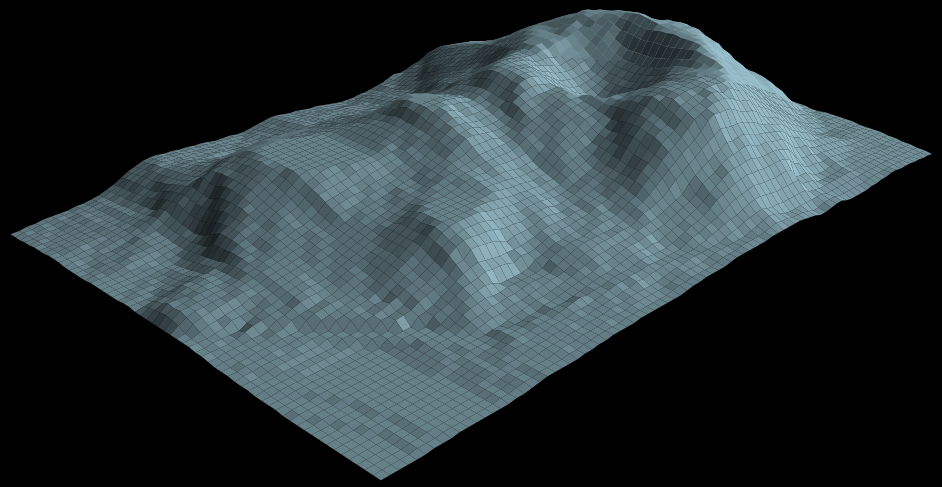
Introducing R

JEFF GILL
Department of Political Science
Division of Biostatistics
Department of Surgery (Public Health Sciences)
Washington University, St. Louis

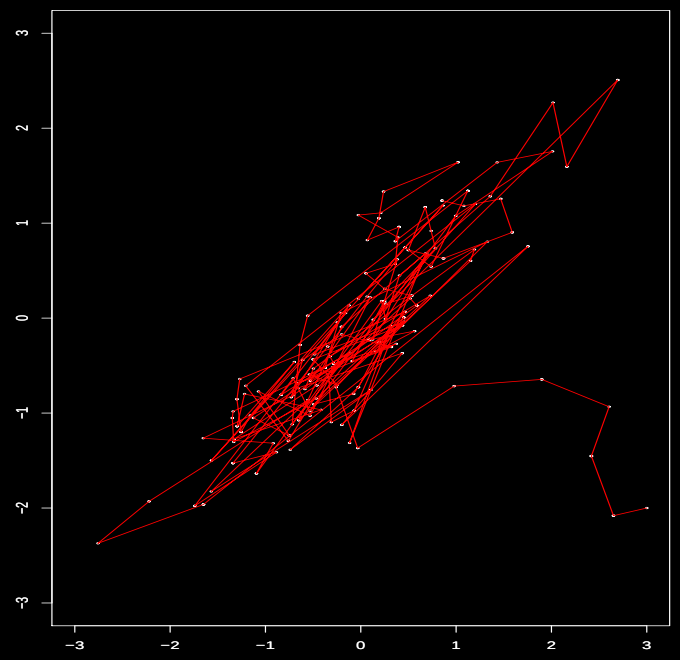


Preliminaries

- Thank R Its Friday (TRIF) is supported by the Washington University TREC Center (`www.obesity-cancer.wustl.edu`), School of Medicine.
- Meeting times and topics (3-4PM):
  - September 27: Downloading R, using basic commands, and loading data
  - October 25: Basic data analysis and introduction to graphics
  - November 22: Running regression models in R
  - January 31, 2014: Analysis of Variance
  - February 28, 2014: Logistic Regression
  - March 28, 2014: Principle Components Analysis
  - April 25, 2014: Survival Analysis
  - May 30, 2014: Nonparametric Data Analysis
- The slides for today are available at `http://jgill.wustl.edu/slides/trif1.pdf`.

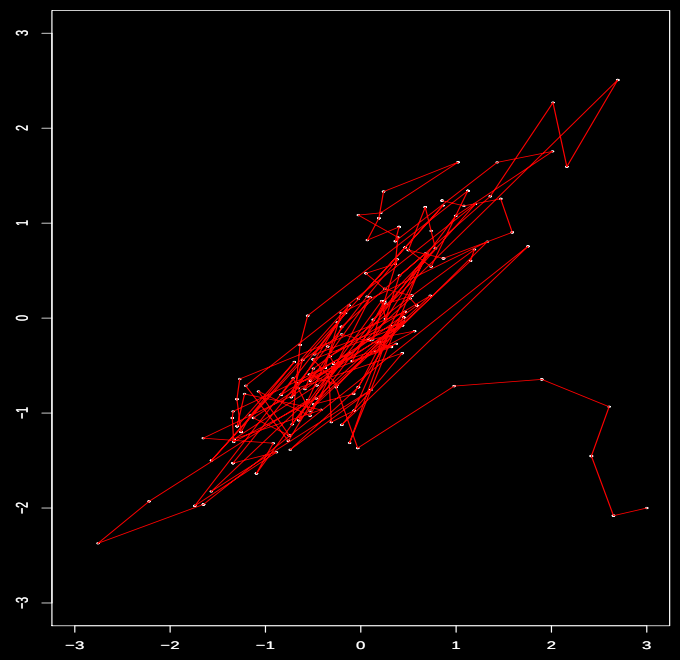
Exactly What the *&#$@*$% Is This R Thing?

- A *language* and *environment* for statistical computing and graphics.



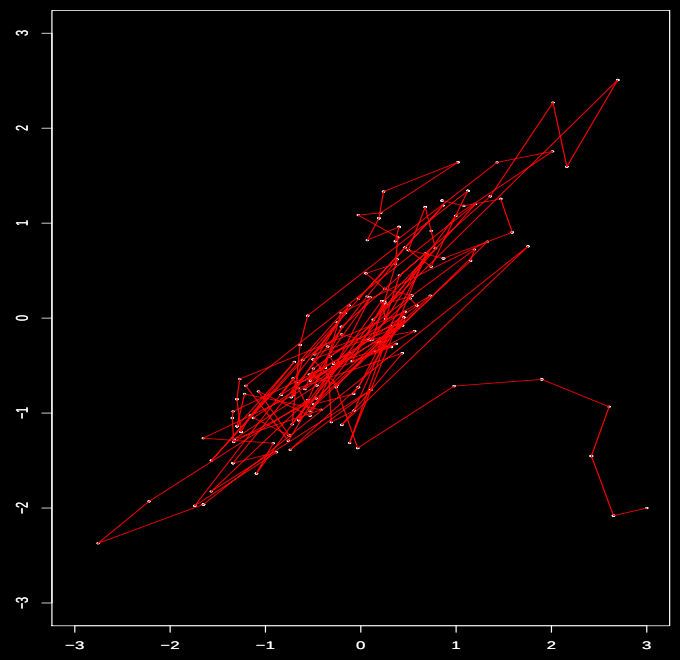
Exactly What the *&#$@*$% Is This R Thing?

- A *language* and *environment* for statistical computing and graphics.
- The most powerful and fully featured statistical environment *on the planet*.



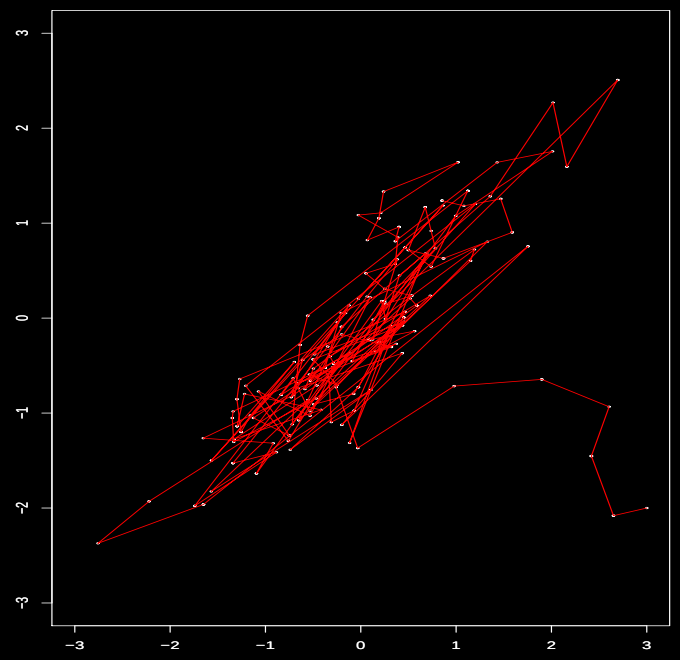
Exactly What the *&#$@*$% Is This R Thing?

- A *language* and *environment* for statistical computing and graphics.
- The most powerful and fully featured statistical environment *on the planet*.
- A GNU project based on S which was developed at Bell Laboratories by John Chambers and colleagues.



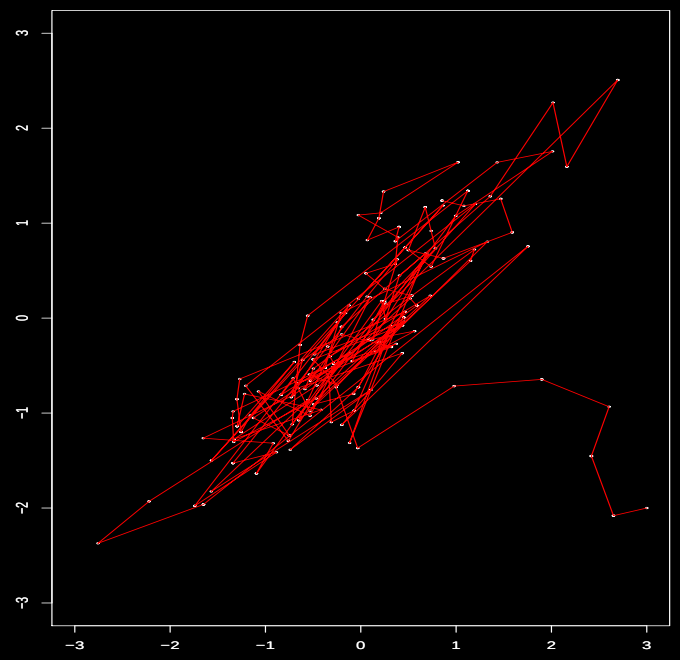
Exactly What the *&#$@*$% Is This R Thing?

- A *language* and *environment* for statistical computing and graphics.
- The most powerful and fully featured statistical environment *on the planet*.
- A GNU project based on S which was developed at Bell Laboratories by John Chambers and colleagues.
- Highly extensible through researcher-developed packages and self-developed functions.



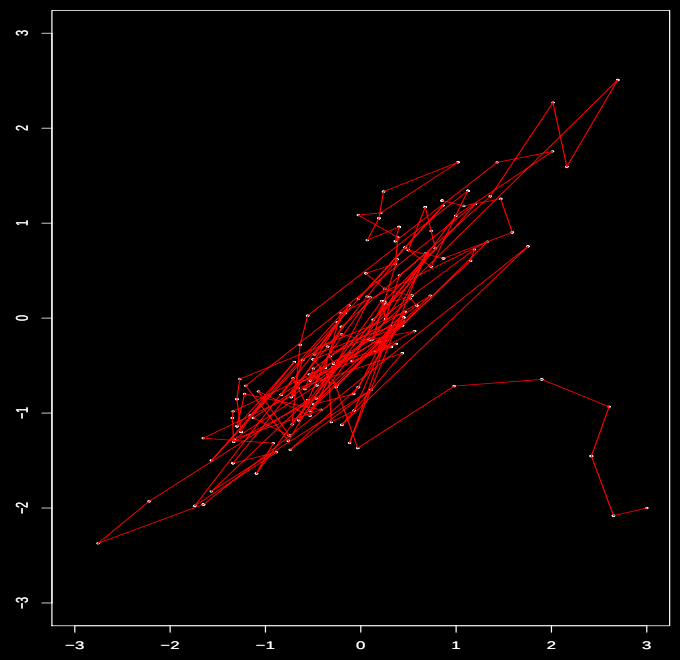
Exactly What the *&#$@*$% Is This R Thing?

- A *language* and *environment* for statistical computing and graphics.
- The most powerful and fully featured statistical environment *on the planet*.
- A GNU project based on S which was developed at Bell Laboratories by John Chambers and colleagues.
- Highly extensible through researcher-developed packages and self-developed functions.
- The primary statistical computing tool for academic research statisticians.



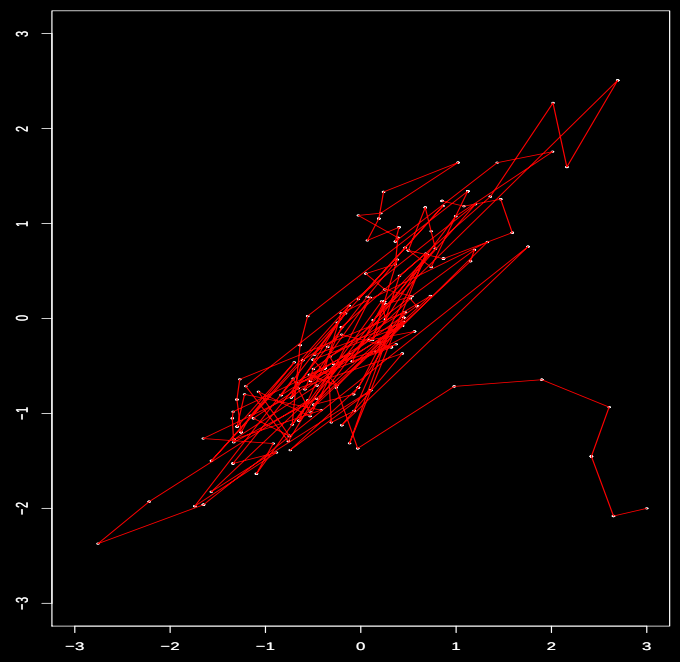
Exactly What the *&#$@*$% Is This R Thing?

- A *language* and *environment* for statistical computing and graphics.
- The most powerful and fully featured statistical environment *on the planet*.
- A GNU project based on S which was developed at Bell Laboratories by John Chambers and colleagues.
- Highly extensible through researcher-developed packages and self-developed functions.
- The primary statistical computing tool for academic research statisticians.
- Available for all operating systems.



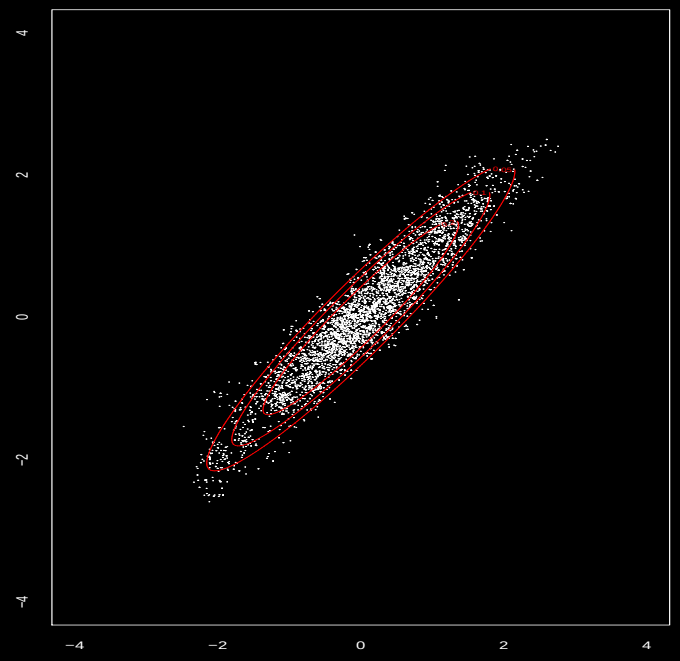
Exactly What the *&#$@*$% Is This R Thing?

- A *language* and *environment* for statistical computing and graphics.
- The most powerful and fully featured statistical environment *on the planet*.
- A GNU project based on S which was developed at Bell Laboratories by John Chambers and colleagues.
- Highly extensible through researcher-developed packages and self-developed functions.
- The primary statistical computing tool for academic research statisticians.
- Available for all operating systems.
- Free!



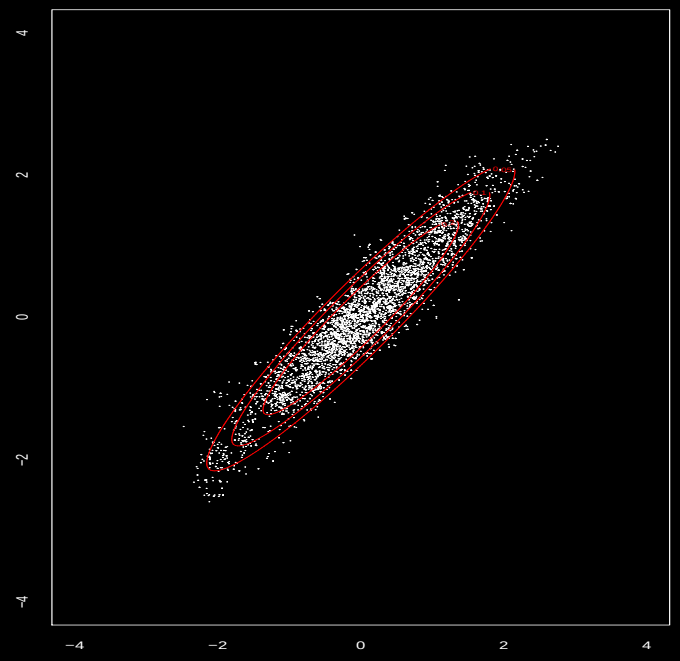
What Do You Get?

- An effective data handling and storage facility.



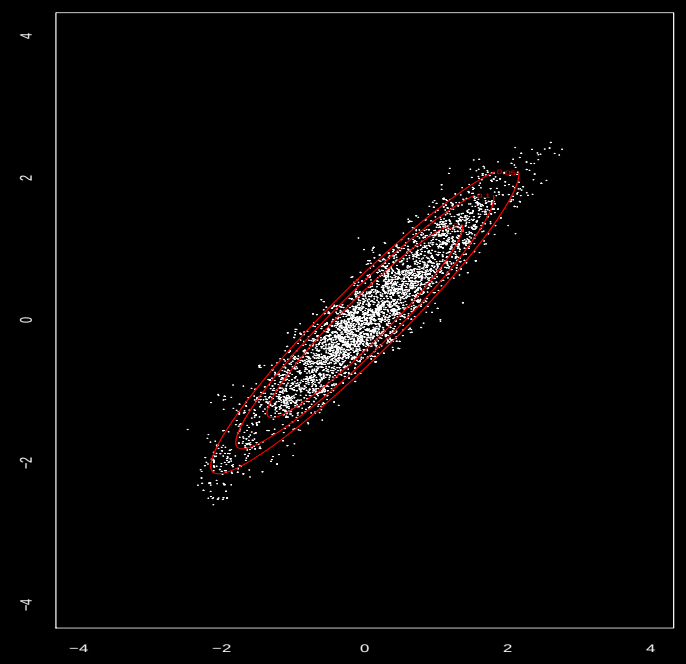
What Do You Get?

- An effective data handling and storage facility.
- A suite of operators for calculations on arrays, in particular matrices.



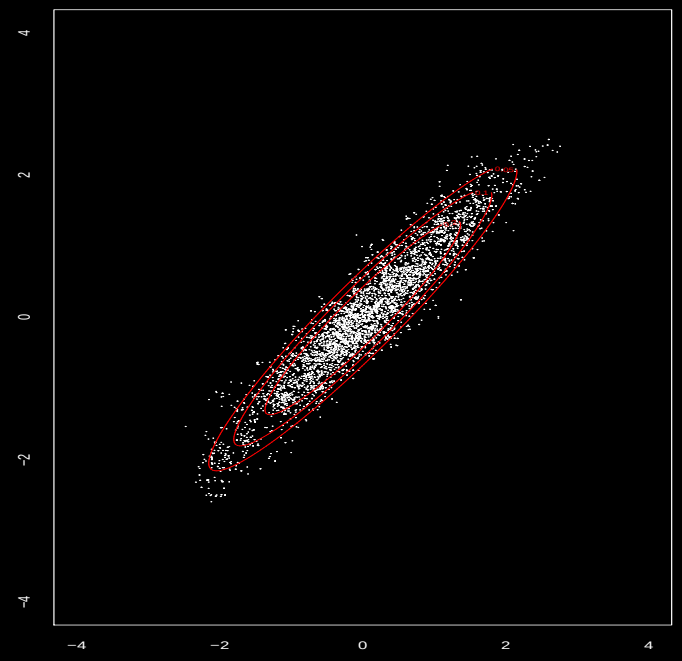
What Do You Get?

- An effective data handling and storage facility.
- A suite of operators for calculations on arrays, in particular matrices.
- A large, coherent, integrated collection of intermediate tools for data analysis.



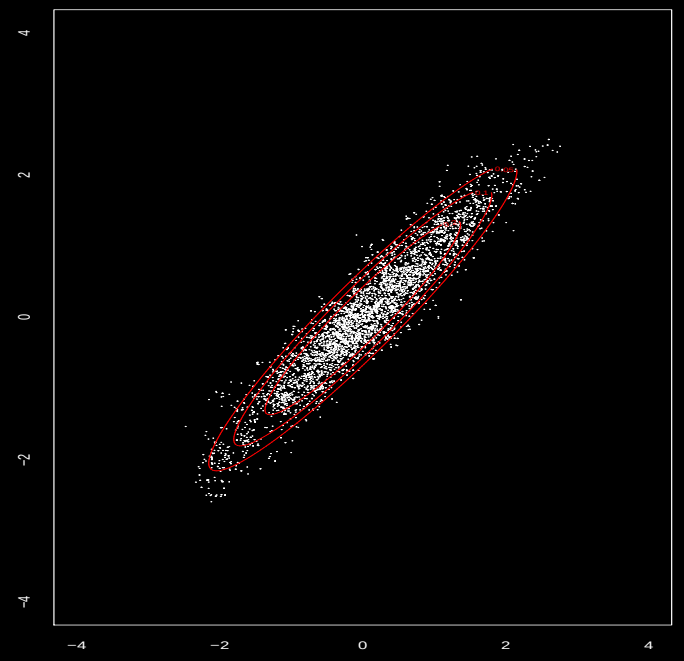
What Do You Get?

- An effective data handling and storage facility.
- A suite of operators for calculations on arrays, in particular matrices.
- A large, coherent, integrated collection of intermediate tools for data analysis.
- Graphical facilities for data analysis and display either on-screen or on hard-copy.



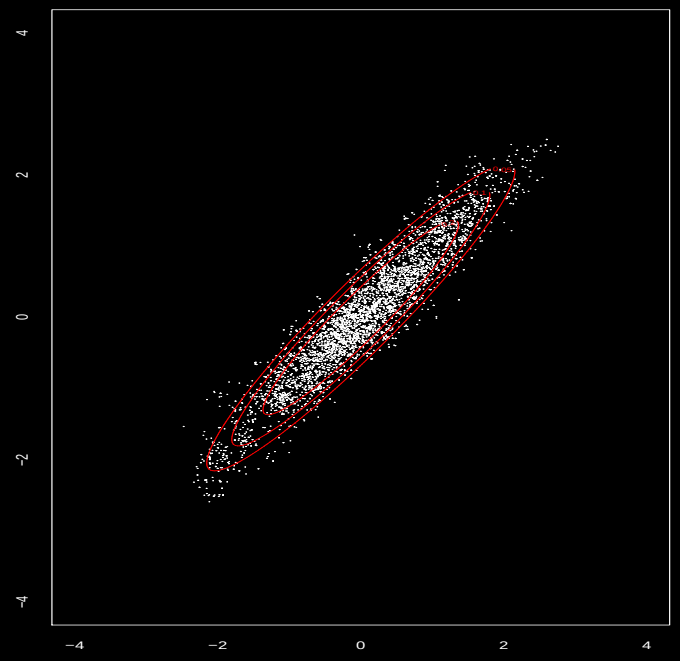
What Do You Get?

- An effective data handling and storage facility.
- A suite of operators for calculations on arrays, in particular matrices.
- A large, coherent, integrated collection of intermediate tools for data analysis.
- Graphical facilities for data analysis and display either on-screen or on hard-copy.
- A well-developed, simple and effective programming language.



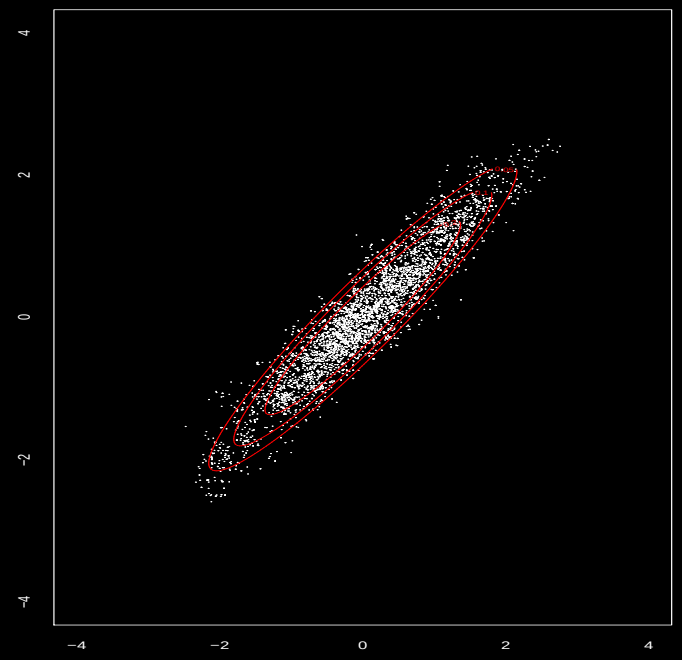
What Do You Get?

- An effective data handling and storage facility.
- A suite of operators for calculations on arrays, in particular matrices.
- A large, coherent, integrated collection of intermediate tools for data analysis.
- Graphical facilities for data analysis and display either on-screen or on hard-copy.
- A well-developed, simple and effective programming language.
- Linkages to `C`, `Fortran`, `SQL`, and other languages.



What Do You Get?

- An effective data handling and storage facility.
- A suite of operators for calculations on arrays, in particular matrices.
- A large, coherent, integrated collection of intermediate tools for data analysis.
- Graphical facilities for data analysis and display either on-screen or on hard-copy.
- A well-developed, simple and effective programming language.
- Linkages to `C`, `Fortran`, `SQL`, and other languages.
- A huge user/developer community of mostly academics.



Major Characteristics

- Object-Oriented: everything in R is an "object," meaning that data structures, functions, created objects are all visible and can be manipulated by name.
- Visible Environment: you see the list of objects you create or download in your R environment window.
- Visible Objects: what is inside the objects is revealed when you type the name of the object.
- Functions: there are millions of functions in R (really!), and you can create your own as well.
- File Coherence: when you quit R you will be asked if you want to save your environment (you generally do), and if you say no every object created will be gone.

Getting R

The Comprehensive R Archive Network

http://cran.wustl.edu/

menarche

Washington ...tem Signon

Political Science Job Rumors | The Comprehensive R Archive Ne...

The Comprehensive R Archive Network

Frequently used pages

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Linux
- MacOS X
- Windows

Source Code for all Platforms

Windows and Mac users most likely want the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- **The latest release** (2010-05-31): R-2.11.1.tar.gz (read what's new in the latest version).
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

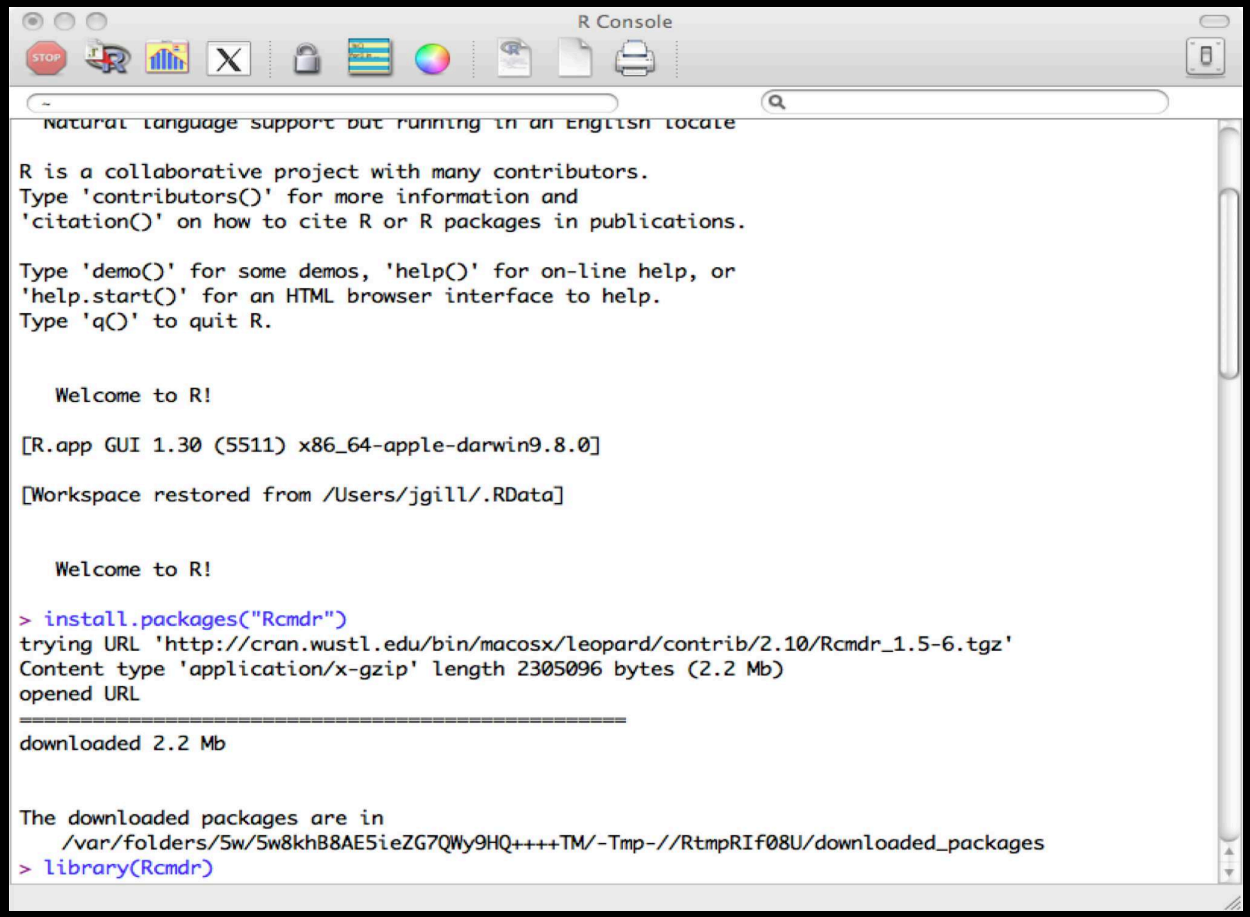
Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Getting R


Install R 3.0.1 for Mac OS X 10.6 or higher (Snow Leopard build)
Welcome to the R 3.0.1 for Mac OS X 10.6 or higher (Snow Leo
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
This installer will guide you through the steps necessary to setup R 3.0.1 (2013-05-16) -- "Good Sport" for Mac OS X 10.6 (Snow Leopard) or higher on your machine.
Go Back
Continue

Starting R


R Console
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.
Welcome to R!
[R.app GUI 1.30 (5511) x86_64-apple-darwin9.8.0]
[Workspace restored from /Users/jgill/.RData]
Welcome to R!
> install.packages("Rcmdr")
trying URL 'http://cran.wustl.edu/bin/macosx/leopard/contrib/2.10/Rcmdr_1.5-6.tgz'
Content type 'application/x-gzip' length 2305096 bytes (2.2 Mb)
opened URL
==========
downloaded 2.2 Mb
The downloaded packages are in
/var/folders/5w/5w8khB8AE5ieZG7QWy9HQ++++TM/-Tmp-//RtmpRIf08U/downloaded_packages
> library(Rcmdr)

Getting Help By Yourself

- Help pages and FAQ exist on CRAN (`http://cran.wustl.edu`).
- Local ways to get help in your `R` environment:

```
help(ls);      ?ls
help.search()
args()
View()
browseEnv()
search()
date()
summary()
demo()
```

Basic Functions That Are Used All the Time

```
ls();    objects()
ls(pattern="asap.mice")
rm();    remove()
find("mice")
library()
summary()
date()
```

Basic Arithmetic Operations

```
*    # Multiply
+    # Add
-    # Subtract
/    # Divide
^    # Exponentiation
%%   # Remainder or modulo operator
%*%  # Matrix multiplication operator
%/%  # Integer divide
%c%  # crossproduct
%o%  # Outer Product
```

Relational Operations

```
!=      # Not-equal-to
<       # Less-than
<=      # Less-than-or-equal-to
>       # Greater-than
>=      # Greater-than-or-equal-to
==      # Equal for testing
```

Data Types

- numeric vs. character: `3.14149` versus `"PI"`
- numeric qualities: complex, double
- `NULL`
- logical: `TRUE`, `FALSE`
- scalar, integer
- vector, matrix, array
- data.frame
- list

Data Related Commands

`is.list(x.y.fit); names(x.y.fit)`	`as.list()`
`is.data.frame(x.y.fit)`	`as.data.frame()`
`is.vector(chile[1:10,2])`	`as.vector()`
`is.matrix(summary(x.y.fit)$cov.unscaled)`	`as.matrix()`
`is.numeric(X)`	`as.numeric()`
`is.integer(X)`	`as.integer()`
`is.real(X)`	`as.real()`
`is.character(Y)`	`as.character()`
`is.function(lm)`	`as.function()`

Assignment

- You can create/assign "objects" according to:

```
X = 3/2
Y = 2 * pi
z <- X + Y
z -> w
( Result.With.Big.Name <- 9/4 + 3^2 - sqrt(2) - pi^(9-1/2) )
[1] -16808
```

- Be aware of case, don't use "_", and pay attention to order of operations.
- Try not to use names of built-in functions (`c`).
- Objects remain until removed:

```
rm(X,Y,z,w,Result.With.Big.Name)
```

Vector Assignment

- Examples:

```
people.ages <- c(13,54,33,19,44,62)
people.names <- c("Tom","Bob","Mary","Fred","Sally","Joe")
names(people.ages) <- names(people.names) <- c("Person 1","Person 2",
                      "Person 3","Person 4","Person 5","Person 6")

people.ages
Person 1 Person 2 Person 3 Person 4 Person 5 Person 6
      13       54       33       19       44       62
people.names
Person 1 Person 2 Person 3 Person 4 Person 5 Person 6
   "Tom"    "Bob"   "Mary"   "Fred"  "Sally"    "Joe"
```

- Note that

```
names
```

is a *function*.

Arithmetic Functions on Vectors

- Basic functions include:

```
sum(people.ages)
[1] 225
min(people.ages)
[1] 13
max(people.ages)
[1] 62
range(people.ages)
[1] 13 62
diff(people.ages)
[1]  41 -21 -14  25  18
cumsum(people.ages)
[1]  13  67 100 119 163 225
people.ages - mean(people.ages)
[1] -24.5  16.5  -4.5 -18.5   6.5  24.5
```

Another Example

- tumor counts:

```
tumor <- c(74, 122, 235, 111, 292, 111, 211, 133, 156, 79)
mean(tumor)
[1] 152.4
var(tumor)
[1] 5113.4
sd(tumor)
[1] 71.508
sqrt( sum( (tumor - mean(tumor))^2 /(length(tumor)-1)))
[1] 71.508
```

Sequences

```
num.vec <- 1:10
num.vec[1:5]
 [1] 1 2 3 4 5
num.vec[num.vec>5]
 [1]  6  7  8  9 10
num.vec < 3
 [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
length(num.vec)
[1] 10
all(num.vec < 10)
[1] FALSE
which(num.vec < 6)
[1] 1 2 3 4 5
rev(1:10)
[1] 10  9  8  7  6  5  4  3  2  1
rep(4,10)
[1] 4 4 4 4 4 4 4 4 4 4
seq(1,11,by=2)
[1]  1  3  5  7  9 11
```

Indices

```
x <- c(1,3,56,6,7,4,3,5)
length(x)
[1] 8
x[1] <- 2
x[2]
[1] 3
x[3:5]
[1] 56  6  7
x[-4]
[1]  2  3 56  7  4  3  5
x[c(1,4,6)]
[1] 2 6 4
names(x) <- c("first","second","third","fourth","fifth","sixth","seventh","eighth")
x["third"]
third
   56
x[4] <- 11
x
     first  second   third  fourth   fifth   sixth seventh  eighth
         2       3      56      11       7       4       3       5
```

Testing

```
x <- c(1,3,56,6,7,4,3,5)
x < 7
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE
which(x < 7)
[1] 1 2 4 6 7 8
x >= 8
[1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
x %in% c(1,4,6)
[1]  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
```

Sequence and Repetition

- Sequences are created in two different ways:

```
# seq(from, to, by, length, along)
seq(1,10,length=5)
[1]  1.00  3.25  5.50  7.75 10.00
seq(0,100,along=c(1,2,3,4,5))
[1]   0  25  50  75 100
-3:3
[1] -3 -2 -1  0  1  2  3
2.5:6.5
[1] 2.5 3.5 4.5 5.5 6.5
```

- Repetition has one general form, `rep(x, times, length)`, for example:

```
rep(1,length=8)
 [1] 1 1 1 1 1 1 1 1
rep(c(1,2,3),4)
 [1] 1 2 3 1 2 3 1 2 3 1 2 3
rep(1:3, each = 5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```

Missing Data

```
x <- c(1,3,65,NA,3,NA)
is.na(x)
[1] FALSE FALSE FALSE  TRUE FALSE  TRUE
sum(is.na(x))
[1] 2
mean(x)
[1] NA
mean(x)         # mean(x,na.fail)
[1] NA
mean(x,na.rm=TRUE)
[1] 18
```

Matrices

- matrices can be created in difference ways:

```
( X <- matrix(c(1,2,3,4,5,6,7,8,9),ncol=3) )
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
x <- c(1,3,56,6,7,4,3,5)
y <- c(21,3,0,-3,32,10,13,11)
( Y <- rbind(x,y) )
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
x    1    3   56    6    7    4    3    5
y   21    3    0   -3   32   10   13   11
( Z <- as.matrix(cbind(rep(1,length=5),rnorm(5))) )
     [,1]       [,2]
[1,]    1  0.2071463
[2,]    1 -0.1237397
[3,]    1  0.7773869
[4,]    1  1.5469857
[5,]    1 -0.2990431
```

Matrices

- We reference matrix values like we did with vectors where two dimensions are now required:

```
X[2,3]
[1] 8
X[2,]
[1] 2 5 8
X[,3]
[1] 7 8 9
X[1:2,3]
[1] 7 8
X[1:2,2:3]
     [,1] [,2]
[1,]    4    7
[2,]    5    8
```

Matrices

- There are the standard matrix operations available:

```
( ZZ <- t(Z) %*% Z )
[1,] 5.000000 2.108736
[2,] 2.108736 3.145143

solve(ZZ)
           [,1]       [,2]
[1,]  0.2788508 -0.1869621
[2,] -0.1869621  0.4433038

chol(ZZ)
         [,1]      [,2]
[1,] 2.236068 0.9430555
[2,] 0.000000 1.5019286

diag(ZZ)
[1] 5.000000 3.145143
```

Matrices

- More matrix operations:

```
det(ZZ)
[1] 11.27895

eigen(ZZ)
$values
[1] 6.376241 1.768902
$vectors
           [,1]       [,2]
[1,] -0.8374328  0.5465403
[2,] -0.5465403 -0.8374328

chol(ZZ)
         [,1]      [,2]
[1,] 2.236068 0.9430555
[2,] 0.000000 1.5019286
```

Ways To Create Matrices

```
matrix
cbind
rbind
t(Z) %*% Z
expand.grid(c(1,2),c(1,2),c(1,2))
  Var1 Var2 Var3
1    1    1    1
2    2    1    1
3    1    2    1
4    2    2    1
5    1    1    2
6    2    1    2
7    1    2    2
8    2    2    2
```

Data Frames

- Data frames are like matrices except that they include different data types not just numeric data: factors and characters.

```
freq <- c(9, 32, 4, 1, 8, 4, 3, 1, 40, 6, 2, 0, 1, 0, 8, 1, 9, 14,
          9, 6, 9, 41, 2, 5, 5, 7, 23, 3, 1, 7, 31, 24, 99, 2, 6, 33)
( psych.df <- data.frame(freq,expand.grid(anxiety=1:3,behavioral=1:2,
                                          depression=1:3,sex=1:2)) )
   freq anxiety behavioral depression sex
1     9       1          1          1   1
2    32       2          1          1   1
3     4       3          1          1   1
4     1       1          2          1   1
:
33   99       3          1          3   2
34    2       1          2          3   2
35    6       2          2          3   2
36   33       3          2          3   2
```

Data Frames

- But these are not factors yet, so do the following:

```
psych.df$anxiety <- factor(psych.df$anxiety, labels=c("Low","Medium","High"),
        ordered=TRUE)
psych.df$behavioral <- factor(psych.df$behavioral, labels=c("Present","Absent"))
psych.df$sex <- factor(psych.df$sex, labels=c("Male","Female"))
psych.df$depression <- factor(psych.df$depression,
        labels=c("Absent","Mild","Severe"), ordered=TRUE)
psych.df
   freq anxiety behavioral depression    sex
 1    9     Low    Present     Absent   Male
 2   32  Medium    Present     Absent   Male
 3    4    High    Present     Absent   Male
 4    1     Low     Absent     Absent   Male
:
33   99    High    Present     Severe Female
34    2     Low     Absent     Severe Female
35    6  Medium     Absent     Severe Female
36   33    High     Absent     Severe Female
```

Loading Data

- There are several different ways of loading data, depending on what format it comes in.
- For manually inputing (small) datasets, create the first line of the data with an R assignment command:

  ```
  x <- c(1,2,3)
  ```

  then you can enter more with:

  ```
  edit(x)
  ```

  which gives a popup window with: `c(1,2,3)` and room to edit/extend, or

  ```
  data.entry(x)
  ```

  which gives a different popup window with the data down a column and room to edit/extend.
- If you want bring code into R as if were typed in the environment window use `source("filename")`, which is the opposite of `sink("filename")`, `sink()`.

Loading Data

- The most common is `read.table`, which has many options, the most common are:

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".", row.names,
           col.names, na.strings = "NA", strip.white = FALSE,
           blank.lines.skip = TRUE, comment.char = "#")
```

  Also, `file` here can be a file on your hard-drive or a file from a URL.

- Here is an example of `read.table`:

```
mouse <- read.table("/Users/jgill/Grant.TREC/CompiledMouseData.dat",header=TRUE)
mouse[1,]
Mouse_Number Age_when_used Body_weight UGS_weight Prostate_Weight
          56           119          NA         NA              NA
Number_male_pups_in_cage Cage_Number Diet_Treatment Age_parents_at_birth
                       4          31              0                  97
Time_Parents_on_diet_before_birth Total_Acini Normal
                               69         170    165
Number_Hyperproliferative
                        5
```

Loading Data

- `read.table` is fussy about having the exact same number of items on each line, so to check this in advance use:

```
count.fields("/Users/jgill/Grant.TREC/CompiledMouseData.dat")
[1] 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
```

- `read.table` actually uses another function called `scan` and formats it according to the `read.table` parameters specified.

- `scan` is used directly to bring in data that is not structured since it treats the data file as one long vector:

```
scan("Class.Multilevel/CODAchain1.txt")[1:10]
Read 400000 items
 [1] 1.0000 1.3832 2.0000 1.6580 3.0000 1.8051 4.0000 1.8193 5.0000 1.9530
```

- The command `read.csv()` is really `read.table` with the option `sep=‘‘,’’`.

Loading Data

- The command `load` brings in an R formatted file that has been saved with the command `save`.
- An example:

```
load("Class.Stat.Comp/prostate.sav")
names(prostate)
 [1] "patno"  "stage"  "rx"     "dtime"  "status" "age"    "wt"     "pf"
 [9] "hx"     "sbp"    "dbp"    "ekg"    "hg"     "sz"     "sg"     "ap"
[17] "bm"     "sdate"
```

- `load` does not have many options:

```
load(file, envir = parent.frame(), verbose = FALSE)
```

where `envir` is used when you have multiple R environments simultaneously (most people do not do this), and `verbose` set equal to `TRUE` will print the variables names on loading.

Loading Data

- To use `load` from a website, do this:

```
connect1 <- url("http://jgill.wustl.edu/data/Pixel.rda")
load(connect1)
close(connect1)
Pixel[1:10,]
   Dog Side day  pixel
1    1    R   0 1045.8
2    1    R   1 1044.5
3    1    R   2 1042.9
4    1    R   4 1050.4
5    1    R   6 1045.2
6    1    R  10 1038.9
7    1    R  14 1039.8
8    2    R   0 1041.8
9    2    R   1 1045.6
10   2    R   2 1051.0
```

Loading Data

- Sometimes we get data files that have no field delimiters, and instead come with a recipe of column assignments.
- The function `read.fwf` performs this function (also with `scan`).
- For example:

```
pa.raw <- read.fwf("Article.P-Agent/mackenzie.fixed.dat", width=c(
    4,1,2,2,2,2,2,1,1,  #end of long XXX's
    1,2,1,2,1,8,1,2,1,  #column 37 finished
    1,1,2,8,1,1,1,1,1,  #colum 54 finished
    1,8,6,1,1,1,2,1,1,
    1,1,1,              #first deck finished
    5,1,6,1,1,1,7,1,    #colum 32, deck 2 starts
    1,1,1,1,1,1,1,1,1,  #var 59 done
    1,1,1,1,1,1,
    1,1,1,1,1,1,        #finished col42
    1,1,1,1,1,1,1,1,1,  #finished col53/var77
    1,1,1,1,1,1,1,      #finished col60/var84
    2,1,1,1,1,1,1,1,    #finished col69/var92
    1,1,2,1,1,1,1,1,1,1)    #finished col80/var102, deck2
```

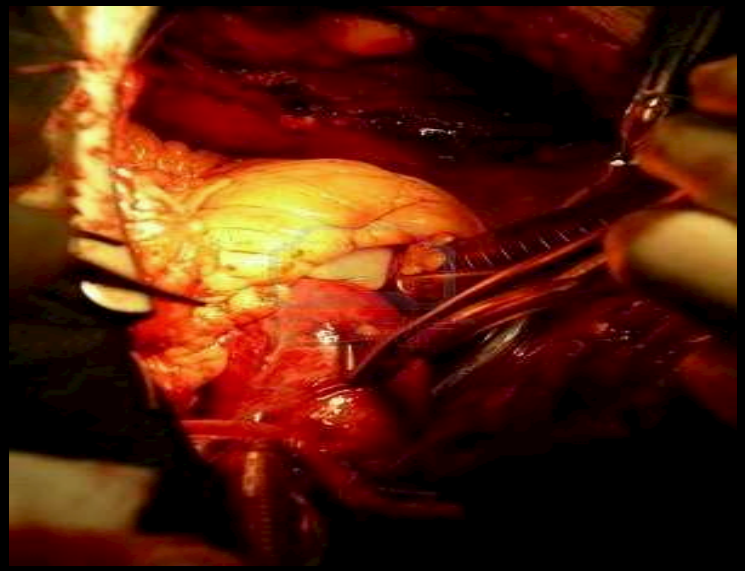
Loading Data

- What about loading data from other statistical software?
- The R library for doing this `foreign` (we'll cover libraries shortly).
- So after typing `library(foreign)`, you have the following functions (and more):
  - `read.dbf`, reads a standard DBF database file into a data frame.
  - `read.dta`, reads a Stata binary data file into a data frame
  - `read.mtp`, reads a Minitab Portable Worksheet into a list
  - `read.octave`, reads Octave text data into a list
  - `read.spss`, reads SPSS data files into a data frame
  - `read.ssd`, uses SAS (you need it installed on the same machine), to convert a ssd file into a transport format, then reads it into a data frame
  - `read.systat`, reads a Systat file into a data frame
  - `read.xport`, reads a SAS XPORT format library and returns a list of data frames
  - `read.S`, reads an Splus version 3 data file.

Cardiac Bypass Surgery Example

- 22 patients undergoing cardiac bypass surgery are randomized into three ventilation groups:
  - Group 1: a 50% nitrous oxide, 50% oxygen mixture continuously for 24 hours after procedure.
  - Group 2: a 50% nitrous oxide, 50% oxygen mixture only during the procedure.
  - Group 3: 35-50% oxygen for 24 hours after procedure.
- Red cell folate levels (RBC Folate) measured in 400 ng/mL (nanograms per milliliter):

Group 1	Group 2	Group 3
243	206	241
251	210	258
275	226	270
291	249	293
347	255	328
354	273	
380	285	
392	295	
	309	



Cardiac Bypass Surgery Example

- Load the data:

```
( cardiac <- read.table("http://jgill.wustl.edu/data/redf.dat",header=TRUE) )
```

```
   Group Folate
1     g1    243
2     g1    251
3     g1    275
4     g1    291
5     g1    347
6     g1    354
7     g1    380
8     g1    392
9     g2    206
10    g2    210
11    g2    226
12    g2    249
13    g2    255
14    g2    273
15    g2    285
16    g2    295
17    g2    309
18    g3    241
19    g3    258
20    g3    270
21    g3    293
22    g3    328
```

Cardiac Bypass Surgery Example

- Some elementary data commands:

```
 apply(table(cardiac),1,sum)
g1 g2 g3
 8  9  5

mean(cardiac$Folate[cardiac$Group == "g1"])
[1] 316.625

table(cardiac)

     Folate
Group 206 210 226 241 243 249 251 255 258 270 273 275 285 291 293 295 309 328 347 354 380 392
   g1   0   0   0   0   1   0   1   0   0   0   0   1   0   1   0   0   0   0   1   1   1   1
   g2   1   1   1   0   0   1   0   1   0   0   1   0   1   0   0   1   1   0   0   0   0   0
   g3   0   0   0   1   0   0   0   0   1   1   0   0   0   0   1   0   0   1   0   0   0   0

sum( table(cardiac)[1,] * sort(cardiac$Folate) )/table(cardiac$Group)[1]
     g1
316.625
```

Cardiac Bypass Surgery Example

- Now run a one-way ANOVA model:

```
folate.aov <- aov(Folate ~ Group, data=cardiac)
summary(folate.aov)
            Df Sum Sq Mean Sq F value Pr(>F)
Group        2  15516    7758   3.711 0.0436
Residuals   19  39716    2090

coef(folate.aov)
(Intercept)     Groupg2     Groupg3
  316.62500   -60.18056   -38.62500
```

- Notice that the estimate of σ^2 is `2090`, the mean square due to α, and the estimate of μ is `316.625`.
- Because of a sum contrast is used here, the effect for `Groupg1` is $-(-60.18056 - 38.62500) = 98.80556$.

Quantiles, with North Carolina County Data

```
NC <- read.table("http://jgill.wustl.edu/data/nc.sub.dat",header=TRUE)

quantile(NC$Percent.Poverty)
    0%    25%    50%    75%   100%
7.100 10.700 13.400 17.825 23.900

quantile(NC$Percent.Poverty,0.6)
60%
15.18

diff(quantile(NC$Percent.Poverty,c(0.25,0.75)))
75%
7.125

IQR(NC$Percent.Poverty)
[1] 7.125
```

What Are Packages?

- User contributed application packages for specific routines.
- Syntax for loading onto your system:

  `install.packages("UsingR","my.library.directory")}.`

- Syntax to start using the package once loaded:

  `library(UsingR)`

- To find your current library locations and packages that are available:

  `library()`

- To find your current *loaded* packages:

  `search()`

  some of which are "autoloaded" when you start R.

Accessing Packages and Their Data

- Start package: `library(glmdm)`
- Load data locally from that package: `data(asia)`
- Look at these data:

```
asia
    ATT DEM FED SYS AUT
1     0  -7   0   0   0
2     0  -8   0   0   0
3     0  -7   0   0   0
4     1   0   0   0   0
:
```

- Print the variable names:

```
names(asia)
[1] "ATT" "DEM" "FED" "SYS" "AUT"
```

Accessing Packages and Their Data

- Attaching related commands:

```
attach(asia)
mean(ATT)
[1] 0.4466667
detach(asia)
```

- This function is very general:

```
with(asia,FUNCTION)
```

which works like this:

```
with(asia,cor(ATT,DEM))
[1] 0.1526725
```

Preview: Running a Regression Model

- Byar DP, Green SB (1980): Bulletin Cancer, Paris, 67:477-488
- `bm`, Bone Metastases: no=0 (420), yes=1 (82).
- `stage`, M0: The cancer has not spread past nearby lymph nodes (289), M1: The cancer has spread beyond the nearby lymph nodes (213).
- `pf`, normal activity 0 (450), some required bed-rest 1 (52).
- `sz`, Size of Primary Tumor (cm^2), median=11.
- `ap`, Serum Prostatic Acid Phosphatase, median=0.7.
- `hg`, Serum Hemoglobin (g/100ml), median=13.7.

Preview: Running a Regression Model

```
library(mgcv)
prostate.df <- read.table("http://jgill.wustl.edu/data/prostate.full.dat",header=TRUE)
prostate.gam1 <- gam(bm ~ stage + pf + log1p(sz) + s(ap) + s(hg),
    family=binomial(link=logit), data=prostate.df)
summary(prostate.gam1)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -20.017      4.087   -4.90  9.7e-07
log1p(sz)      0.375      0.197    1.91    0.057
stage          4.495      1.024    4.39  1.1e-05
pf             1.036      0.464    2.23    0.026

Approximate significance of smooth terms:
       edf Ref.df Chi.sq p-value
s(ap) 3.32   4.03   11.0  0.0270
s(hg) 1.72   2.20   12.2  0.0029
```

Lists

- Lists are simply collections of various objects of different types and sizes.
- Since regression procedures provide different types of information (coefficients, AIC, residuals, etc.), it make sense to deliver this in a list.
- For instance,

```
names(prostate.gam1)
 [1] "coefficients"      "residuals"         "fitted.values"     "family"
 [6] "deviance"          "null.deviance"     "iter"              "weights"
[11] "df.null"           "y"                 "converged"         "boundary"
[16] "reml.scale"        "aic"               "rank"              "sp"
[21] "outer.info"        "scale.estimated"   "scale"             "Vp"
[26] "Ve"                "edf"               "edf1"              "F"
[31] "nsdf"              "sig2"              "method"            "smooth"
[36] "var.summary"       "cmX"               "model"             "control"
[41] "pterms"            "assign"            "xlevels"           "offset"
[46] "min.edf"           "optimizer"         "call"
```

Manipulating Lists

```
temp.list <- list("X"=c(1,2,3),"chars"=c("name1","name2"),
                  "example.matrix"=matrix(c(1,2,3,4),ncol=2))
names(temp.list)
[1] "X"              "chars"          "example.matrix"
temp.list$example.matrix
     [,1] [,2]
[1,]    1    3
[2,]    2    4
temp.list[1]
$X
[1] 1 2 3
temp.list[[1]]
[1] 1 2 3
temp.list[[1]][1]
[1] 1
temp.list[[3]][1,1]
[1] 1
```

Manipulating Lists

```
temp.list$X
[1] 1 2 3
temp.list$chars
[1] "name1" "name2"
temp.list$example.matrix
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Naming Matrices

- Matrices do not begin with names, unless inherited from other objects used to create them, but they can be assigned with the list format:

```
dimnames(ZZ)    NULL
dimnames(ZZ) <- list(c("r1","r2"),c("c1","c2"))
ZZ
         c1       c2
r1 5.000000 2.108736
r2 2.108736 3.145143
rownames(ZZ)
[1] "r1" "r2"
colnames(ZZ)
[1] "c1" "c2"
rownames(ZZ) <- c("R1","R2")
colnames(ZZ) <- c("C1","C2")
dimnames(ZZ)
[[1]]
[1] "R1" "R2"
[[2]]
[1] "C1" "C2"
```

Tabular Analaysis

- Back to the psychology data:

```
   freq anxiety behavioral depression    sex
1     9     Low    Present     Absent   Male
2    32  Medium    Present     Absent   Male
3     4    High    Present     Absent   Male
4     1     Low     Absent     Absent   Male
:
```

- R uses its standard modeling language (discussed in detail in future sessions).
- Basic tables are relatively simple:

```
xtabs(freq ~ depression + sex, data=psych.df)
          sex
depression Male Female
    Absent   58     72
    Mild     52     46
    Severe   33    195
```

Tabular Analaysis

- Here's another, smaller, table:

```
xtabs(freq ~ behavioral + sex, data=psych.df)
          sex
behavioral Male Female
   Present   98    213
   Absent    45    100
```

- We can also add a χ^2 test by wrapping `summary()` around this statement:

```
summary(xtabs(freq ~ behavioral + sex, data=psych.df))
Call: xtabs(formula = freq ~ behavioral + sex, data = psych.df)
Number of cases in table: 456
Number of factors: 2
Test for independence of all factors:
    Chisq = 0.010443, df = 1, p-value = 0.9186
```

Tabular Analaysis

- More complex tables are created with `ftable`:

```
ftable(xtabs(freq ~ depression + behavioral + anxiety, data = psych.df))
                      anxiety Low Medium High
depression behavioral
Absent     Present             18     38   13
           Absent              42     10    9
Mild       Present              8      8   63
           Absent               9      3    7
Severe     Present             32     24  107
           Absent               3     15   47
```

Next Month

- Some basic data summary commands for exploring data.
- Introduction to graphics.
- Tricks for customizing plots.
- Graphing model results.