R - An Open Source Option for Statistical Computing and Visualization

Joey Bernard - ACEnet

What we'll cover

- An Introduction to R
- Working with Modules

An Introduction to R

What is R?

- R is a language and environment for statistical computing and graphics
- Developed at Bell Labs
- Very similar to S, open source
- Main web site is at
  - http://www.r-project.org

What's Available?

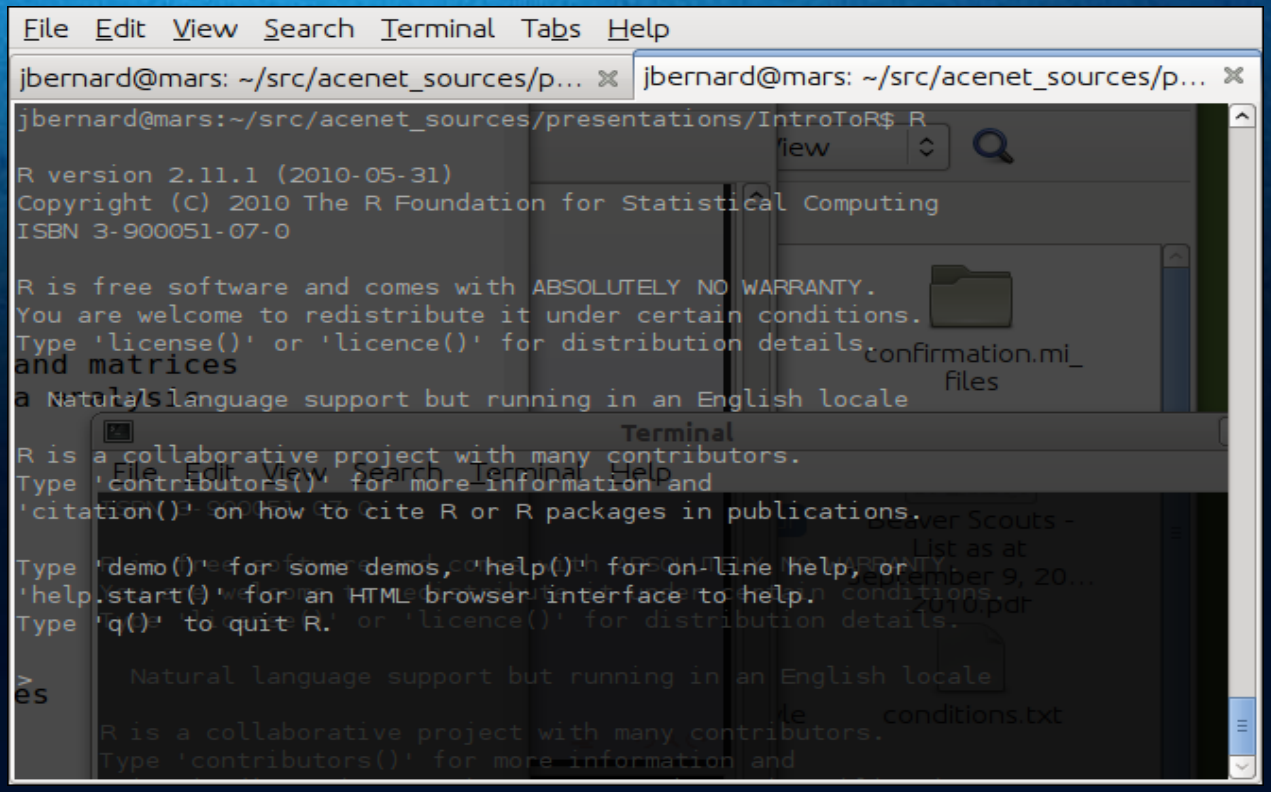
- Effective data handling
- Suite of operators for calculations on arrays and matrices
- Large collection of intermediate tools for data analysis
- Graphical facilities
- Full programming language, so you can develop new functions

How do we start and stop?

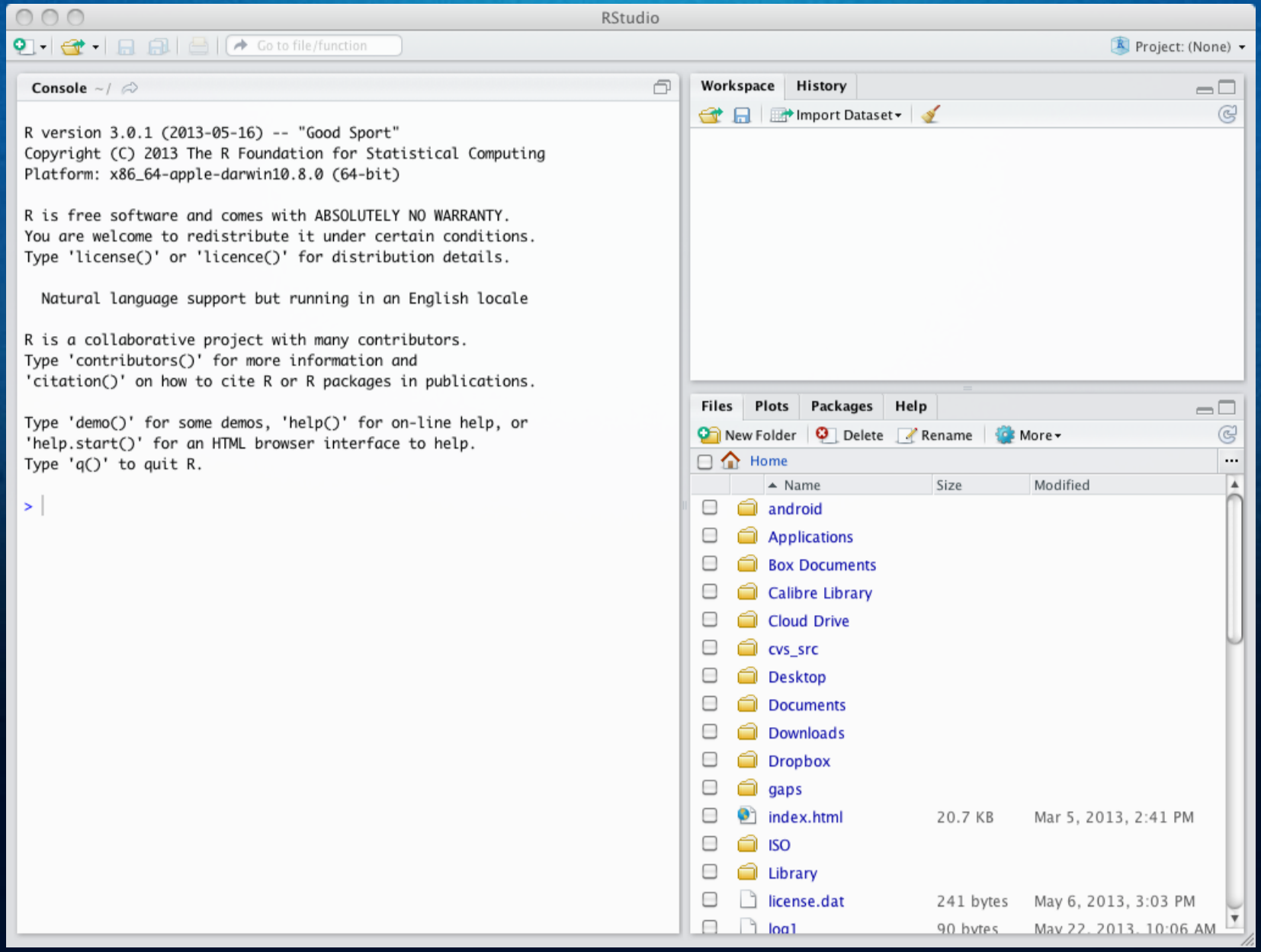
- R has both command line and graphical interfaces
- Examples of GUI's: RCommander, Rgui
- We will use RStudio in the workshop
- To start the command line interface

  R
- To shutdown R, use the quit command

  q()



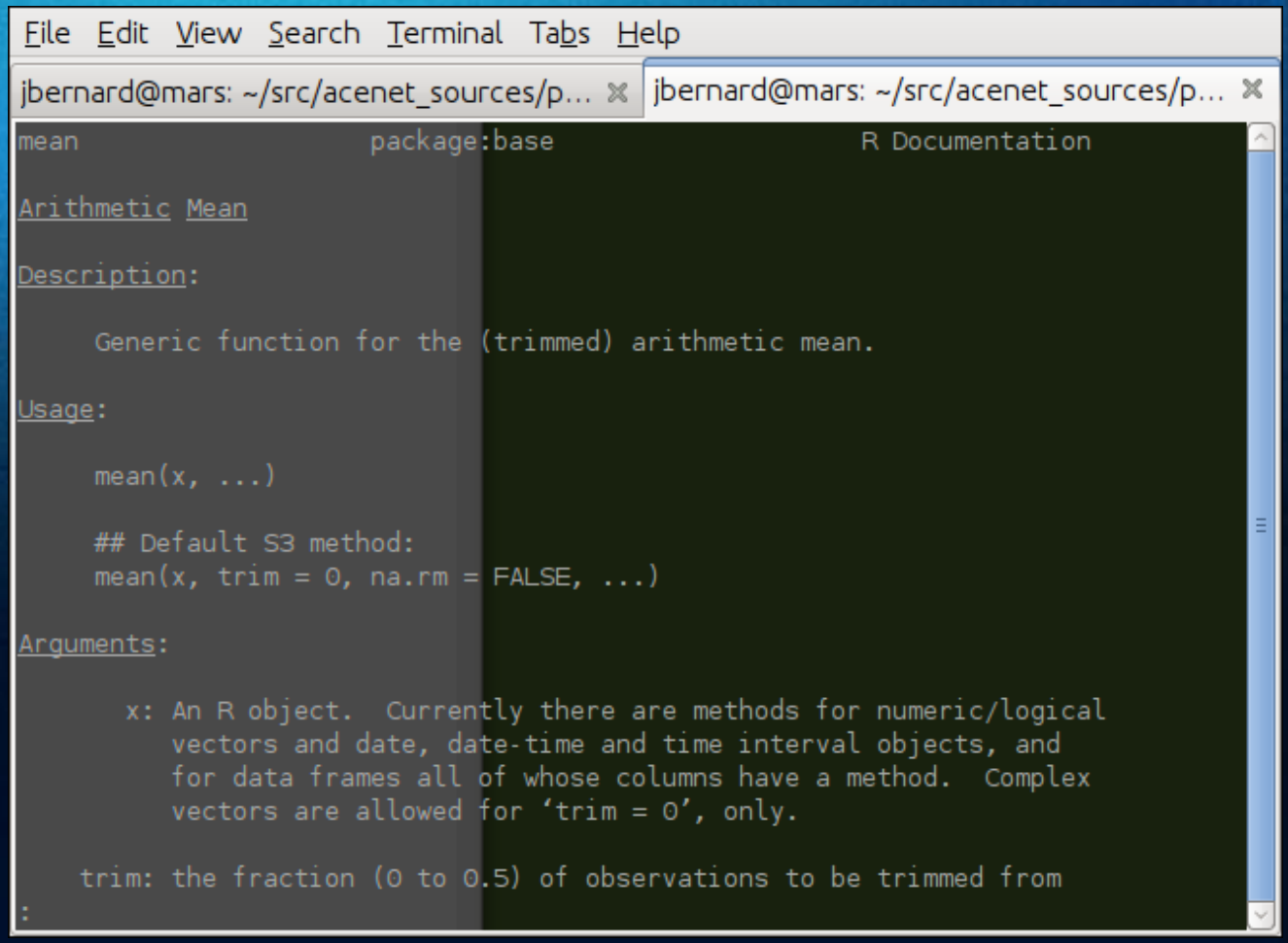
RStudio GUI


RStudio
Go to file/function
Project: (None)
Console ~/
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Workspace
History
Import Dataset
Files
Plots
Packages
Help
New Folder
Delete
Rename
More
Home
Name
Size
Modified
android
Applications
Box Documents
Calibre Library
Cloud Drive
cvs_src
Desktop
Documents
Downloads
Dropbox
gaps
index.html
20.7 KB
Mar 5, 2013, 2:41 PM
ISO
Library
license.dat
241 bytes
May 6, 2013, 3:03 PM
log1
90 bytes
May 22, 2013, 10:06 AM

Getting Help

- R has help files for all of the commands
- There are two methods

help(command)
?command



Running Scripts

- If you have a series of R commands you wish to run as one block, you can use the source command

  source("commands.R")
- If you want to send your output to a file, you can use the sink command

  sink("output.lst")
- To reset the output back to the console

  sink()

Loading Data

- The first step is to load data
- If the data sets are small, you can do it directly in R by using the concatenation function "c"

```
data1 = c(2,3,4,2,0,1,2)
```

- This dumps the vector of values into the variable data1, which you can use in functions

```
mean(data1)
[1] 2
```

Loading Data 2

- Larger amounts of data can be read in from files containing tables
- Tables of data need to have a special format
  - The first line of the file should have a name for each variable in the data
  - Each additional line of the file has as its first item a row label and the values for each variable
- This data can be read in using the function read.table()

```
DataToUse <- read.table("research.data")
```

Playing with data

- Individual data items can be manipulated with "[]" characters
- Printing out the second item is

```
typos[2]
[1] 3
```

- Setting a new value for the third item

```
typos[3] = 5
```

- Vectors can have sub-vectors

```
typos.list1 = c(1,2,3,4)
typos.list2 = c(2,4,6,8)
```

Playing with data - 2

- Getting the length of a vector

  ```
  length(typos)
  [1] 7
  ```

- Finding location of a certain value

  ```
  typos == 3
  ```

Statistics Basics

- The first things you probably want to see: mean, variance, standard deviation

```
> whale = c(74,122,235,111,292,111,211,133,156,79)
> mean(whale)
[1] 152.4
> var(whale)
[2] 5113.378
> std(whale)
Error: couldn't find function "std"
```

- No standard deviation? We'll make our own

```
> std = function(x) sqrt(var(x))
> std(whale)
[3] 71.50789
```

Statistics Basics - 2

- Looking a little harder, we find the built-in standard deviation function

```
> sd(whale)
[4] 71.50789
```

Statistics Basics - 3

- Get a frequency table of unique values

```
> x = c("yes", "no", "no", "yes", "yes")
> table(x)
x
   no   yes
    2     3
```

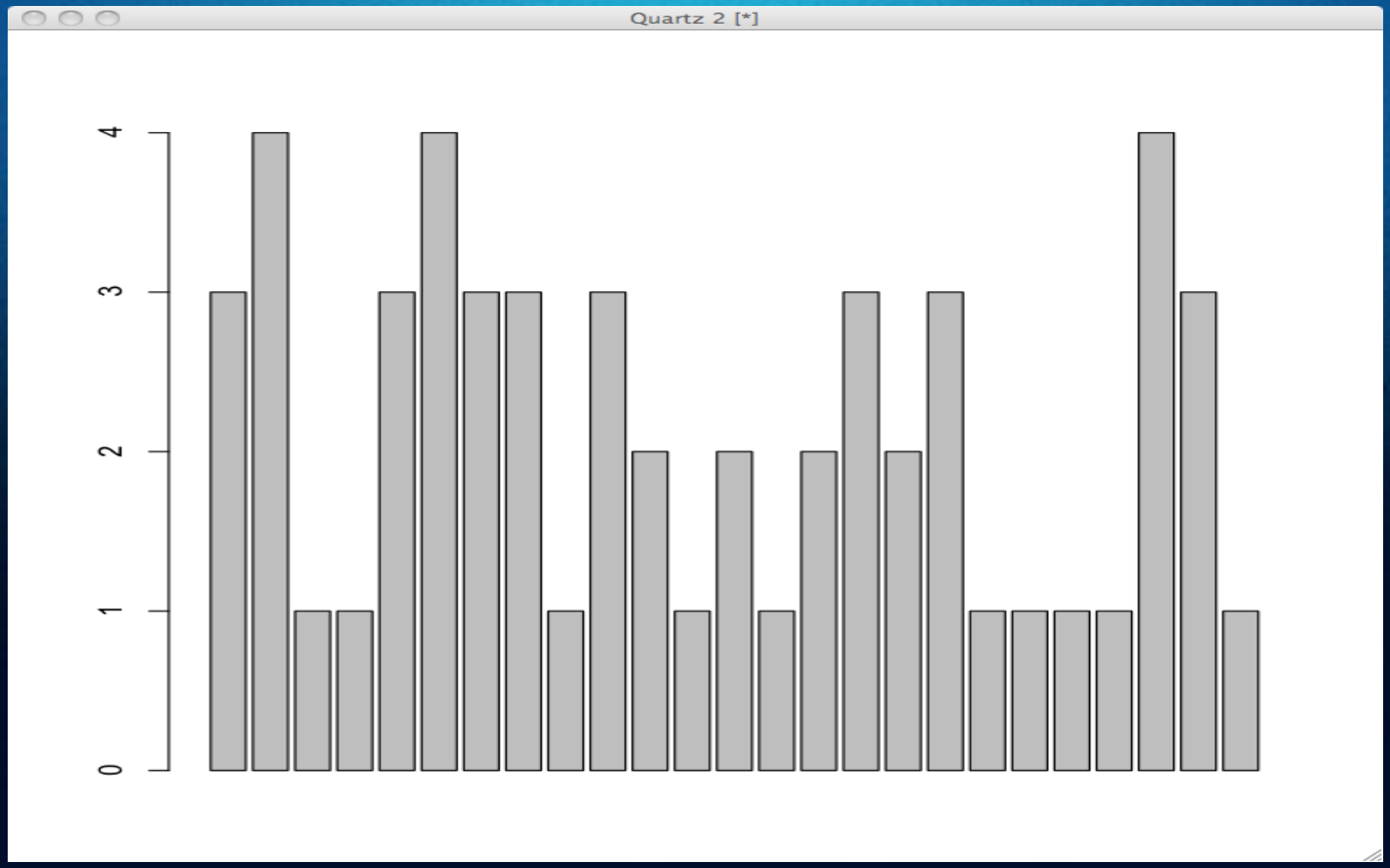
- Working with factors for categorical data

```
> x
[1] "yes" "no" "no" "yes" "yes"
> factor(x)
[1] yes no no yes yes
Levels: no yes
```

Graphics Basics

Barplot of data, where each value is plotted individually

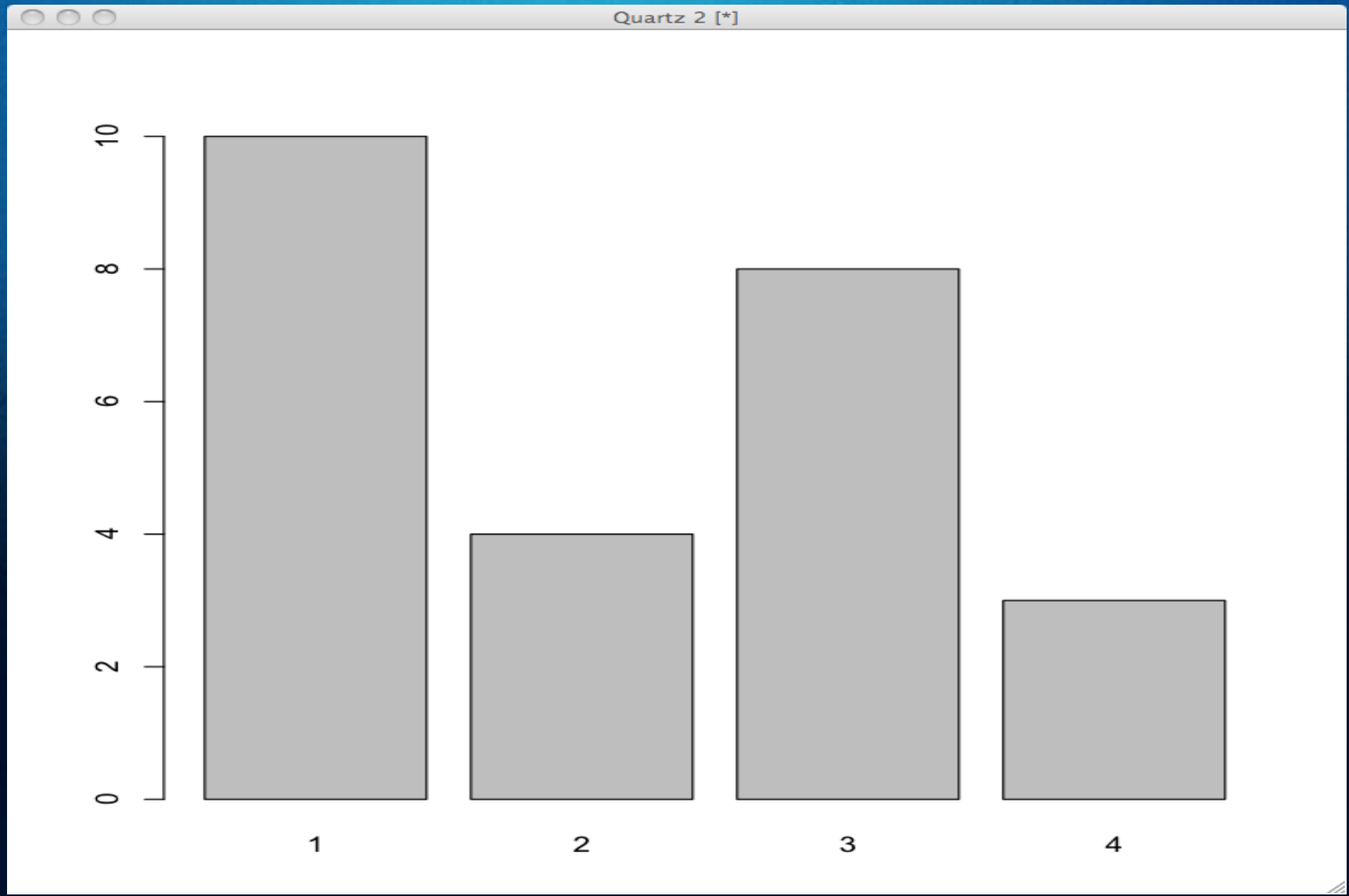
```
> bear = c(3,4,1,1,3,4,3,3,1,3,2,1,2,1,2,3,2,3,1,1,1,1,4,3,1)
> barplot(bear)
```



Graphics Basics - 2

Barplot of frequency of data

```
> barplot(table(bear))
```

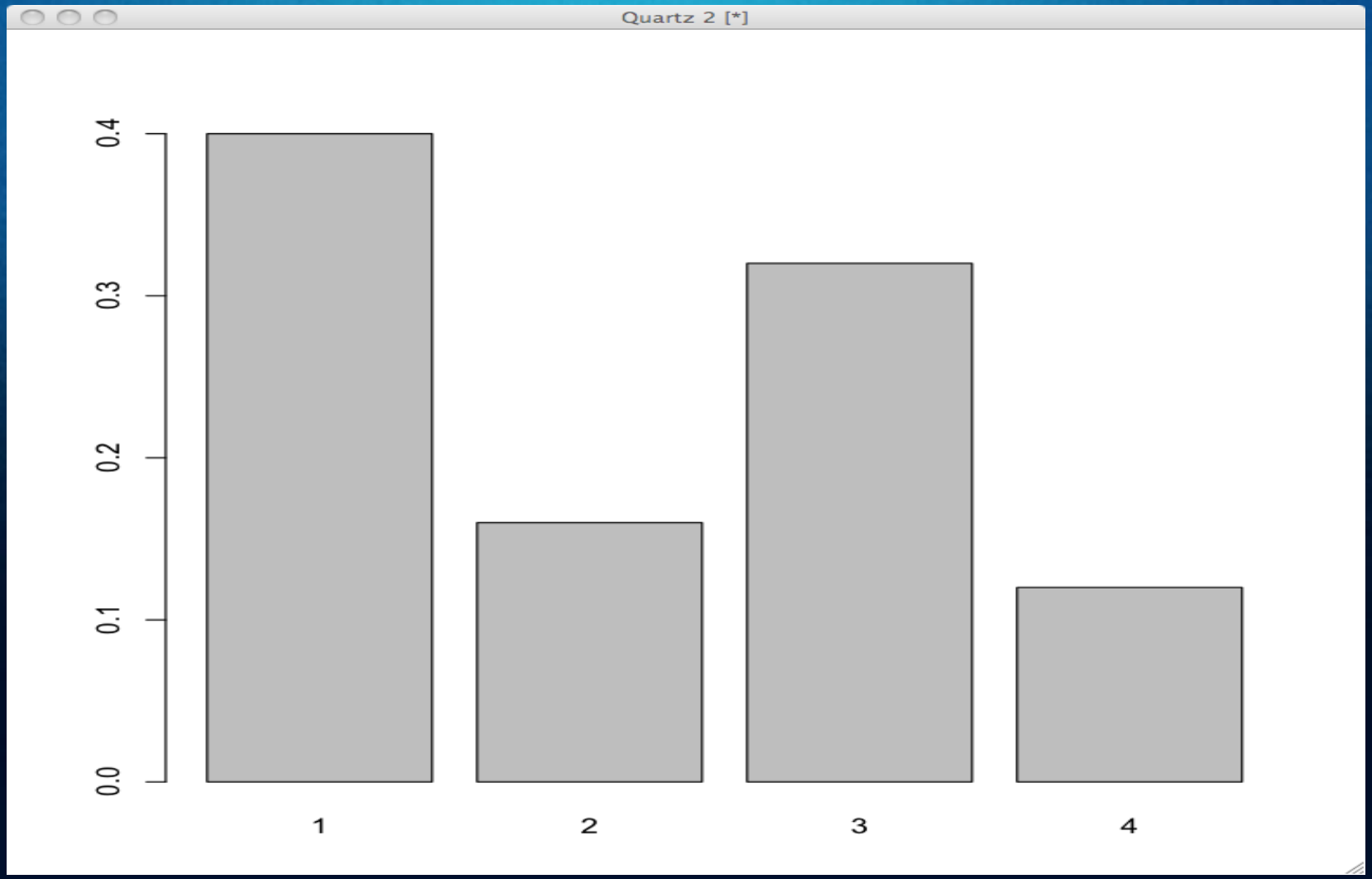


Graphics Basics - 3

You can even plot the results of functions

e.g. barplot of proportions

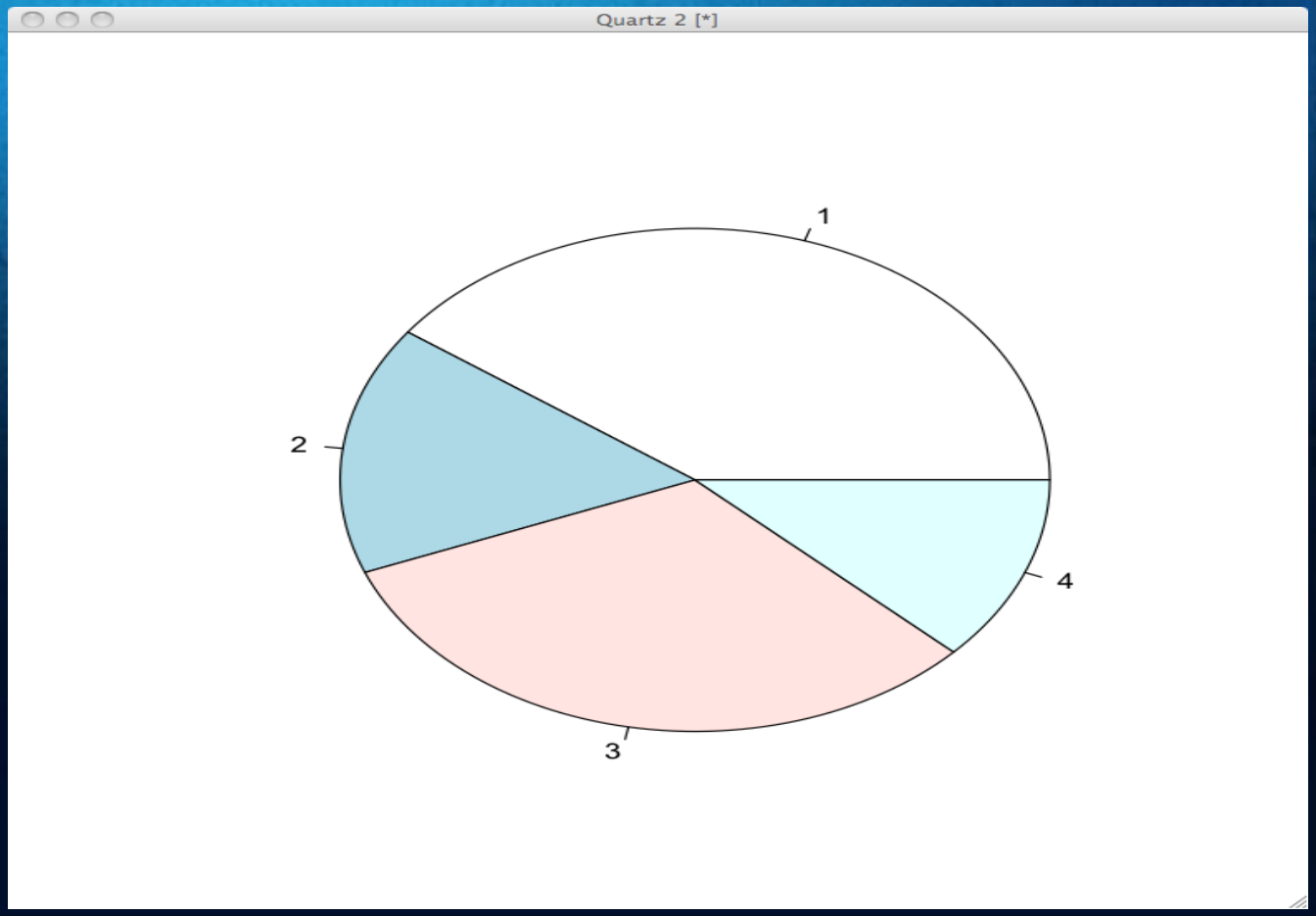
```
> barplot(table(bear)/length(bear))
```



Graphics Basics - 4

Basic Pie Charts

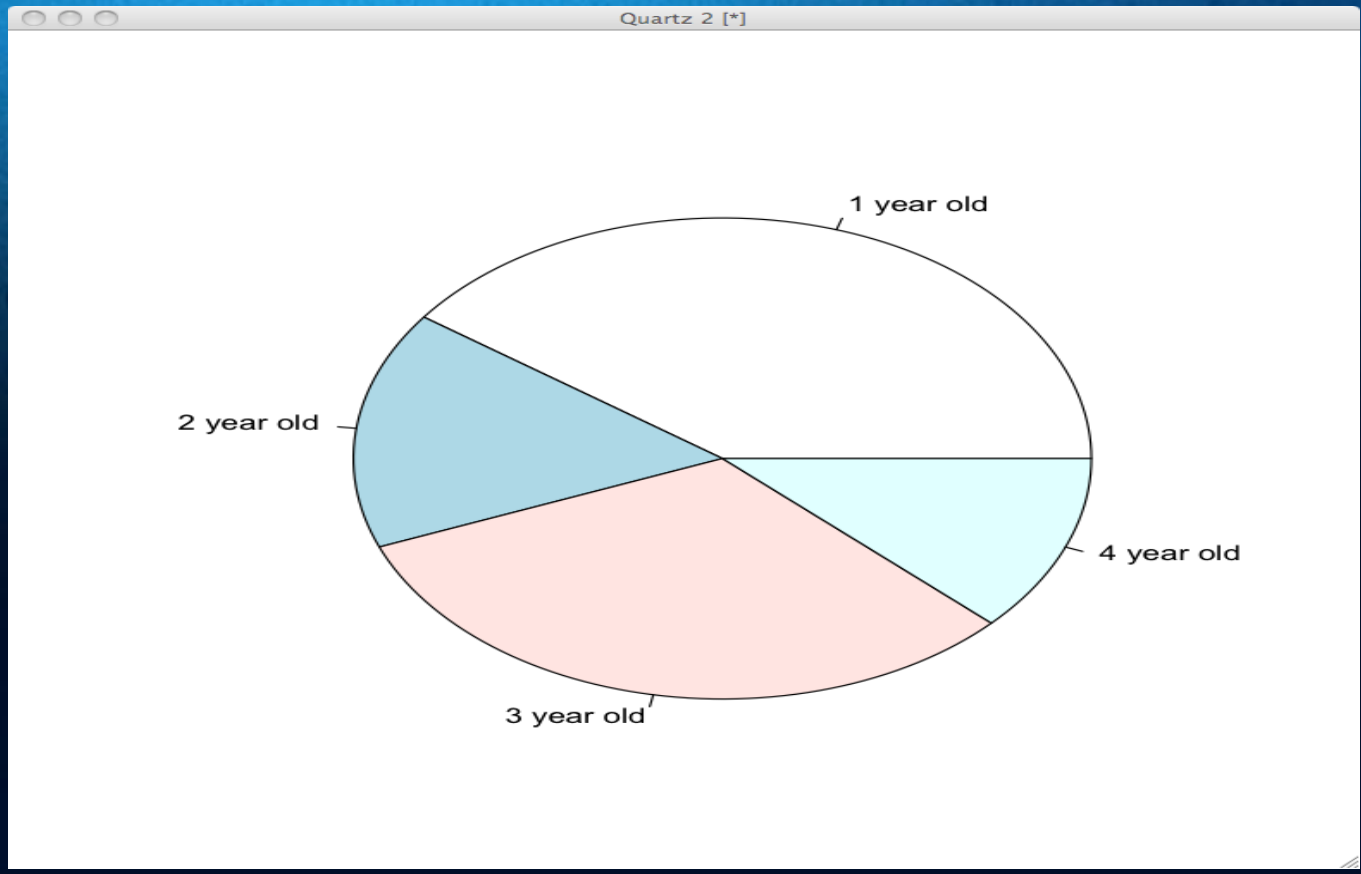
```
> bear.counts = table(bear)
> pie(bear.counts)
```



Graphics Basics - 5

Adding labels is easy

```
> names(bear.counts) = c("1 year old", "2 year old", "3 year old", "4 year old")
> pie(bear.counts)
```



Loading Modules

- You can load a module with the library command

  ```
  > library("ts")       # load the time series module
  ```

- You can find out what modules or libraries are available

  ```
  > library()
  ```

- There are lots of sample data sets available

  ```
  > data()
  ```

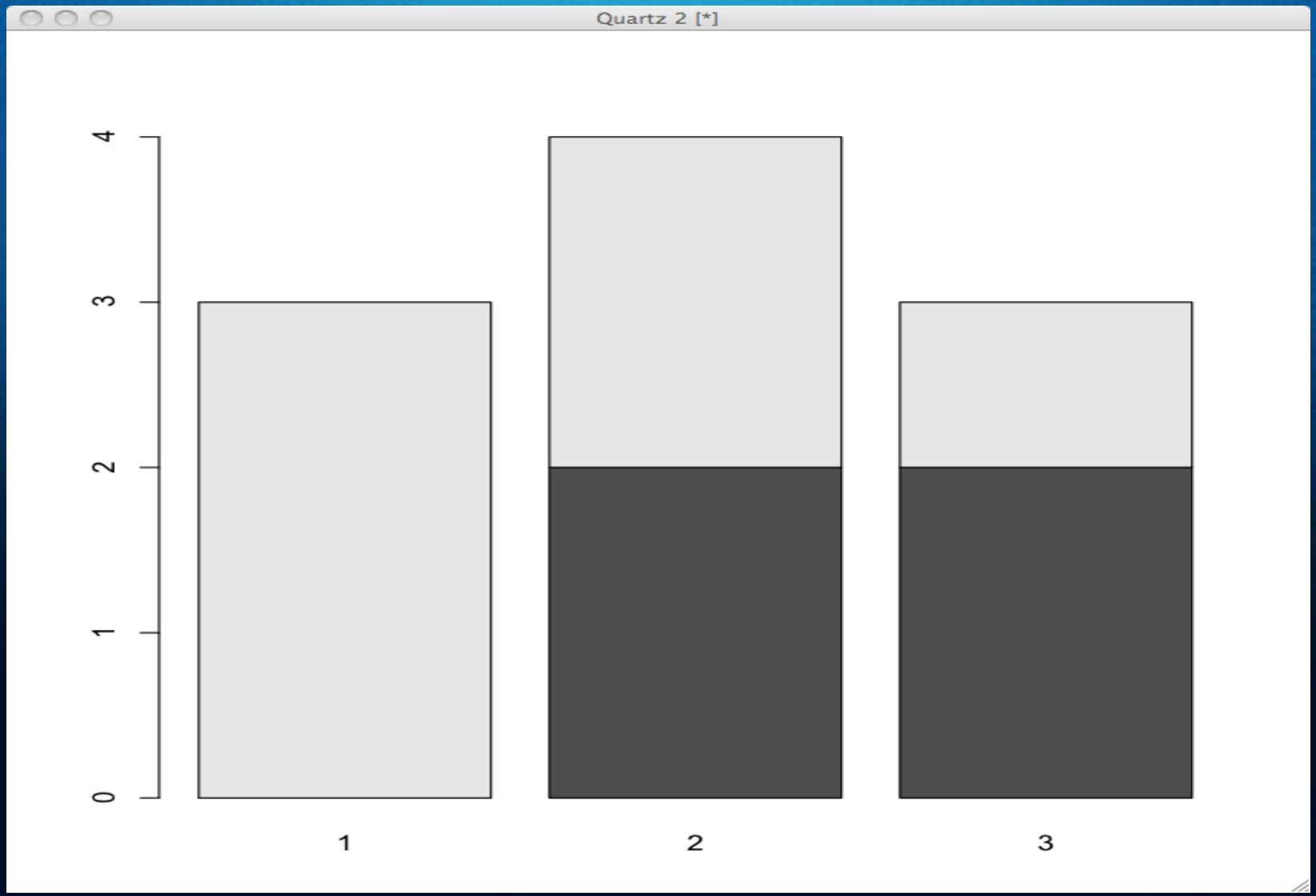
Multivariate Data

- You can generate a summary table of bivariate data

```
> smokes = c("Y","N","N","Y","N","Y","Y","Y","N","Y")
> amount = c(1,2,2,3,3,1,2,1,3,2)
table(smokes, amount)
        amount
smokes   1 2 3
      N  0 2 2
      Y  3 2 1
```

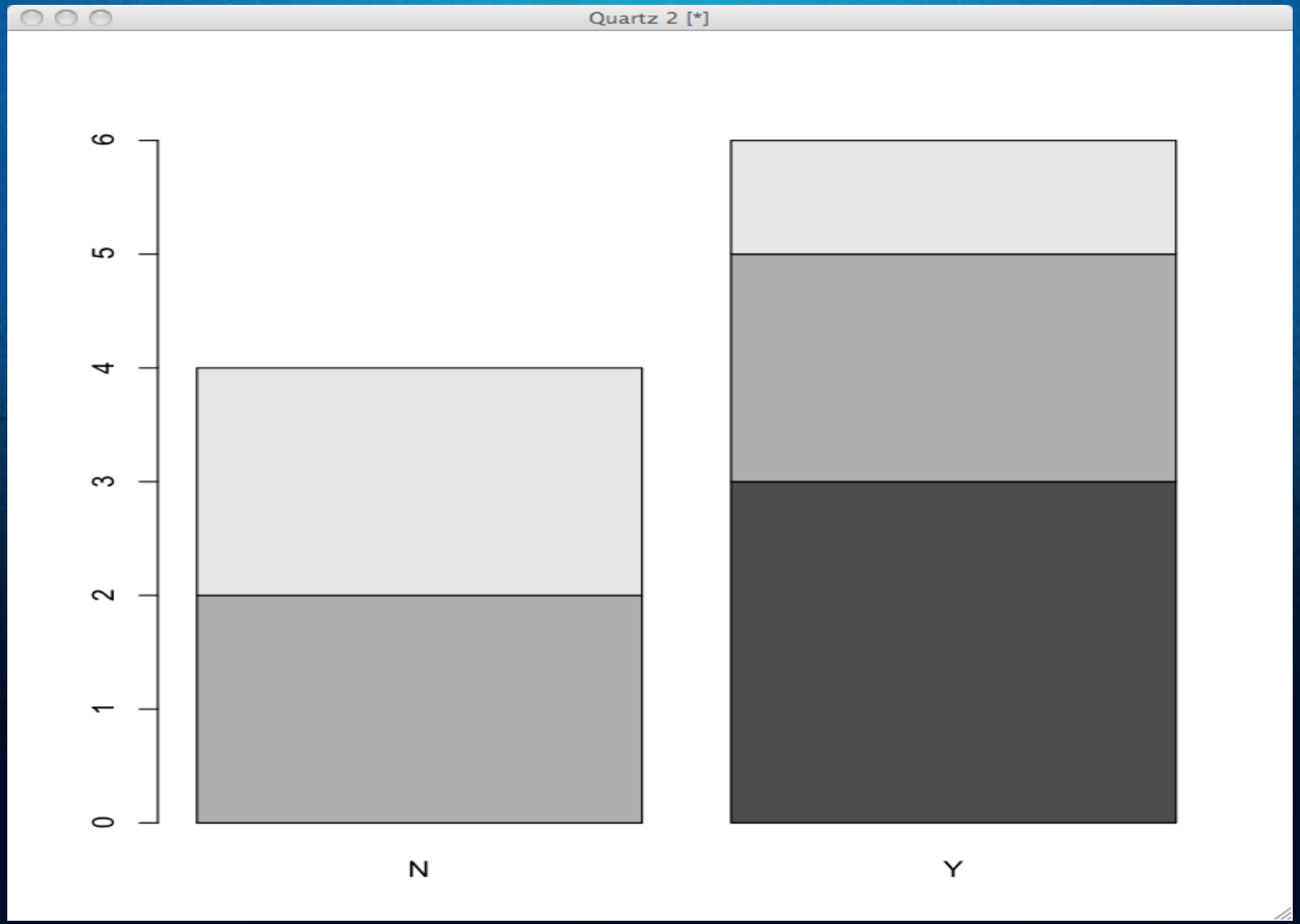
Plotting Bivariate Data

> barplot(table(smokes,amount))



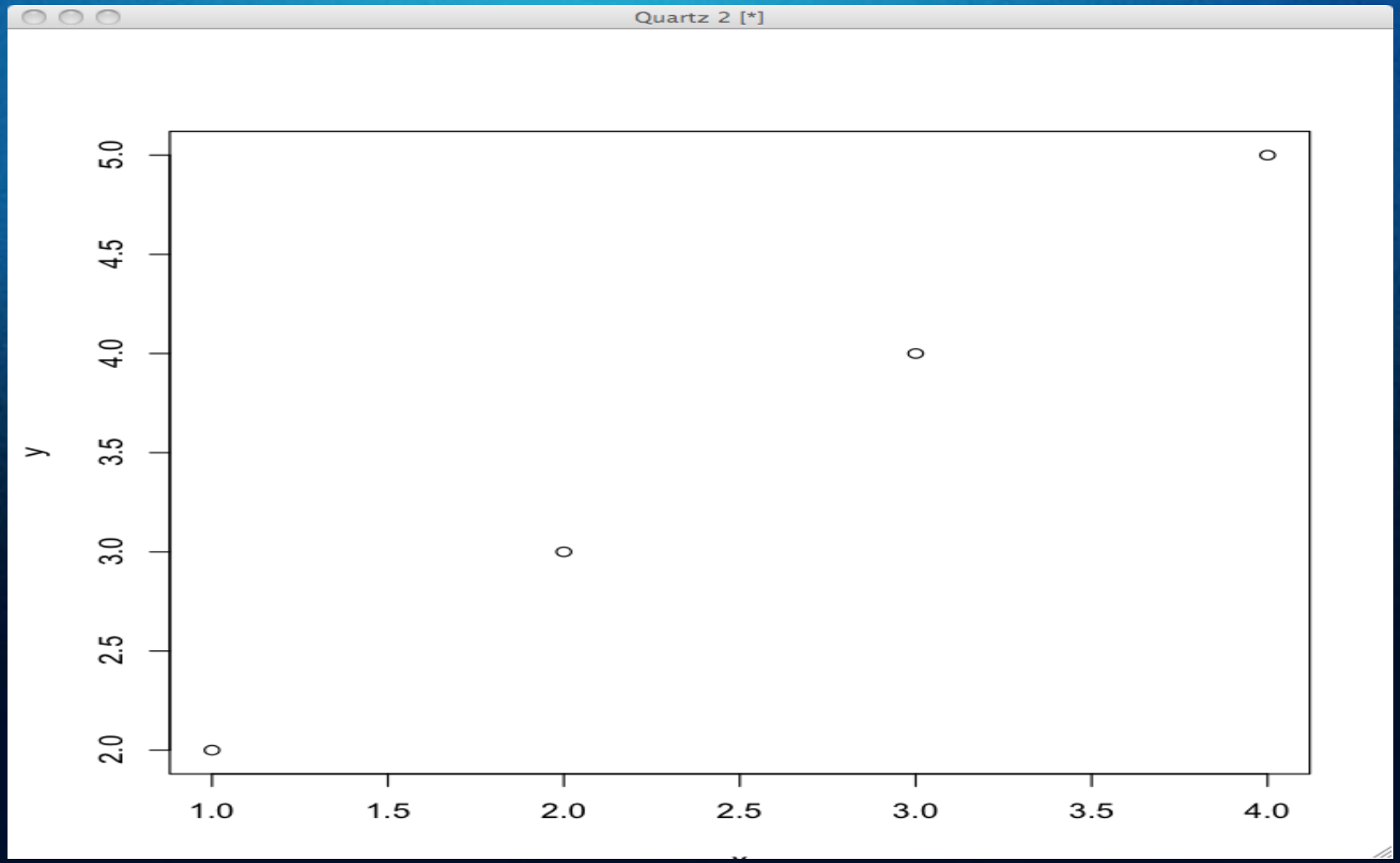
Plotting Bivariate Data - 2

> barplot(table(amount,smokes))



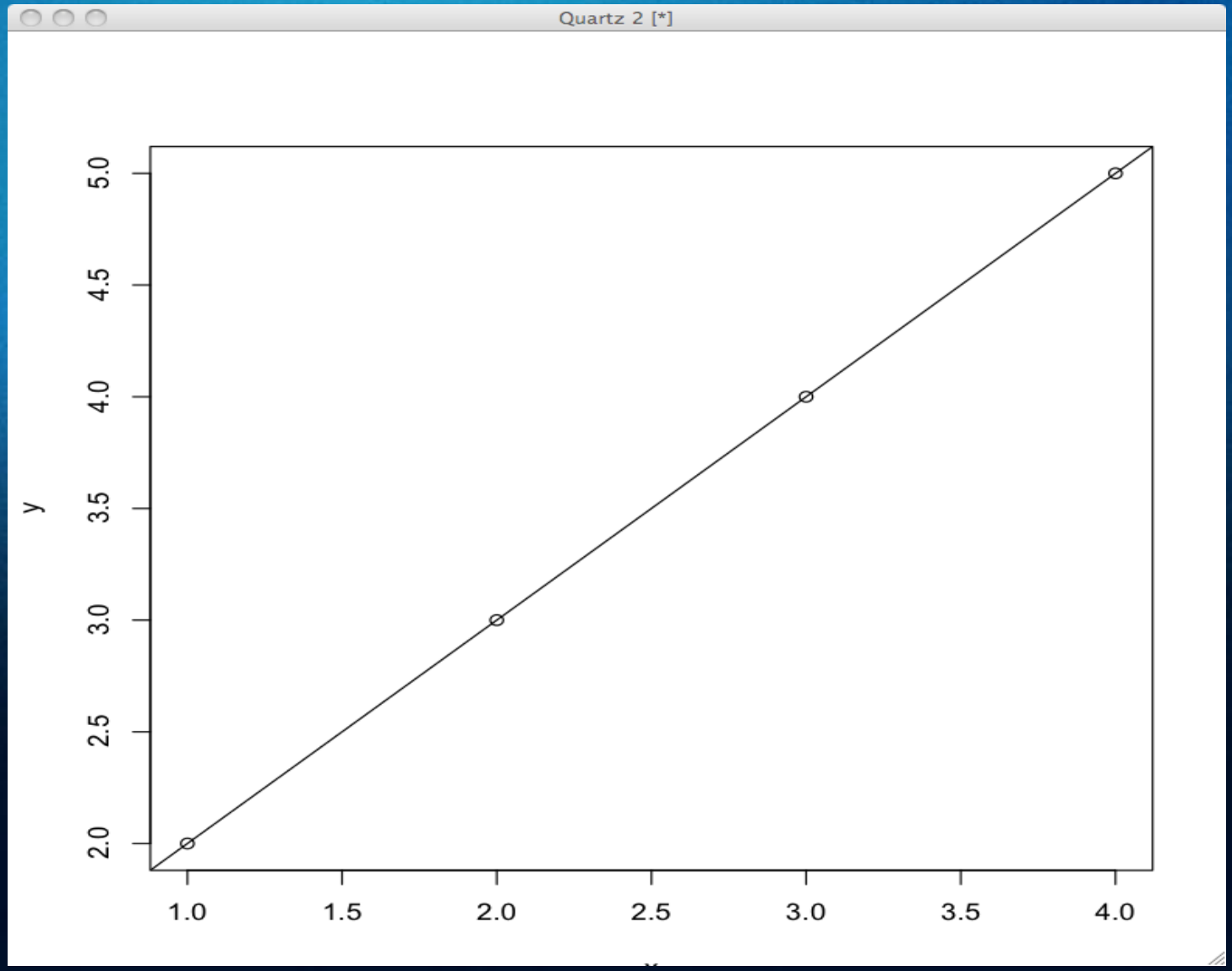
Plotting Bivariate Data - Scatterplots

```
> x = c(1,2,3,4)
> y = c(2,3,4,5)
> plot(x,y)
```



Least Squares Fit

```
> x = c(1,2,3,4)
> y = c(2,3,4,5)
> plot(x,y)
> abline(lm(y~x))
```



Least Squares Fit

```
> lm(y~x)
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)                    x
          1                    1
```

- You can access individual coefficients by storing it first

```
> lsq.res = lm(y~x)
> coef(lsq.res)
(Intercept)           x
          1           1
> coef(lsq.res)[1]
(Intercept)
          1
```

Testing a Hypothesis

- Let's say you ask 100 people a question and you get 42 yes answers
- Your hypothesis is that the true answer should be 50

```
> prop.test(42, 100, p=.5)

        1-sample proportions test with continuity correction

data:  42 out of 100, null probability 0.5
X-squared = 2.25, df = 1, p-value = 0.1336
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.3233236 0.5228954
sample estimates:
   p
0.42
```

Chi-Squared Test

- Lets say we want to check to see if a text is English
- The average letter distribution is
  E - 29, T - 21, N - 17, R - 17, O - 16
- In our text, we see the following
  E - 100, T - 110, N - 80, R - 55, O - 14

```
> x = c(100,110,80,55,14)
> probs = c(29,21,17,17,16)/100
> chisq.test(x, p=probs)

Chi-squared test for given probabilities

data:  x
X-squared = 55.3955, df = 4, p-values = 2.685e-11
```

- This shows that the text is not likely to be English

Outputting Graphics

- You probably want graphs and charts for publications
- Plots are sent to a graphics device - the default is the screen
- There are other graphics devices for various image file formats
  - png, jpeg, pdf, postscript, etc.

```
> png("filename.png")
> plot(x,y)
```

- This will save your plot to the file "filename.png". You can find other options by checking the help.

How do we install?

- There is a central repository of packages - CRAN
  - http://cran.r-project.org
- Use the command "install.packages" to automatically download, configure, compile and install packages
  - > install.packages("pkg1")
- You can install more than one at a time
  - > install.packages(c("pkg1","pkg2"))
- There are options to control installation
- The most useful is dependency checking
  - > install.packages("pkg1", dependencies = TRUE)
  - This will download and install any missing dependencies to pkg1

CRAN Task Views

There are pages which provide groups of packages useful for particular tasks

Clinical Trials	Clinical Trial Design, Monitoring, and Analysis
Experimental Design	Design of Experiments (DoE) and Analysis of Experimental Data
Genetics	Statistical Genetics
Medical Imaging	Medical Image Analysis
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods

Plotting and Graphics

- The base package provides a large number of statistical graphs
- You can build entirely new types of graphs
- There are several extra packages, including "grid" and "lattice"
- Based on a graphic display, you write your graph to this
- Divided into three basic groups
  - High-Level - plotting functions create a new plot on the graphics device
  - Low-Level - plotting functions to add more information to an existing plot
  - Interactive - graphics functions to allow you to interactively add information to, or extract information from, an existing plot

plot() function

- plot(x,y) or plot(xy) - x and y are vectors, plotting y against x. Second form is either a two-element list or two-column matrix
- plot(x) - if x is a time series, produces a time-series plot. If x is a vector, plots it against the vector index. If x is complex vector, plots imaginary versus real
- plot(f) or plot(f,y) - f is factor object, y is numeric vector. First form makes bar plot of f; second form makes boxplots of y for each level of f.

Multivariate Data

- pairs(x) - x is a numeric matrix or data frame, produces a pairwise scatterplot matrix of the variables defined by the columns of x. Every column of x is plotted against every other column and the resulting n(n-1) plots are arranged in a matrix
- coplot(a ~ b | c) - a and b are numeric vectors, c is a vector or factor object. Produces a number of scatterplots of a against b for given values of c.

Other high-level plots

- qqnorm(x), qqline(x), qqplot(x,y) - distribution-comparison plots. First form plots numeric vector x against expected Normal order scores. Second adds a line through distribution and data quartiles. Third plots quantiles of x against those of y to compare respective distributions.
- hist(x), hist(x,nclass=n), hist(x,breaks=b,...) - produces a histogram of vector x. If number of classes isn't good, can be set to n. Or, explicitly set breaks.
- image(x,y,z), contour(x,y,z), persp(x,y,z) - image plots valus of z as colors, contour plots contour lines, persp plots a 3D surface

Arguments for high-level plots

- add=TRUE - act like a low-level function and superimpose on the current plot
- axes=FALSE - suppress drawing axes, if you want to add your own with axis function
- log="x",log"y",log="xy" - causes x, y or both to be logarithmic
- type=
  - "p" - plot individual points (default)
  - "l" - plot lines
  - "b" - points connected by lines
  - "o" - points overlaid by lines
  - "h" - vertical lines from points to zero axis
  - "n" - no plotting at all. Axes are still drawn, and coordinate system set up, ready for low-level functions

Arguments - 2

- xlab=string, ylab=string - labels for x and y axes
- main=string - figure title, placed at top
- sub=string - subtitle, placed at bottom

Low-level functions

- points(x,y), lines(x,y) - add points or connected lines to the current plot
- text(x,y,labels) - add text at location x,y. labels is a vector, so labels[i] is plotted at (x[i],y[i])
- abline(a,b), abline(h=y), abline(v=x), abline(lm.obj) - adds a line of slope b and intercept a. h=y specifies y-coordinates for heights of horizontal lines. v=x specifies x-coordinates for vertical lines. lm. obj is coefficients from model-fitting functions
- polygon(x,y) - draws polygon based on ordered vertices (x,y)
- title(main,sub) - adds title "main" to top of graph, and optionally "sub" to bottom
- axis(side,...) -add an axis, where side is 1-4, counting clockwise from the bottom. Extra options for ticks, labels, etc.

Low-level functions - 2

- legend(x,y,legend,...) - adds a legend at location (x,y). Plotting characters, line styles, colors, etc. are identified with labels in "legend". At least one other argument, v, with corresponding values of the plotting unit must be given
  - legend(, fill=v) - colors for filled boxes
  - legend(, col=v) - colors for points or lines
  - legend(, lty=v) - line styles
  - legend(, lwd=v) - line widths
  - legend(, pch=v) - plotting characters

Mathematical Annotation

- Sometimes useful to add mathematical symbols and formulae
- You can do this by using expression rather than a text string in text, mtext, axis or title
- Example: adding the Binomial probability formula

```
> text(x, y, expression(paste(bgroup("(", atop(n,x), ")"), p^x, q^{n-x})))
```

- More information available with
  - help(plotmath)
  - example(plotmath)
  - demo(plotmath)

Interacting with Graphs

- locator(n, type) - user can select point coordinates with mouse left-clicks. Will keep collecting until n points have been selected, or another mouse button is clicked. type allows for plotting at those points; the default is no plotting. Very useful when trying to figure out where to put text, etc.
- identify(x, y, labels) - allows users to pull up more information about point within the x,y region.

```
> plot(x,y)
> identify(x,y)
```

This lets a user click on the plot, and get the index of the nearest point.

Graphics Parameters

The par() function

- par() - returns the full list of parameters for the current graphics device
- par(c("col", "lty")) - with a character vector argument, returns only the named parameters as a list
- par(col=4, lty=2) - with named arguments, sets the values of the named graphics parameters, and returns the original values of the parameters as a list

Parameters - Graphical Elements

- pch="+" - use this character when plotting points
- lty=2 - line type. 0 is invisible. 1 is solid. 2 and up is dotted and/or dashed
- lwd=2 - line widths
- col=2 - color used for points, lines, etc. Number in the color palette (see ?palette)
- col.axis, col.lab, col.main, col.sub - color for axis annotation, x and y labels, main title, subtitle
- font=2 - 1 is plain, 2 is bold, 3 is italic, 4 is bold italic, 5 is a symbol font
- font.axis, font.lab, font.main, font.sub

Parameters - axes and tick marks

- lab=c(5,7,12) - first two numbers are the number of tick intervals on x and y axes. Third number is length of axis labels, including decimal point
- las=1 - orientation of axis labels. 0 is parallel to axis, 1 is horizontal, 2 is perpendicular to axis
- mgp=c(3,1,0) - positions for axis components. First is distance from axis label to axis position, second is distance to tick labels, and third is distance from axis position to axis line (usually 0). Positive numbers measure outside plot region, negative numbers inside
- tck=0.01 - length of tick marks, as a fraction of the size of the plotting region

Multiple Figures

You can create an n by m array of figures on a single page

- mfcol=c(3,2) mfrow=c(2,4) - set size of multiple figure array. First value is rows, second is columns. mfcol fills by column, mfrows fills by row.
- mfg=c(2,2,3,2) - position of current figure. First two numbers is row and column of figure. Next two numbers is size of array.
- fig=c(4,9,1,4)/10 - position of current figure on page. Values are positions of the left, right, bottom and top edges respectively, as a percentage of page measured from bottom left corner. Good for arbitrary placed figures.

Device Drivers

- You need to start a device driver before you start drawing
- You can get a list of all supported ones with help(Devices)
- Common ones are
  - X11() - X11 window system
  - windows() - Windows
  - quartz() - Mac OS X
  - postscript() - printing on postscript printers, or to a PS file
  - pdf() - drawing to a PDF file
  - png() - drawing to a PNG file
  - jpeg() - drawing to a JPEG file
- When you are done, shutdown the device with "dev.off()"

PostScript files for typeset documents

- By passing in a filename, will send it to that file
- Plot will be in landscape, unless horizontal=FALSE
- Can control size of graphic with width and height
  - > postscript("file.ps", horizontal=FALSE, height=5)
- Encapsulated PostScript more useful when used to be included in other documents
- Use onefile=FALSE
  - > postscript("file.eps", horizontal=FALSE, onefile=FALSE, height=8, width=6)

Multiple Devices

- Every time you start a device R opens a new device, which becomes current
- dev.list() - return number and name of all active devices
- dev.next() dev.prev() - returns number and name of next or previous device
- dev.set(which=k) - set current device to be the k^{th} one
- dev.off(k) - shutdown k^{th} device
- dev.copy(device, ..., which=k) - make a copy of device k, using function "device" (e.g, postscript) and any options for "device" given by ...
- graphics.off() - terminate all graphics on the list, except the null device