments (G
- The tawny package contains estimator based on random matrix
- The SV package uses indirect inference to estimate non-Gaussi
- The orderbook package can be used to analyses market micros
- The schwartz97 package can be used to model the Schwartz (1
- The rrv package provides functions for modelling portfolio retur
  weight; modelling returns with empirical cumulative distribution f

Risk management

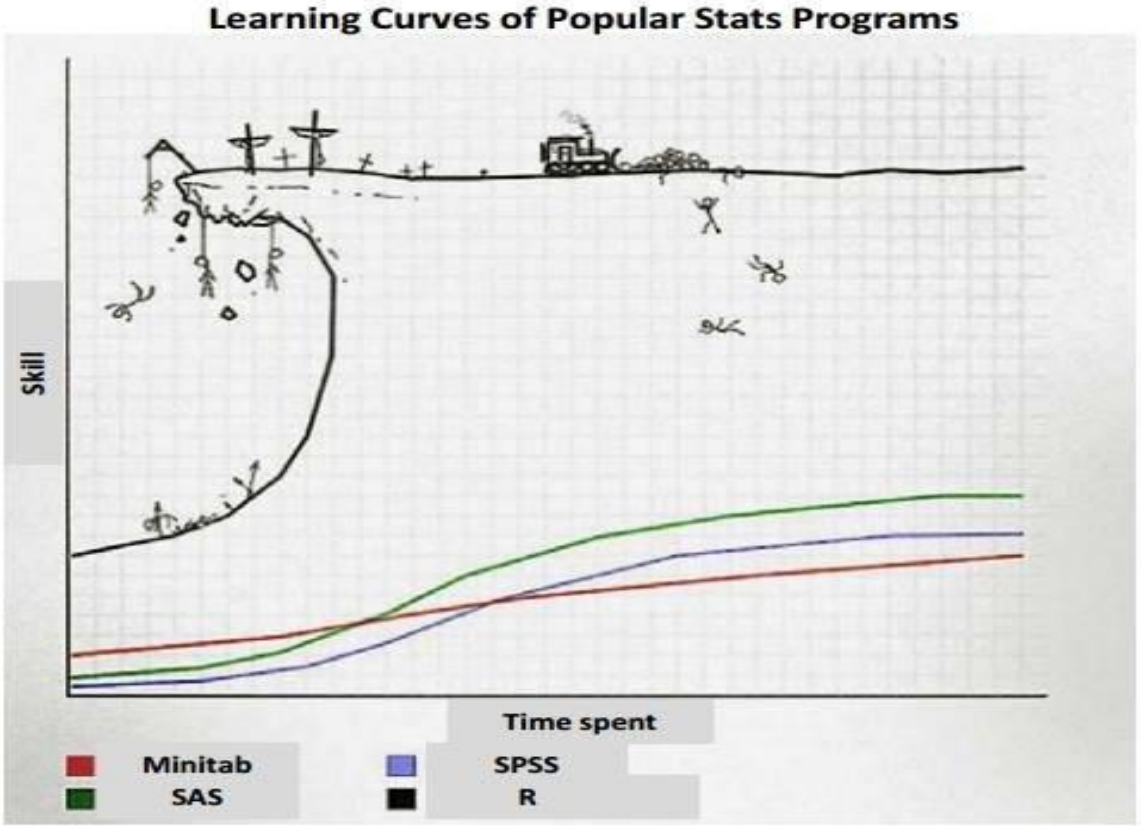
Again- Why R?

- It's free!
- It runs on a variety of platforms including Windows, Unix and MacOS.
- It provides an unparalleled platform for programming new statistical methods in an easy and straightforward manner.
- It contains advanced statistical routines not yet available in other packages.
- It has state-of-the-art graphics capabilities.

Warning: R has a (steep) learning curve

- First, while there are many introductory tutorials (covering data types, basic commands, the interface), none alone are comprehensive.
- In part, this is because much of the advanced functionality of R comes from hundreds of user contributed packages. Hunting for what you want can be time consuming, and it can be hard to get a clear overview of what procedures are available.

Steep Learning Curve



Another warning

- I am older than dirt. Well, not really but close.
- I used R before it existed and was Splus.
- So since I am a dinosaur, I use R in a very rudimentary fashion.
- Our hip, young TFs use R all the time for their research and have more modern skills-so go to section to learn more about R.

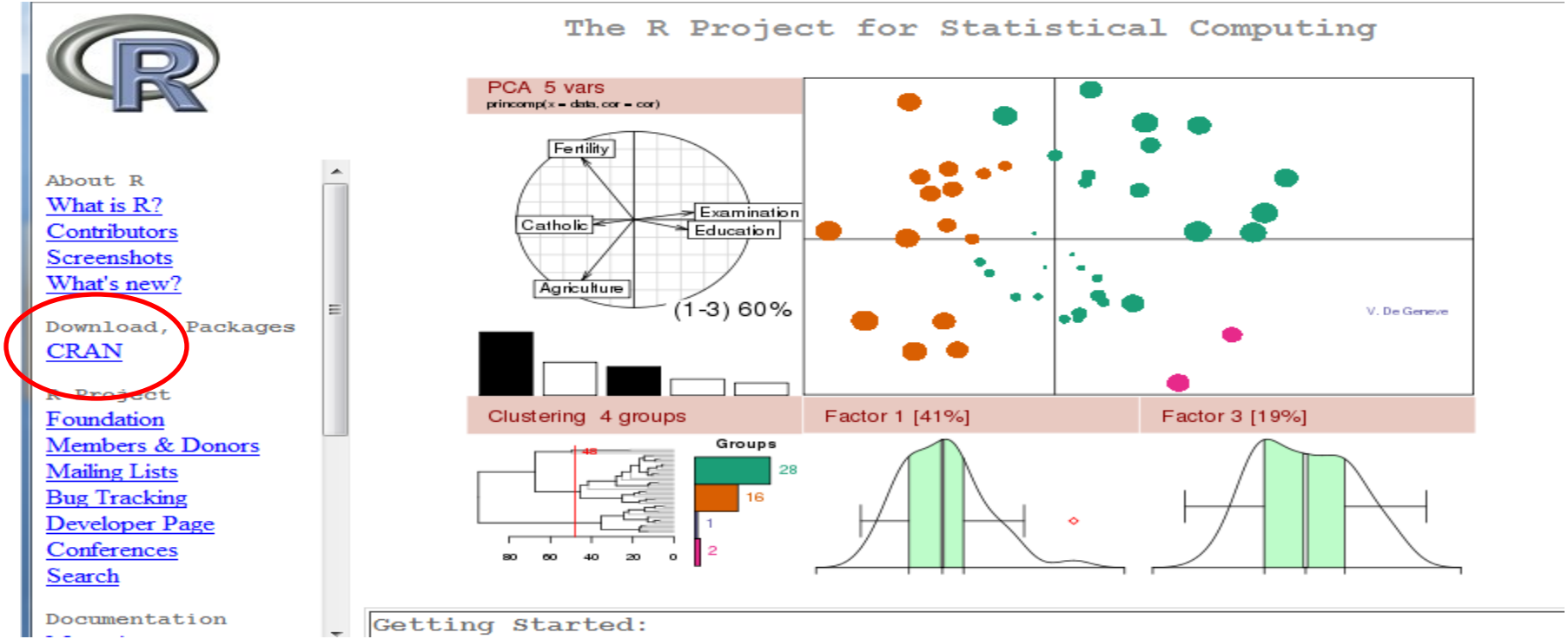
R's Ups and Downs

- Plusses
  - Completely free, just download from Internet
  - Runs on many operating systems
  - Many add-on packages for specialized uses
  - Open source

- Minuses
  - Obscure terms, intimidating manuals, odd symbols, inelegant output (except graphics)

Installing R

- www.r-project.org/
- download from CRAN
- select a download site
- download the base package at a minimum
- download contributed packages as needed
- Then (maybe) download RStudio: http://www.rstudio.com/


The R Project for Statistical Computing
About R
What is R?
Contributors
Screenshots
What's new?
Download, Packages
CRAN
R Project
Foundation
Members & Donors
Mailing Lists
Bug Tracking
Developer Page
Conferences
Search
Documentation
PCA 5 vars
princomp(x = data, cor = cor)
Fertility
Catholic
Examination
Education
Agriculture
(1-3) 60%
V. De Geneve
Clustering 4 groups
Groups
28
16
1
2
Factor 1 [41%]
Factor 3 [19%]
Getting Started:

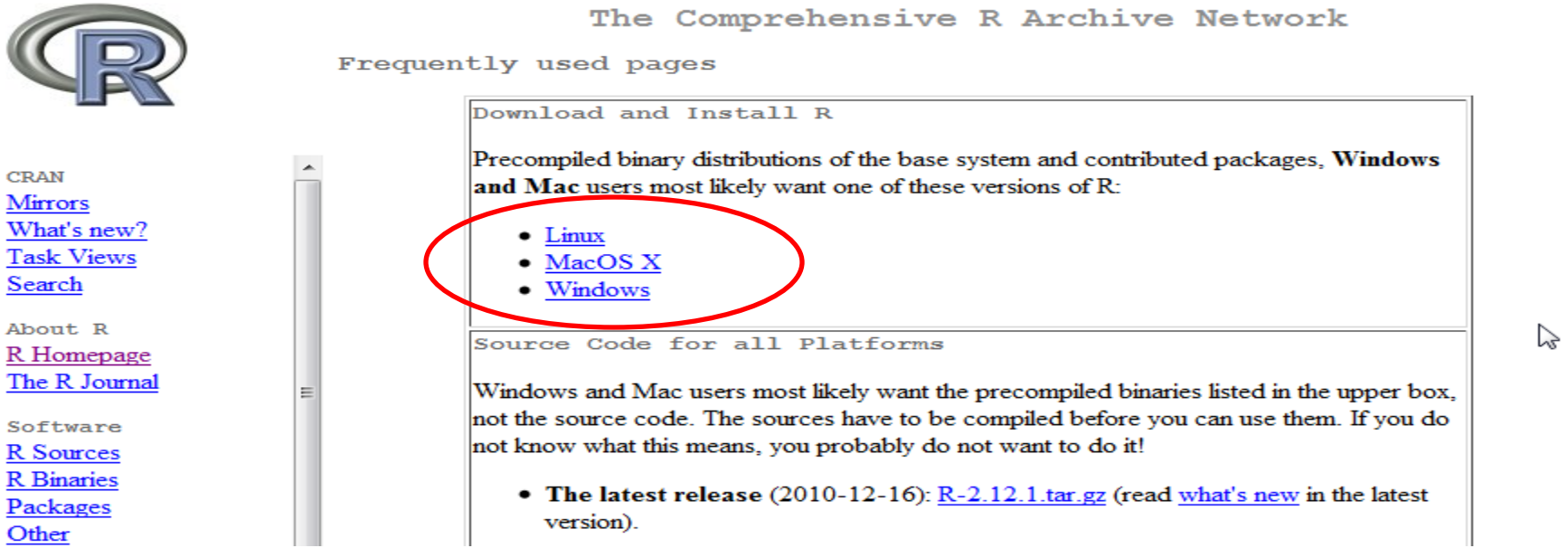
Select a Mirror

- This is were you download from. USA!USA!

USA

http://cran.cnr.Berkeley.edu	University of California, Berkeley, CA
http://cran.stat.ucla.edu/	University of California, Los Angeles, CA
http://streaming.stat.iastate.edu/CRAN/	Iowa State University, Ames, IA
http://ftp.ussg.iu.edu/CRAN/	Indiana University
http://rweb.quant.ku.edu/cran/	University of Kansas, Lawrence, KS
http://watson.nci.nih.gov/cran_mirror/	National Cancer Institute, Bethesda, MD
http://cran.mtu.edu/	Michigan Technological University, Houghton, MI
http://cran.wustl.edu/	Washington University, St. Louis, MO
http://cran.case.edu/	Case Western Reserve University, Cleveland, OH
http://ftp.osuosl.org/pub/cran/	Oregon State University
http://lib.stat.cmu.edu/R/CRAN/	Statlib, Carnegie Mellon University, Pittsburgh, PA
http://cran.mirrors.hoobly.com	Hoobly Classifieds, Pittsburgh, PA
http://mirrors.nics.utk.edu/cran/	National Institute for Computational Sciences, Oak Ridge, TN
http://cran.revolutionanalytics.com	Revolution Analytics, Dallas, TX
http://cran.fhcrc.org/	Fred Hutchinson Cancer Research Center, Seattle, WA
http://cran.cs.wwu.edu/	Western Washington University, Bellingham, WA

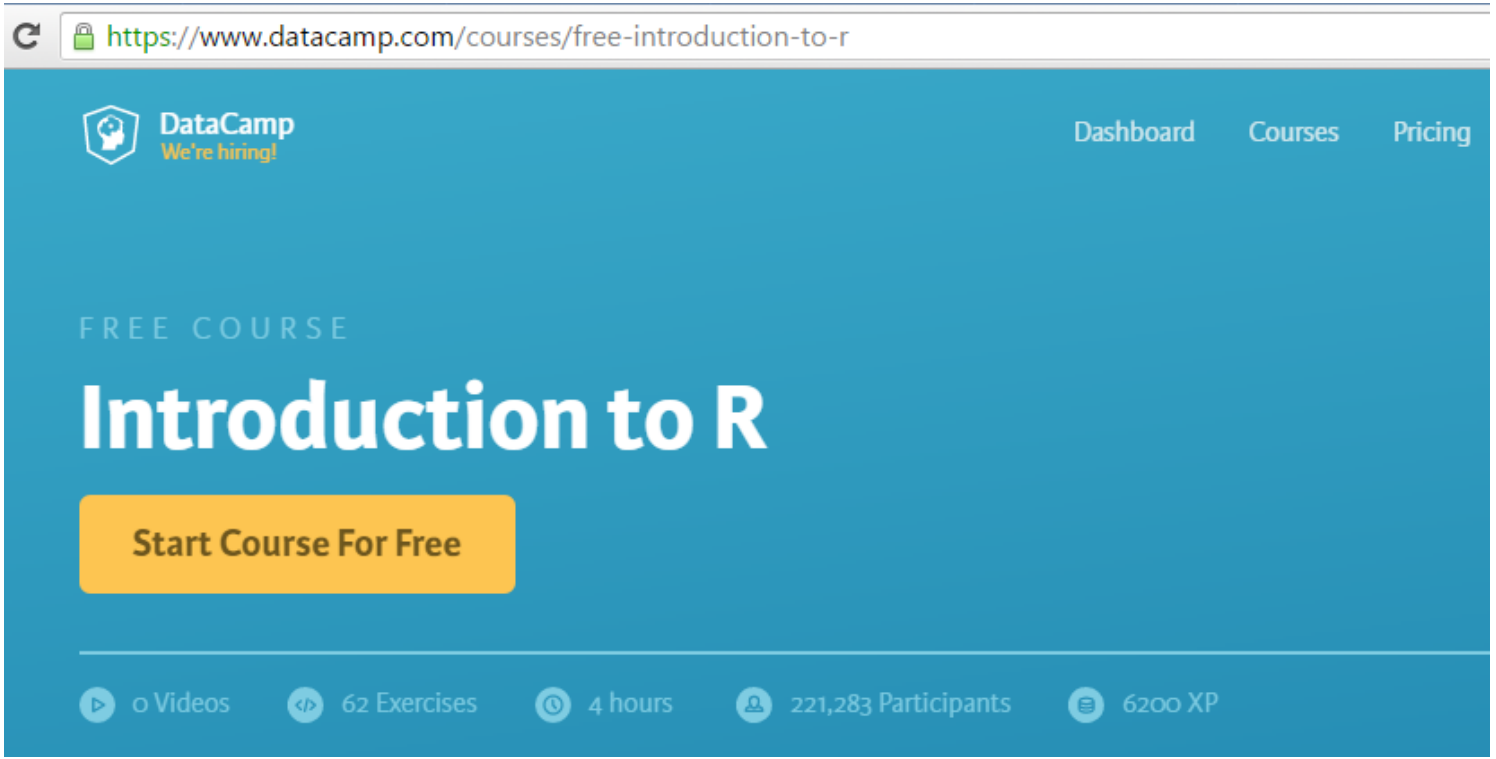
Precompiled Binaries



Tutorials

- From R website under “Documentation”
  - “Manual” is the listing of official R documentation
    - An Introduction to R
    - R Language Definition
    - Writing R Extensions
    - R Data Import/Export
    - R Installation and Administration
    - The R Reference Index
- The Class website has several good introductory tutorials which is really all you need.

Learning R

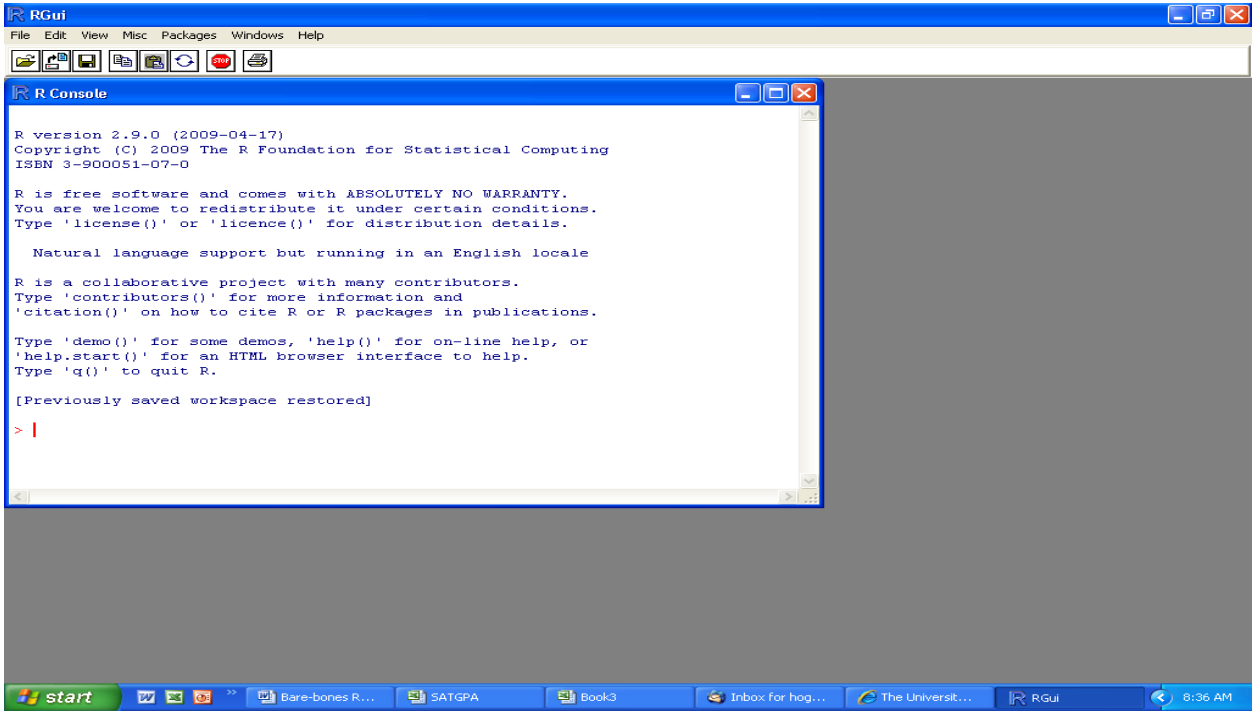

https://www.datacamp.com/courses/free-introduction-to-r
DataCamp
We're hiring!
Dashboard
Courses
Pricing
FREE COURSE
Introduction to R
Start Course For Free
0 Videos
62 Exercises
4 hours
221,283 Participants
6200 XP

Actually Running R

- What will happen when you click on the R icon?

3.2.5 is the version number….you might download a newer version.

The Basic R Screen

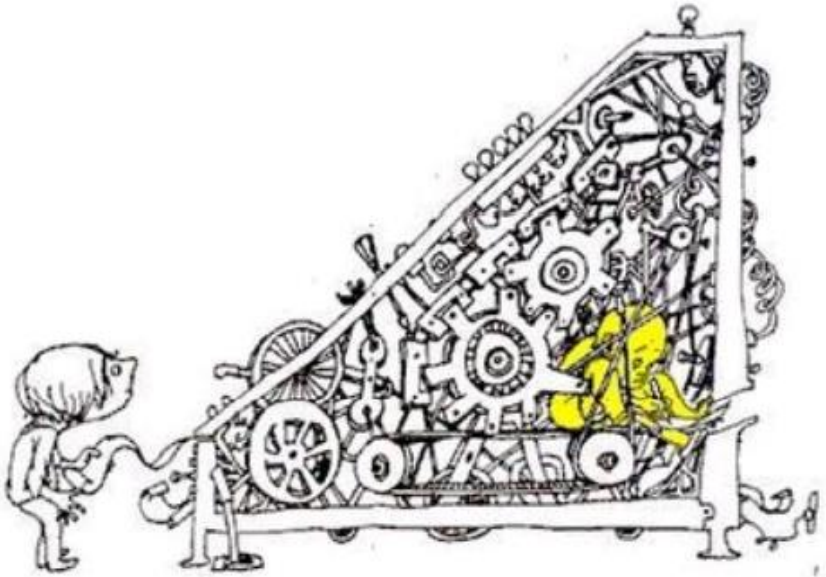


The R prompt (>)

- > This is the “R prompt.”
  - It says R is ready to take your command.

- R is an **interpretative language**, anything you type in it tries to interpret for you.

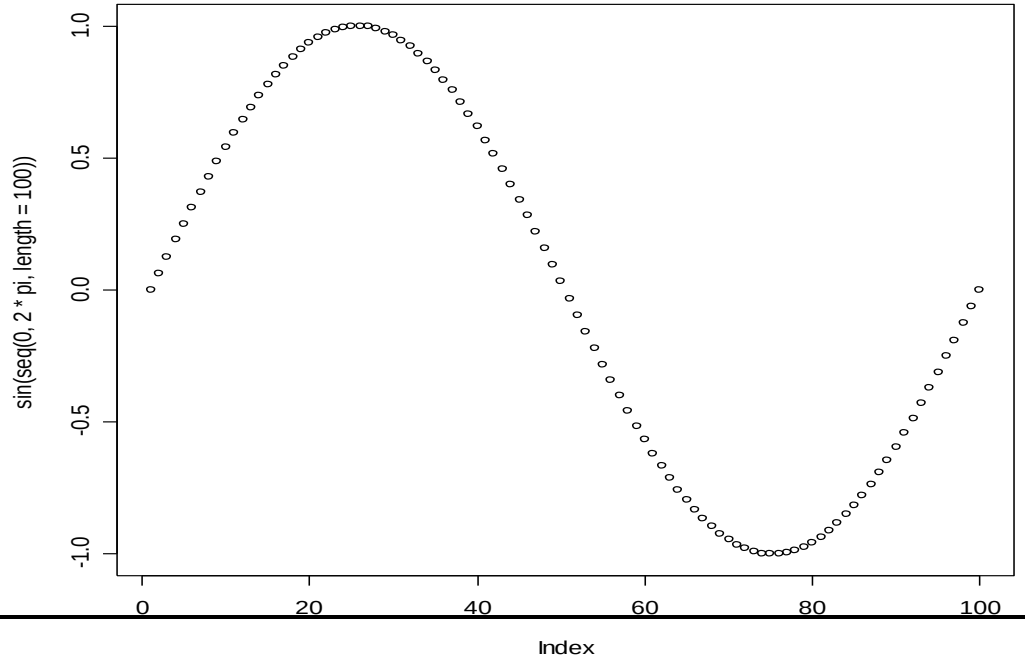
R is an overgrown calculator



```
> ((2+6/3)*5^2)/4
[1] 25

> pi
[1] 3.141593

> log(pi)
[1] 1.14473

> sin(.5)/cos(.3)*sqrt(10)
[1] 1.586956
```

R is a graphing calculator

```
> plot(sin(seq(0, 2*pi, length=100)))
```



R can do vectors

- A **vector** is a sequence of values, all of the same type
- `c()` function returns a vector containing all its arguments in order

```
> students = c("Sean", "Louisa", "Frank", "Farhad", "Li")
> midterm = c(80, 90, 93, 82, 95)
```

- Typing the variable name at the prompt causes it to display

```
> students
[1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

Indexing

- vec[1] is the first element, vec[4] is the 4th element of vec

```
> students
[1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
> students[4]
[1] "Farhad"
```

- vec[-4] is a vector containing all but the fourth element

```
> students[-4]
[1] "Sean"   "Louisa" "Frank"  "Li"
```

Vector Arithmetic

- Operators apply to vectors “pairwise” or “elementwise”:

```
> midterm = c(80, 90, 93, 82, 95) #midterm exam scores
> final = c(78, 84, 95, 82, 91) # Final exam scores

> (midterm+final)/2
[1] 79 87 94 82 93

> course.grades = 0.4*midterm + 0.6*final
> course.grades
[1] 78.8 86.4 94.2 82.0 92.6
```

Another Example Vectorized Math

```
> weight = c(110, 180, 240) ## create vector of weights
> height = c(5.5, 6.1, 6.2) ## create vector of heights
> bmi = (weight*4.88)/height^2 ## divides element wise
> bmi
[1] 17.74545 23.60656 30.46826
>
> x=1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> sum(x)
[1] 55
> prod(x)
[1] 3628800
```

Pairwise Comparisons

- Is the final score higher than the midterm score?

```
> final>midterm
[1] FALSE FALSE  TRUE FALSE FALSE
```

- Boolean operators can be applied elementwise

```
> (final < midterm) & (midterm > 80)
[1] FALSE  TRUE FALSE FALSE  TRUE
```

Functions on Vectors

Command	Description
`sum(vec)`	sums up all the elements of `vec`
`mean(vec)`	mean of `vec`
`median(vec)`	median of `vec`
`min(vec), max(vec)`	the largest or smallest element of `vec`
`sd(vec), var(vec)`	the standard deviation and variance of `vec`
`length(vec)`	the number of elements in `vec`
`pmax(vec1, vec2), pmin(vec1, vec2)`	example: `pmax(quiz1, quiz2)` returns the higher of quiz 1 and quiz 2 for each student
`sort(vec)`	returns the `vec` in sorted order
`order(vec)`	returns the index that sorts the vector `vec`
`unique(vec)`	lists the unique elements of `vec`
`summary(vec)`	gives a five-number summary
`any(vec), all(vec)`	useful on Boolean vectors

Functions on Vectors

```
course.grades
```

```
[1] 78.8 86.4 94.2 82.0 92.6
```

```
mean(course.grades) # mean grade
```

```
[1] 86.8
```

```
median(course.grades)
```

```
[1] 86.4
```

```
sd(course.grades) # grade standard deviation
```

```
[1] 6.625708
```

More Functions on Vectors

```
sort(course.grades)
```

```
[1] 78.8 82.0 86.4 92.6 94.2
```

```
max(course.grades) # highest course grade
```

```
[1] 94.2
```

```
min(course.grades) # lowest course grade
```

```
[1] 78.8
```

STOP: Understand Workflow

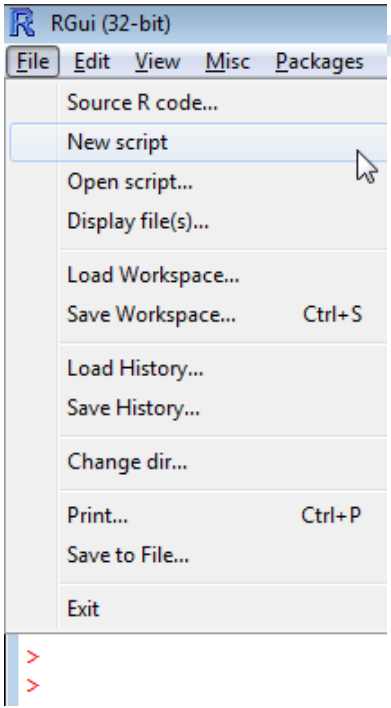
work·flow
/ˈwərkflō/

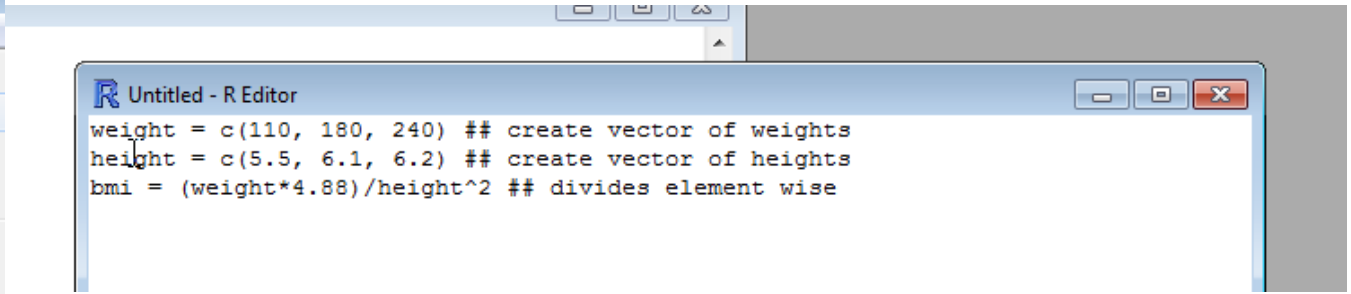
noun

the sequence of industrial, administrative, or other processes through which a piece of work passes from initiation to completion.

- Its important to works smart with R and keep your commands in a file for easy access [for editing, sending to people, using again and again].
- Rstudio is even fancier with this-see video on class web site.

Use an R Script File





- **To run the code in the script, highlight and hit ctrl-r**
- **To save the script for later use, choose File->Save**

Phew…Using R: Creating a Data Set

- > Scores = c(22, 34, 18, 29, 36)

  c means “concatenate” in R

  – in plain English “treat as data set”

- Now do:

  >Scores

R will print the data set

Important Rules

1. We created a variable
2. Variable names are case sensitive
3. No blanks in name

   (can use _ or . to join words, but not -)
4. Start with a letter (capitol or lowercase)

Non-numeric Data

- Enclose in quotes, single or double
- Separate entries with comma
- Example:

```
> names = c(“Mary”, “Tom”, “Ed”, “Dan”, “Meg”)
```

Finding objects you created

- To list the objects that you have in your current R session use the function ls().

  > ls()

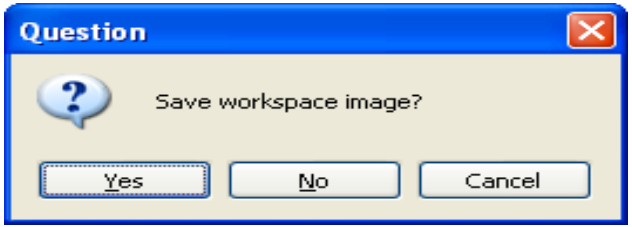
  [1] "x" "y"

- If you assign a value to an object that already exists then the contents of the object will be overwritten with the new value (without a warning!). Use the function rm to remove one or more objects from your session.

  > rm(x, y)

Saving Stuff

- To exit: quit( ) or q() [the function quit!]
- Brings up this screen:



- Do what you want: Yes or No
  - Do Yes,
  - then re-open R, get Scores & names

Using R Functions: Simple Stuff

- Commands for mean, sd, summary

  (NB: function names case sensitive)

  - mean(Scores)
  - sd(Scores)
  - summary(Scores)

- We will review some basic statistics next class.

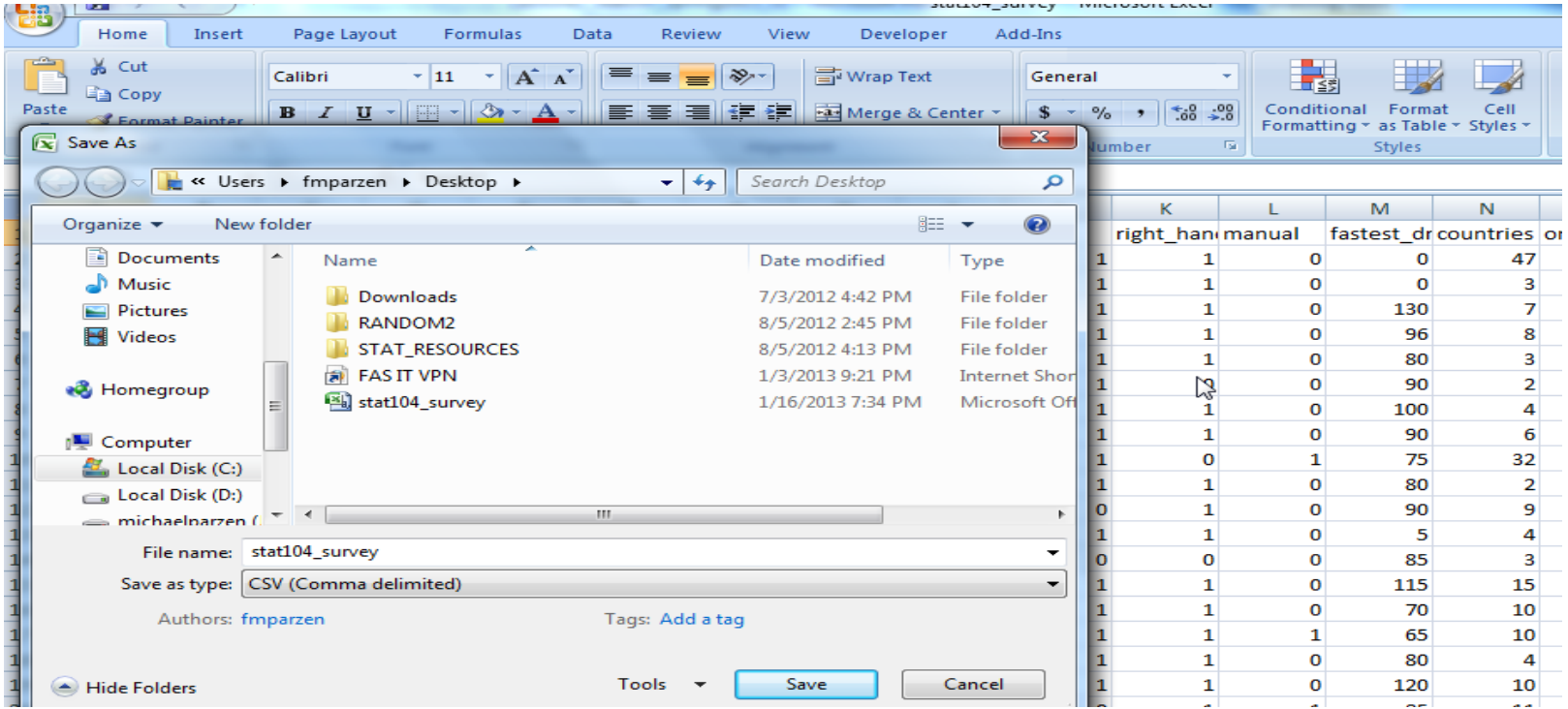
R functions

- A zillion of ‘em
- R’s big strength, most common use
- For examples:
  - Help
    - Google “+R +cran +standard deviation”
    - Enter name of a function [e.g., help(sd)]
  - Yields lots (!) of information

Importing Data

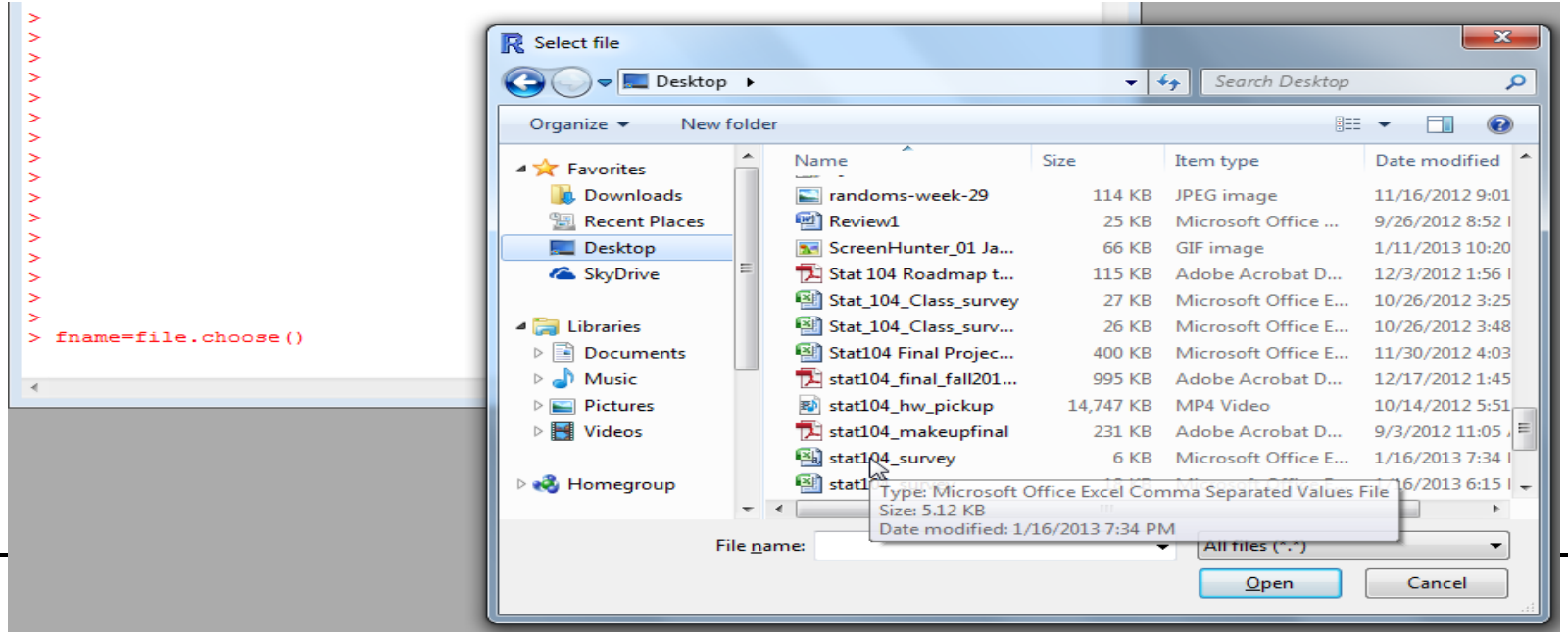
- How do we get data into R?
- Remember we have no point and click…
- First make sure your data is in an easy to read format such as CSV (Comma Separated Values).
- In Excel, you can use the “save-as” command to convert any file to csv format.

CSV from Excel



First we need to get the filename

- fname = file.choose()
- The file.choose command opens a window in



Now we need to read it in

- mydata = read.csv(fname)

```
> fname=file.choose()
> mydata=read.csv(fname)
> dim(mydata)
[1] 114  17
> names(mydata)
 [1] "height"        "weight"        "male"          "cellphones"
 [5] "looks"         "smoke"         "sleep"         "parzen_age"
 [9] "haircut"       "snap"          "right_handed"  "manual"
[13] "fastest_drive" "countries"     "only_child"    "text_day"
[17] "second_toe"
```

This data is from a Stat 104 class survey……..

Read data in from the internet

```
fname="http://people.fas.harvard.edu/~mparzen/stat107/stat104_survey.csv"
mydata=read.csv(fname)

> dim(mydata)
[1] 114  17

> head(mydata)
  height weight male cellphones looks smoke sleep parzen_age haircut snap
1     72    165    1          2  30.0     0     8         27       0    1
2     68    118    0          6  50.0     0     3         30      50    1
3     73    175    1          6  99.0     0     6         31      25    1
4     71    145    1          4  55.0     0     7         33      20    1
5     72    135    1          3  50.0     0     7         35      19    1
6     73    167    1          4   0.2     0     6         36       0    1
  right_handed manual fastest_drive countries only_child text_day second_toe
1            1      0             0        47          0       30          0
2            1      0             0         3          0      300          1
3            1      0           130         7          0       60          1
4            1      0            96         8          0       50          0
5            1      0            80         3          0       20          1
6            0      0            90         2          0       35          0
```

R commands are vectorized

- Vectorized sounds like a fancy word.
- It just means you can do a lot of things at once:

```
> mean(mydata)
      height       weight         male    cellphones        looks
 67.85087719 147.75438596    0.51754386    4.06140351 30.27807018
       smoke        sleep   parzen_age       haircut         snap
  0.05263158   6.74561404  45.24561404  34.67543860   0.89473684
right_handed       manual fastest_drive     countries   only_child
  0.90350877   0.26315789  90.14912281    8.40350877   0.16666667
    text_day   second_toe
 72.88157895   0.36842105

> sd(mydata)
      height       weight         male    cellphones        looks
   4.9977561   27.5107578    0.5018983     1.9968342   24.6447889
       smoke        sleep   parzen_age       haircut         snap
   0.2242827    1.2305938    6.6339985    29.1257559    0.3082471
right_handed       manual fastest_drive     countries   only_child
   0.2965673    0.4422915   28.9002656     7.4592928    0.3743234
    text_day   second_toe
 282.2299736    0.4845061
```

Working with data.

- Accessing columns.
- The variable **mydata** has our data in it…. But you can't see it directly.
- Well you can, but it's a huge 114x17 matrix

```
> dim(mydata)
[1] 114  17
> mydata
    height weight male cellphones looks smoke sleep parzen_age haircut snap
1       72    165    1          2  30.0     0   8.0          27       0    1
2       68    118    0          6  50.0     0   3.0          30      50    1
3       73    175    1          6  99.0     0   6.0          31      25    1
4       71    145    1          4  55.0     0   7.0          33      20    1
5       72    135    1          3  50.0     0   7.0          35      19    1
6       73    167    1          4   0.2     0   6.0          36       0    1
7       72    155    1          5  25.0     0   6.0          37       3    1
8       67    117    0          4  50.0     0   6.0          37      16    1
```

Selecting Columns

- To select a column use `mydata$column`.

```
> names(mydata)
 [1] "height"        "weight"        "male"          "cellphones"
 [5] "looks"         "smoke"         "sleep"         "parzen_age"
 [9] "haircut"       "snap"          "right_handed"  "manual"
[13] "fastest_drive" "countries"     "only_child"    "text_day"
[17] "second_toe"
> mydata$height
  [1] 72 68 73 71 72 73 72 67 71 69 71 75 63 72 79 63 65 54 67 70 65 54 69 66 64
 [26] 63 69 66 68 72 74 63 64 66 72 68 69 67 73 71 69 68 67 69 64 80 71 71 75 53
 [51] 71 75 64 68 72 68 73 66 68 74 69 68 68 70 64 64 70 65 63 70 73 66 70 76 63
 [76] 66 61 67 63 74 75 63 70 74 69 69 63 73 70 69 62 62 72 67 59 50 67 70 68 70
[101] 63 60 64 64 72 64 68 66 62 72 68 71 70 63
> mydata$sleep
  [1] 8.0 3.0 6.0 7.0 7.0 6.0 6.0 6.0 8.0 7.0 7.0 7.0 7.0 8.0 6.0 7.0 5.0 4.0
 [19] 7.0 5.0 5.0 6.0 5.0 5.0 7.0 8.0 7.0 8.0 7.0 7.0 7.0 7.0 7.0 7.0 7.0 6.0
 [37] 7.5 7.0 7.0 7.0 8.0 7.0 8.5 7.0 8.0 8.0 7.0 7.0 8.0 6.0 8.0 7.0 8.0 7.5
 [55] 7.0 7.0 7.0 7.0 7.0 6.0 7.0 7.0 5.0 7.0 6.0 8.0 8.0 8.0 7.0 7.0 8.0 6.0
 [73] 7.0 7.0 7.0 7.0 8.0 8.0 7.0 7.0 6.5 5.0 8.0 5.5 5.0 6.0 7.0 7.0 7.0 8.0
 [91] 7.0 4.0 5.0 6.0 9.0 8.0 7.0 9.0 6.0 7.0 9.0 5.0 8.0 7.0 6.5 8.0 7.0 5.0
[109] 6.0 6.0 7.0 4.0 7.0 1.0
```

Selecting Columns

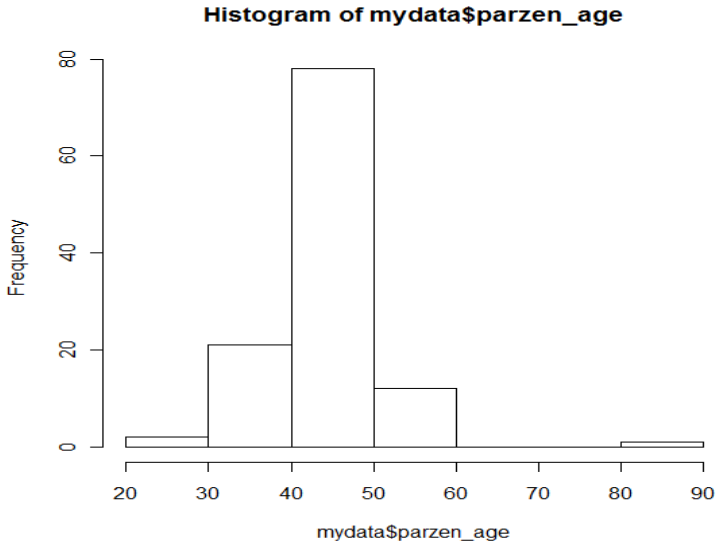
- One can also explicitly ask for different columns of the data matrix
- mydata[,5] returns all rows of the fifth column
- mydata[1,] returns the first row, all columns
- mydata[1:10,] returns the first ten rows
- mydata[,5:7] returns all rows for columns 5-7.
- head(mydata) – first few rows of data
- tail(mydata) – last few rows of data

Working with data.

- Subsetting data.
- Use a logical operator to do this.
  - ==, >, <, <=, >=, <> are all logical operators.
  - Note that the “equals” logical operator is two = signs.
- Example:
  - `mydata[mydata$male == 1,] (note the comma!)`
  - This will return the rows where Gender is male (coded 1).
  - Remember R is case sensitive!
  - This code does nothing to the original dataset.
  - `mydata.male = mydata[mydata$male == 1,]` gives a dataset with the appropriate rows.

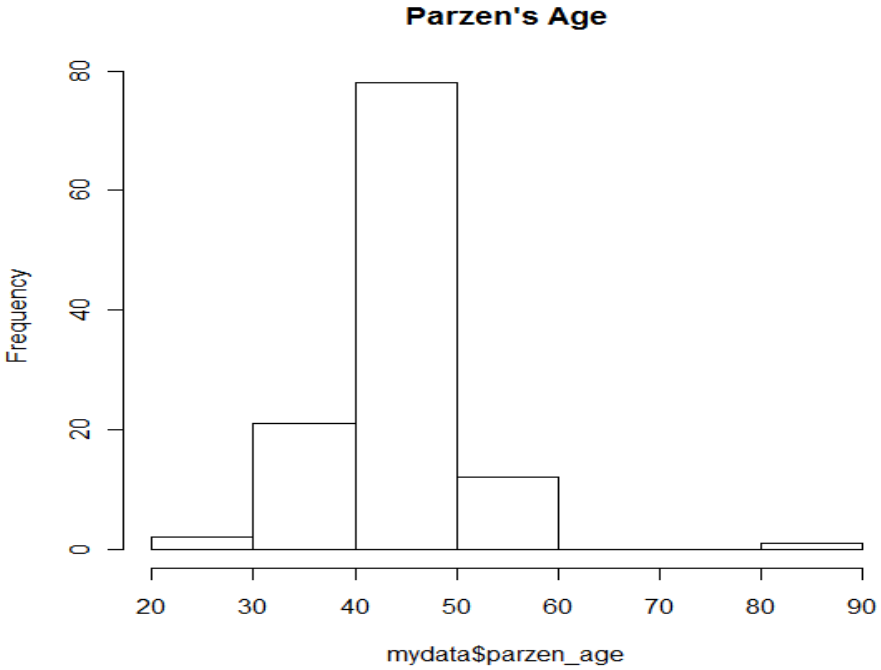
Basic Graphics

- Histogram
  - `hist(mydata$parzen_age)`



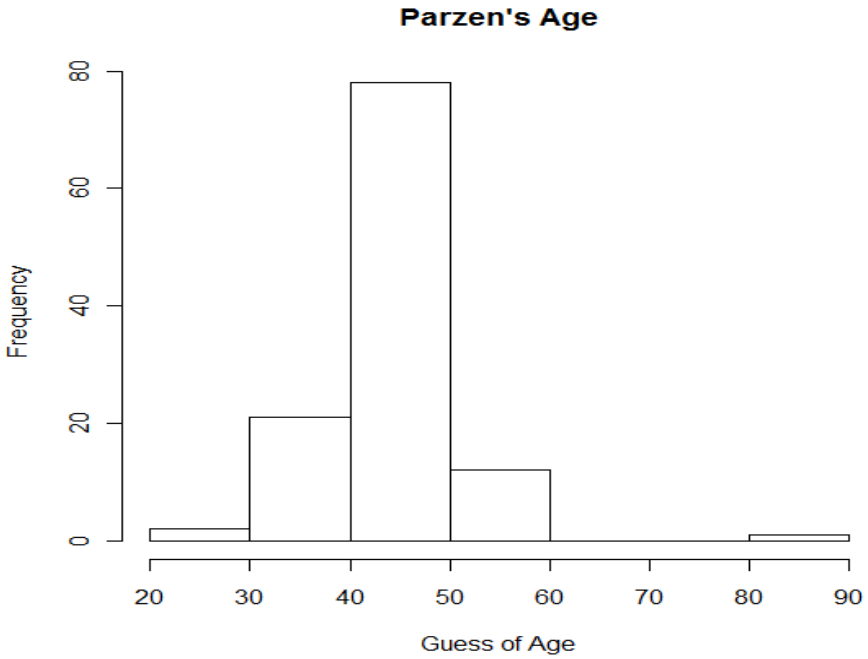
Basic Graphics

- Add a title…
  - The “main” statement will give the plot an overall heading.
  - `hist(mydata$parzen_age ,main="Parzen's Age")`
  - `Pro tip-use arrow keys to repeat previous commands, then edit the previous command.`



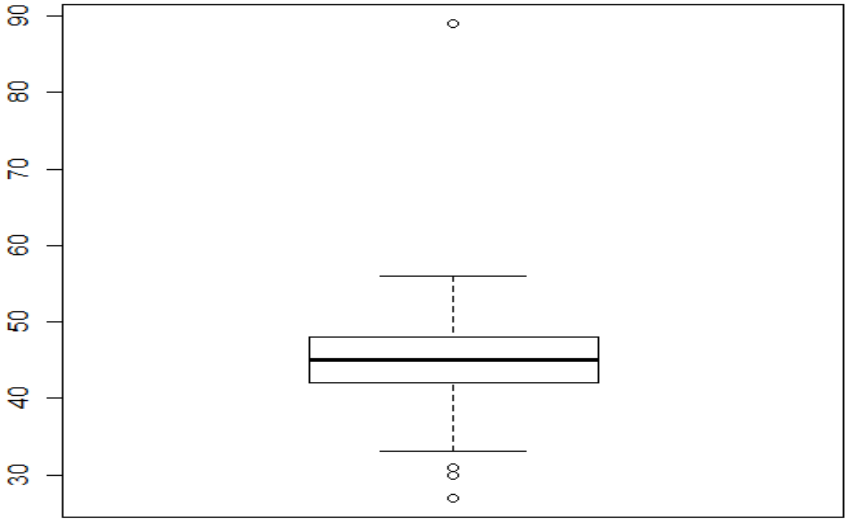
Basic Graphics

- Adding axis labels…
- Use “xlab” and “ylab” to label the X and Y axes, respectively.
- `hist(mydata$parzen_age, main="Parzen's Age",xlab="Guess of Age", ylab="Frequency")`



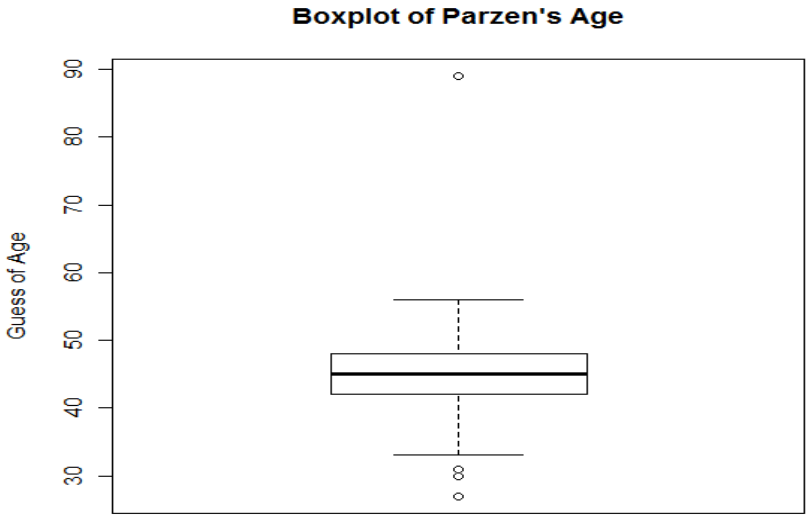
Basic Plots

- Box Plots
  - `boxplot(mydata$parzen_age)`



Boxplots

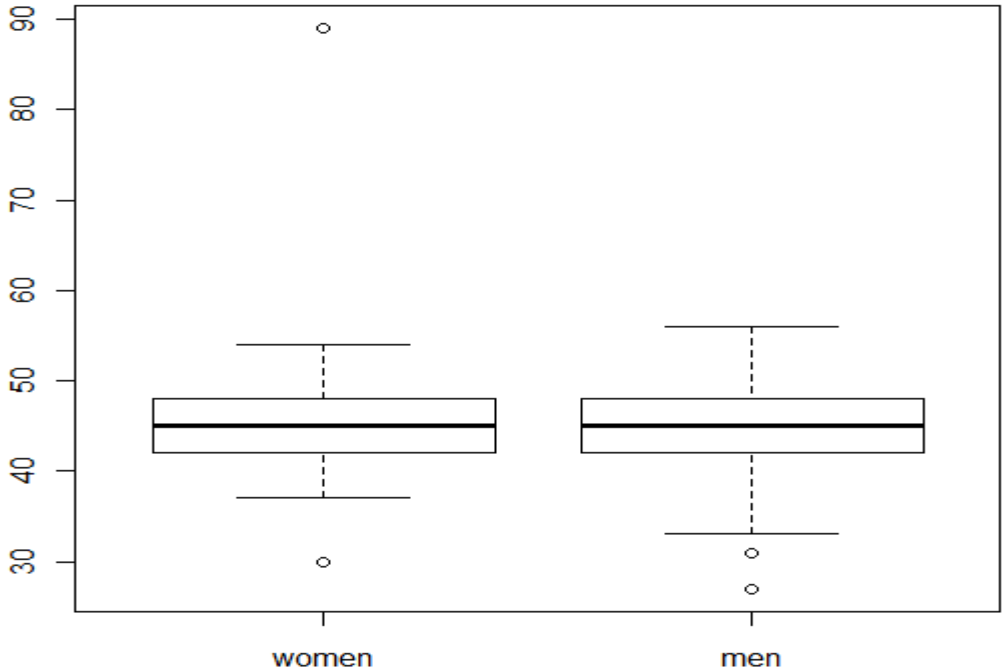
- Change it!
- `boxplot(mydata$parzen_age ,main="Boxplot of Parzen's Age",ylab="Guess of Age")`



Box-Plots - Groupings

- What if we want several box plots side by side to be able to compare them.
- First Subset the Data into separate variables.
  - `mydata.m=mydata[mydata$male==1,]`
  - `mydata.w=mydata[mydata$male==0,]`
- Then Create the box plot.
  - `boxplot(mydata.m$parzen_age,mydata.f$parzen_age)`
  - `boxplot(mydata.w$parzen_age,mydata.m$parzen_age, names=c("women","men"))`

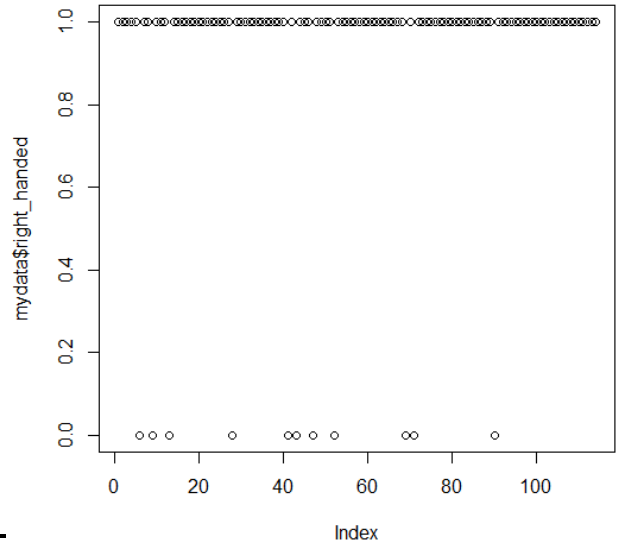
Boxplots – Groupings



Discrete Data Plots

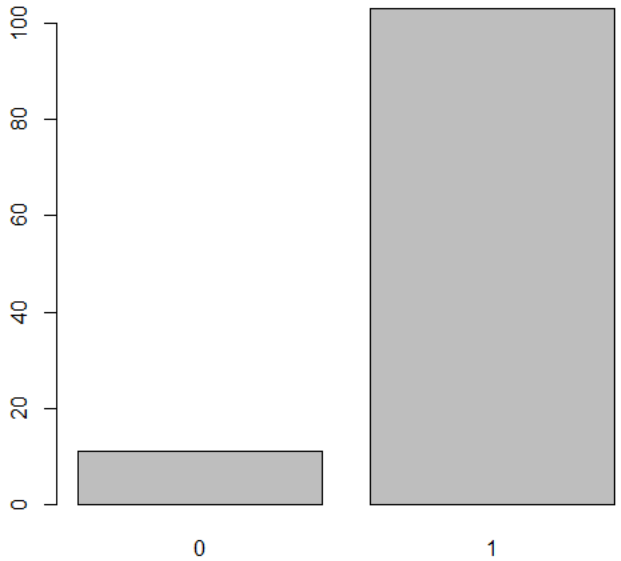
- We have a variable for whether someone is right handed

```
> plot(mydata$right_handed)
```



Discrete Data Plots

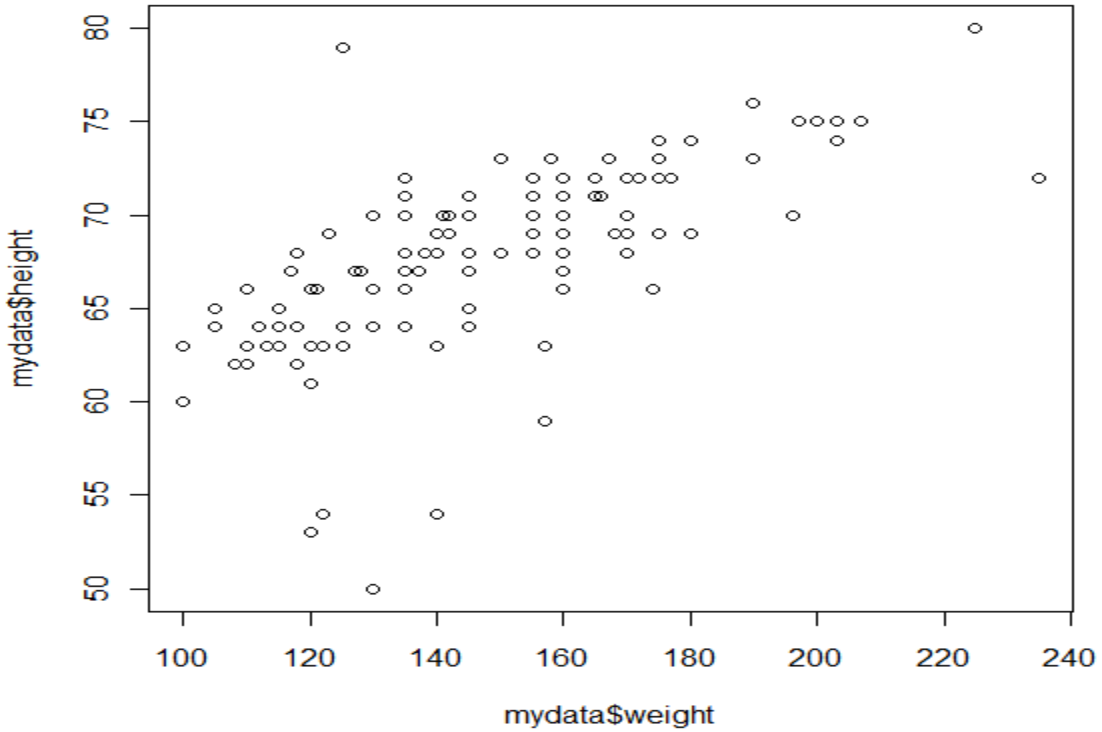
```
> counts=table(mydata$right_handed)
> counts

  0   1
 11 103
> barplot(counts)
```



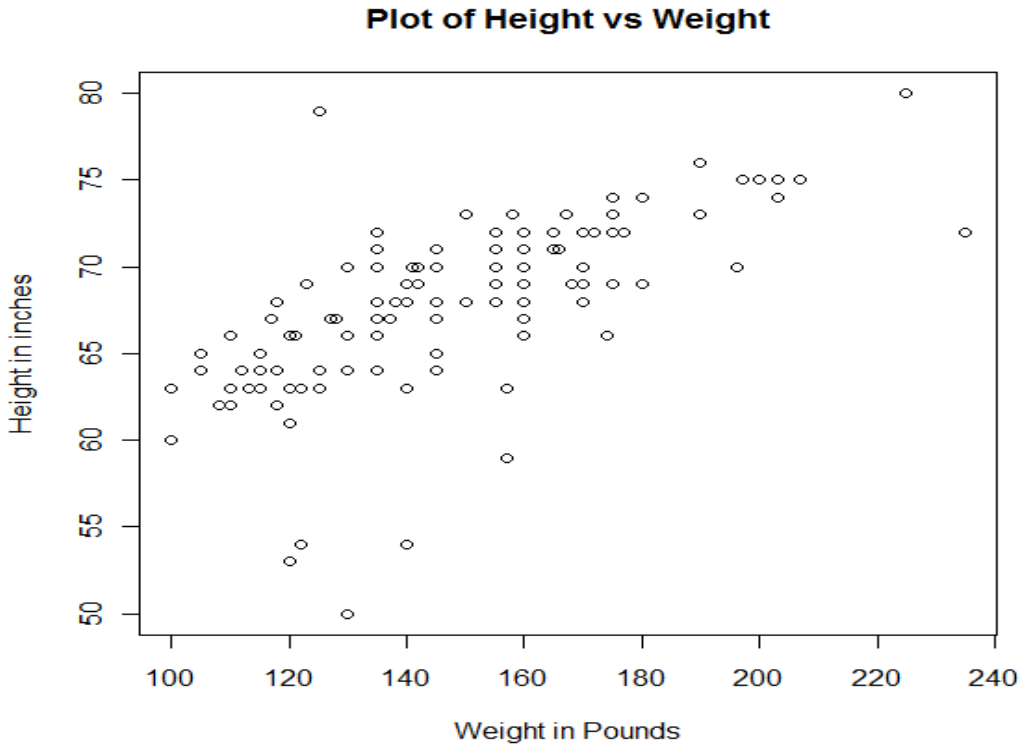
Scatter Plots

- Suppose we have two variables and we wish to see the relationship between them.
- A scatter plot works very well.
- R code:
  - `plot(x,y)`
- Example
  - `plot(mydata$weight,mydata$height)`

Scatterplots



Scatterplots




```
plot(mydata$weight,mydata$height,xlab="Weight in Pounds",
 ylab="Height in inches", main="Plot of Height vs Weight")
```

Line Plots

- Often data comes through time.
- We will cover the quantmod package more in a little bit.
- Consider $AAPL stock

```
> library(quantmod)
> getSymbols("AAPL")
[1] "AAPL"
> head(AAPL)
           AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted
2007-01-03     86.29     86.58    81.90      83.80   309579900      11.19449
2007-01-04     84.05     85.95    83.82      85.66   211815100      11.44295
2007-01-05     85.77     86.20    84.40      85.05   208685400      11.36147
2007-01-08     85.96     86.53    85.28      85.47   199276700      11.41757
2007-01-09     86.45     92.98    85.15      92.57   837324600      12.36603
```

Line Plots


$AAPL Daily Closing Prices
20
40
60
80
100
120
Jan 03 2007
Jul 01 2009
Jan 03 2012
Jul 01 2014

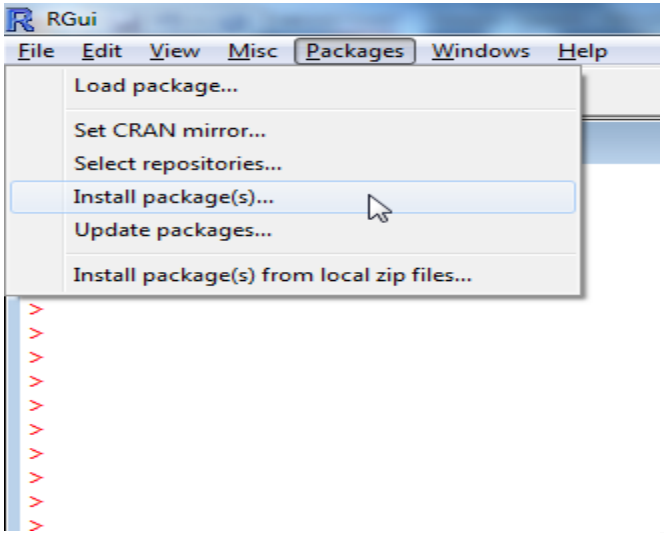
R Packages

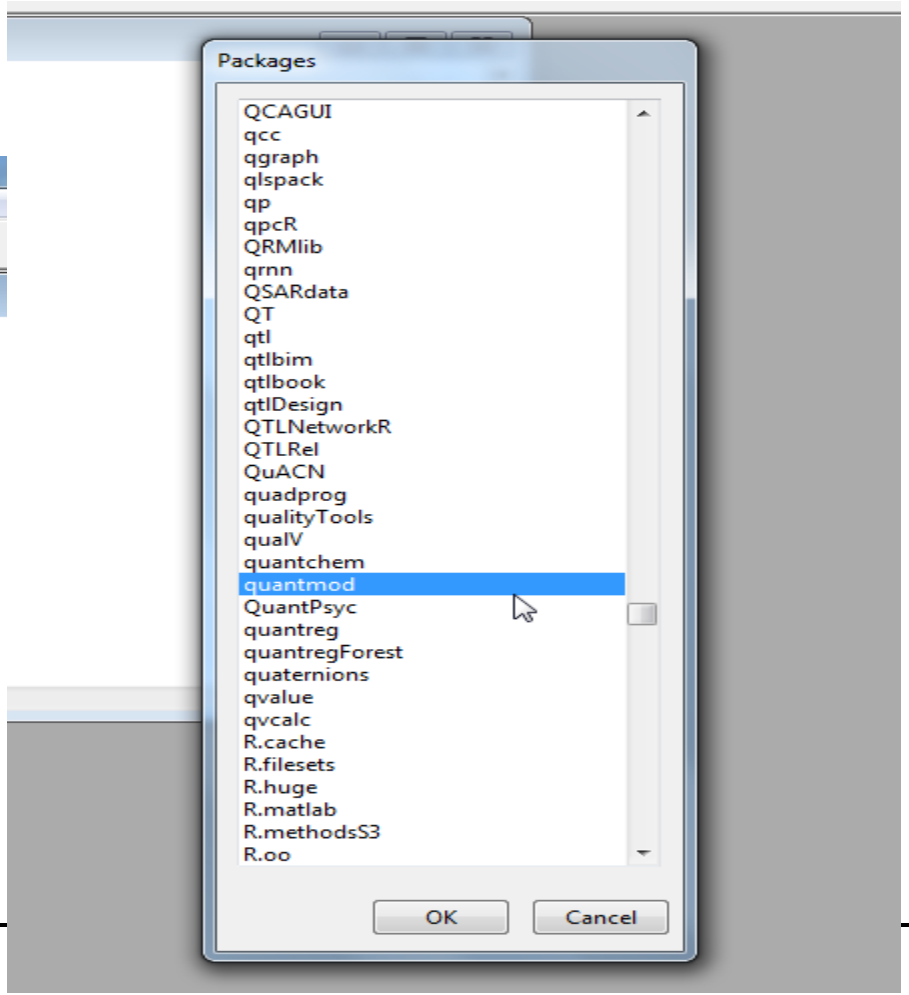
- One of the strengths of R is that the system can easily be extended.
- The system allows you to write new functions and package those functions in a so called `R package' (or `R library'). The R package may also contain other R objects, for example data sets or documentation.
- Just a few examples, there are packages for portfolio optimization, drawing maps, exporting objects to html, time series analysis, spatial statistics and the list goes on and on.

The QUANTMOD Package

- From the package description:
  - The quantmod package for R is designed to assist the quantitative trader in the development, testing, and deployment of statistically based trading models.

Installing





Or just say
install.packages("quantmod")

The Library Command

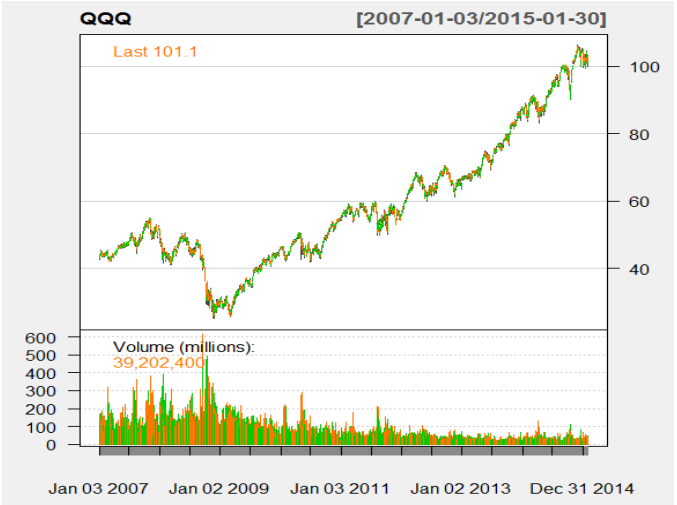
- Remember, once the package has been installed, it isn't automatically loaded into R.
- To use the package, you need to specify it via the library command. This has to be done once per session.
- Syntax: library(quantmod) [or whatever the package name is]

An Introductory Package

- There is a new R package called Swirl that runs an introduction to R inside R.
- See http://swirlstats.com/students.html for instructions
- `install.packages("swirl")`
- `library(swirl)`
- `swirl()`

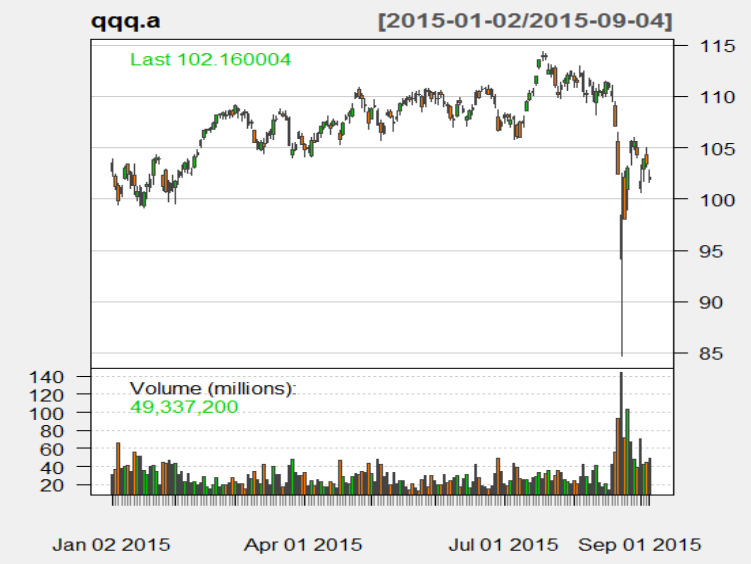
Quantmod Example (more next time)

```
> library(quantmod)
> getSymbols("QQQ")
[1] "QQQ"
chartSeries(QQQ, theme=chartTheme('white')
```



Quantmod Example

```
library(quantmod)
getSymbols("QQQ",from="2015-01-01")
qqq.a=adjustOHLC(QQQ)
chartSeries(qqq.a, theme=chartTheme('white'))
```

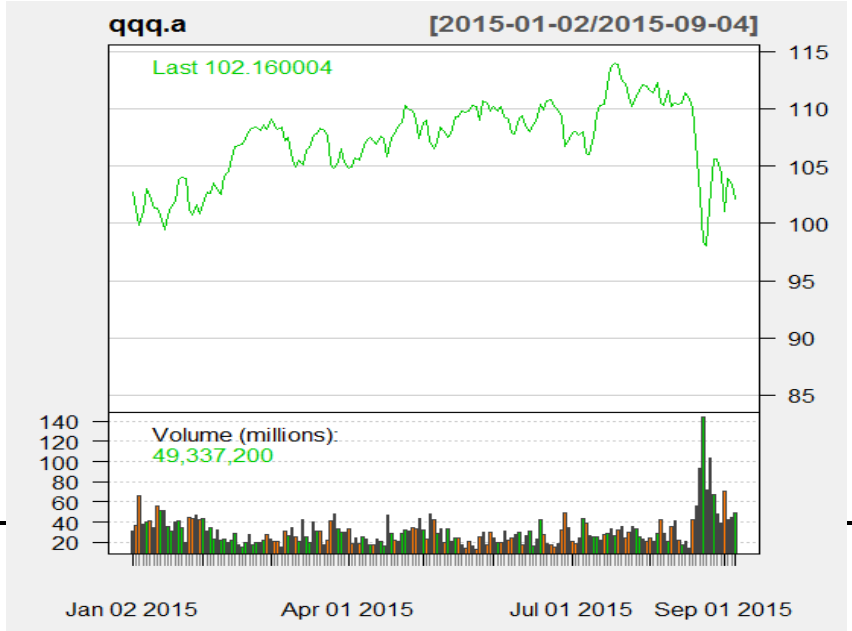


This is bar plot showing the range of each day

Quantmod Graph

- One can change the type of chart using the “type=“ switch. Typical R behavior

```
> chartSeries(qqq.a, theme=chartTheme('white'),type="l")
```



What does getSymbols do?

- Returns a **matrix** with the desired data

```
> dim(qqq.a)
[1] 171   6
> head(qqq.a)
           QQQ.Open QQQ.High   QQQ.Low QQQ.Close QQQ.Volume QQQ.Adjusted
2015-01-02 103.5119 103.9609 102.20493 102.70379   31148800    102.70378
2015-01-05 102.2648 102.3745 100.90791 101.19725   36521300    101.19725
2015-01-06 101.3469 101.5165  99.39141  99.84037   66205500     99.84037
2015-01-07 100.5088 101.3669 100.25940 101.12741   37577400    101.12741
2015-01-08 102.0154 103.2625 101.87569 103.06296   40212600    103.06296
2015-01-09 103.3722 103.4122 101.78589 102.38452   41410100    102.38452
```

- getSymbols is an unusual function since it automatically (and silently) creates an object.

Let's review this object

- There are some shortcut commands

```
> tail(Op(qqq.a))
           QQQ.Open
2015-08-28   105.08
2015-08-31   105.03
2015-09-01   101.68
2015-09-02   102.89
2015-09-03   104.32
2015-09-04   101.97
> tail(Cl(qqq.a))
           QQQ.Close
2015-08-28    105.62
2015-08-31    104.31
2015-09-01    101.05
2015-09-02    103.90
2015-09-03    103.39
2015-09-04    102.16
```

Op(), Vo(), Ad(), Cl(),Hi(),Lo()

Paneling Graphics

- Suppose we want more than one graphic on a panel.
- We can partition the graphics panel to give us a framework in which to panel our plots.
- `par(mfrow = c( nrow, ncol))`

Number of rows

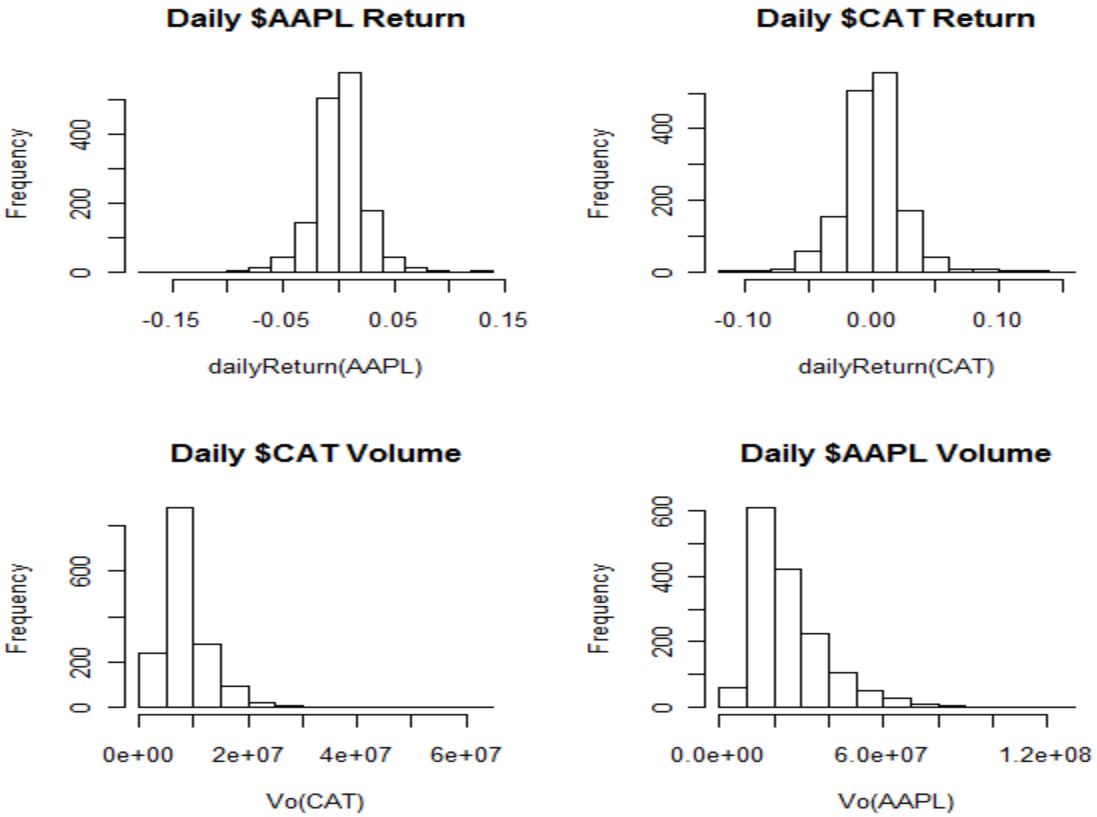
Number of columns

Paneling Graphics

- Consider the following

```
> par(mfrow=c(2,2))
> hist(dailyReturn(AAPL),main="Daily $AAPL Return")
> hist(dailyReturn(CAT),main="Daily $CAT Return")
> hist(Vo(CAT),main="Daily $CAT Volume")
> hist(Vo(AAPL),main="Daily $AAPL Volume")
```

Paneling Graphics



Functions

- We have seen a lot of built-in functions: mean(), sd(), plot(), read.csv()…
- An important part of programming and data analysis is to write custom functions
- Functions help make code **modular**
- Functions make debugging easier
- Remember: this entire class is about applying *functions* to *data*

A Really Simple Function

- A really simple function

```
addOne = function(x) {
  x+1
}
```

- This function adds 1 to its input

```
> addOne(10)
[1] 11
```

Another function example

- Three number summary function

```
threeNumberSummary = function(x) {
  c(mean=mean(x), median=median(x), sd=sd(x))
}

> x <- rnorm(100, mean=5, sd=2) # Vector of 100 normals with mean 5 and sd 2

> threeNumberSummary(x)
    mean   median       sd
4.766603 4.754881 1.944234
```

R Function Example

- Code

```
mysquare = function(x)
{
  cat("The square of ",x,"is",x*x,"\n")
}
```

- What do you think this function does?

Running the Function

```
> mysquare(5)
The square of  5 is 25
> mysquare(10)
The square of  10 is 100
> mysquare("mike")
Error in x * x : non-numeric argument to binary
operator
> mysquare(mike)
The square of  3 is 9
> mike
[1] 3
```

Modifying the Function

- This function now also returns a value

```
mysquare = function(x)
{
  cat("The square of ",x,"is",x*x,"\n")
  return(x*x)
}
```

Running the new function

```
> mysquare(5)
The square of  5 is 25
[1] 25
> mike
[1] 3
> mike=mysquare(5)
The square of  5 is 25
> mike
[1] 25
```

Things you should know

- How to install R
- How to read csv files into R
- How to load packages into R (quantmod)
- How to write very simple functions in R