ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY

SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Applied Statistics and Experimental Design

Applied Statistics with R

Applied Statistics with R and Excel

- R programming language
  - R installation
  - R Studio environment
  - R Basics
  - R Data Types
  - Data Manipulation
  - Functions

- Exploratory Data Analysis with R
- Excel and Data Analysis

Outline

- Why R, and R Paradigm
- References, Tutorials and links
- R Overview
- R Interface
- R Workspace
- Help
- R Packages
- Input/Output
- Reusing Results

Why R ?

- What is R ?
  - a programming “environment”
  - object-oriented
  - freeware
  - provides calculations on matrices
  - excellent graphics capabilities
  - supported by a large user network
- What is R Not ?
  - a statistics software package
  - menu-driven
  - quick to learn
  - a program with a complex graphical interface

Comparison of data analysis packages

Name	Advantages	Disadvantages	Open source?	Typical users
R	Library support; visualization	Steep learning curve	Yes	Finance; Statistics
Matlab	Elegant matrix support; visualization	Expensive; incomplete statistics support	No	Engineering
SciPy/NumPy /Matplotlib	Python (general-purpose programming language)	Immature	Yes	Engineering
Excel	Easy; visual; flexible	Large datasets	No	Business
SAS	Large datasets	Expensive; outdated programming language	No	Business; Government
Stata	Easy statistical analysis		No	Science
SPSS	Like Stata but more expensive and worse			

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

Tutorials cont.

- “Contributed” documentation are tutorials and manuals created by R users
  - Simple R
  - R for Beginners
  - Practical Regression and ANOVA Using R
- R FAQ
- Mailing Lists (listserv)
  - r-help

Tutorials

Each of the following tutorials are in PDF format.

- P. Kuhnert & B. Venables, An Introduction to R: Software for Statistical Modeling & Computing
- J.H. Maindonald, Using R for Data Analysis and Graphics
- B. Muenchen, R for SAS and SPSS Users
- W.J. Owen, The R Guide
- D. Rossiter, Introduction to the R Project for Statistical Computing for Use at the ITC
- W.N. Venebles & D. M. Smith, An Introduction to R

Web links

- Paul Geissler's excellent R tutorial
- Dave Robert's Excellent Labs on Ecological Analysis
- Excellent Tutorials by David Rossitier
- Excellent tutorial an nearly every aspect of R **MOST of these notes follow this web page format**
- Introduction to R by Vincent Zoonekynd
- R Cookbook
- Data Manipulation Reference
- R time series tutorial
- R Concepts and Data Types
- Interpreting Output From lm()
- The R Wiki
- An Introduction to R
- Import / Export Manual
- R Reference Cards
- KickStart
- Hints on plotting data in R
- Regression and ANOVA
- Appendices to Fox Book on Regression
- JGR a Java-based GUI for R [Mac|Windows|Linux]
- A Handbook of Statistical Analyses Using R(Brian S. Everitt and Torsten Hothorn)

R Overview

R is a comprehensive statistical and graphical programming language and is a dialect of the S language:

1988 - S2: RA Becker, JM Chambers, A Wilks
1992 - S3: JM Chambers, TJ Hastie
1998 - S4: JM Chambers

R: initially written by Ross Ihaka and Robert Gentleman at Dep. of Statistics of U of Auckland, New Zealand during 1990s.

Since 1997: international “R-core” team of 15 people with access to common CVS archive.

R Overview

You can enter commands one at a time at the command prompt (>) or run a set of commands from a source file.

There is a wide variety of data types, including vectors (numerical, character, logical), matrices, data frames, and lists.

To quit R, use

>q()

Most functionality is provided through built-in and user-created functions and all data objects are kept in memory during an interactive session.

Basic functions are available by default. Other functions are contained in packages that can be attached to a current session as needed

A key skill to using **R** effectively is learning how to use the built-in help system. Other sections describe the working environment, inputting programs and outputting results, installing new functionality through packages and etc.

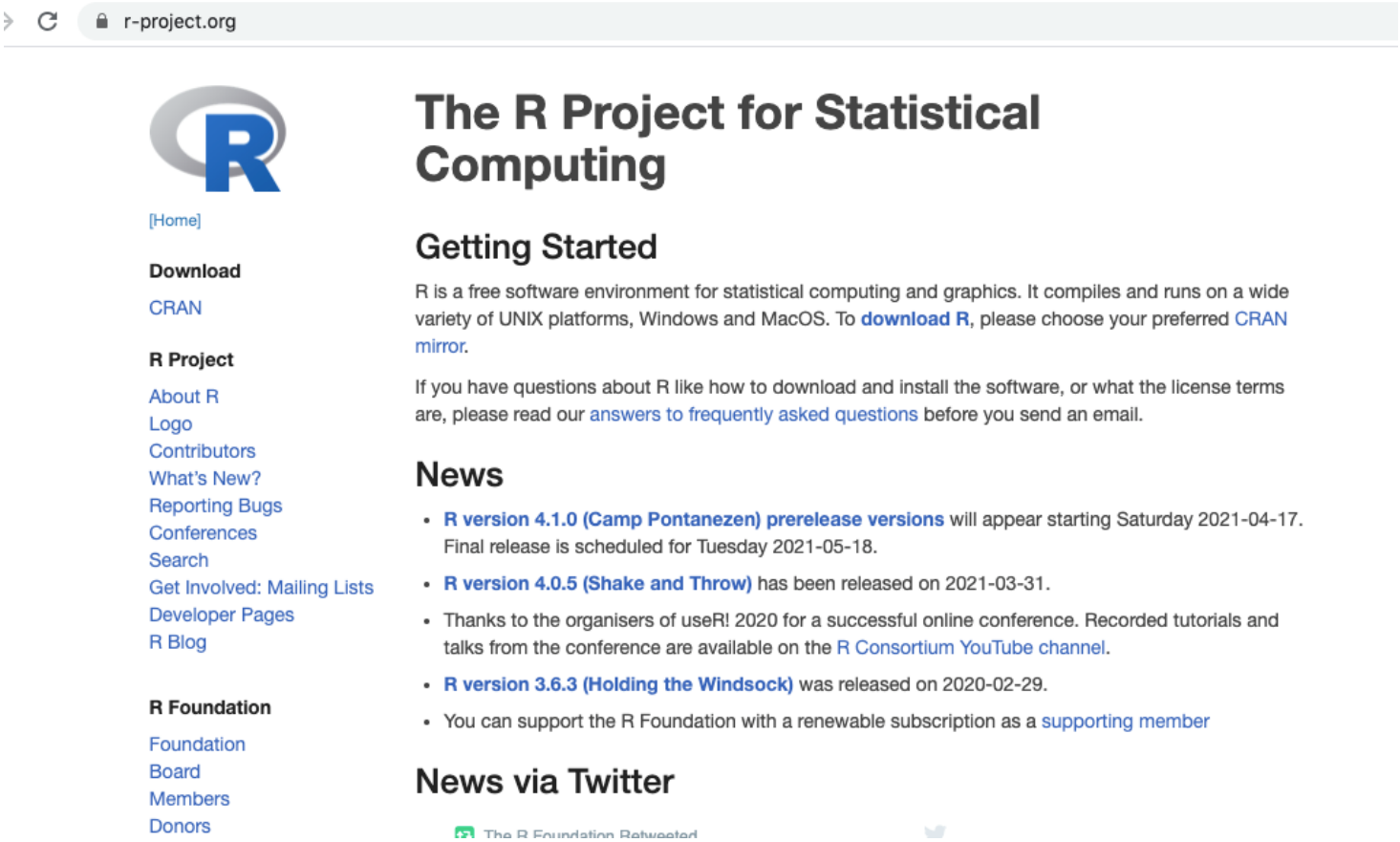
A fundamental design feature of **R** is that the output from most functions can be used as input to other functions. This is described in reusing results.

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

Installing R

- R website



Installing 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

The Comprehensive R Archive Network

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:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download 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 (2021-03-31, Shake and Throw) R-4.0.5.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

Installing R

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

R for Windows

Subdirectories:

base	Binaries for base distribution. This is what you want to **install R for the first time**.
contrib	Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges).
Rtools	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

R Interface

Start the R system, the main window (RGui) with a sub window (R Console) will appear

In the `Console' window the cursor is waiting for you to type in some R commands.

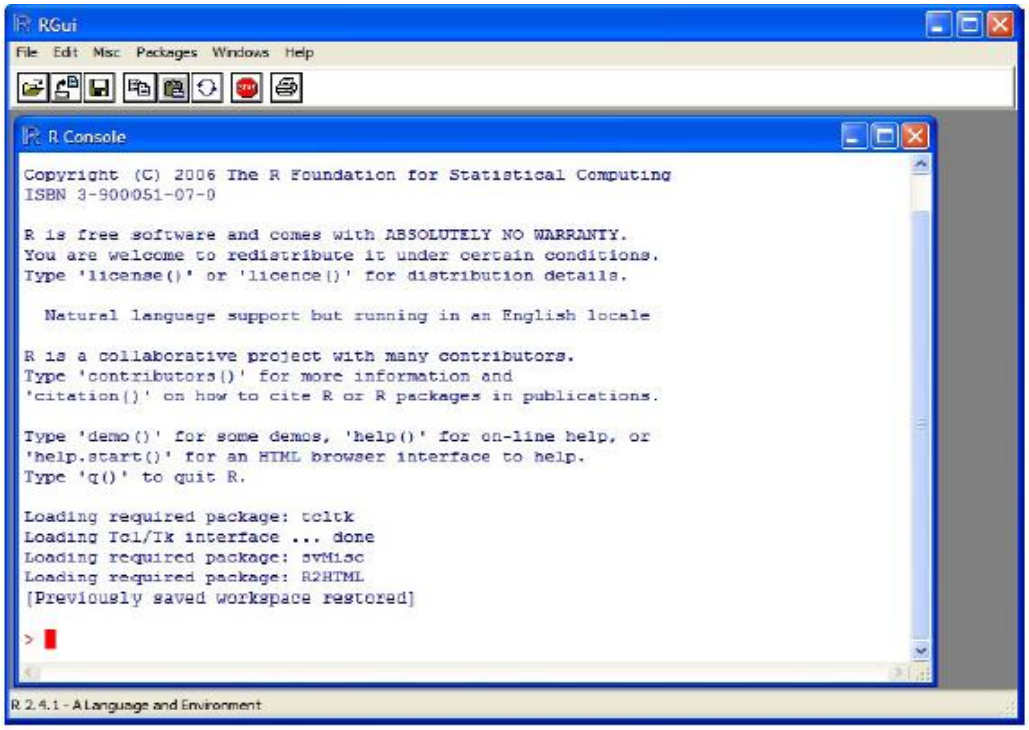
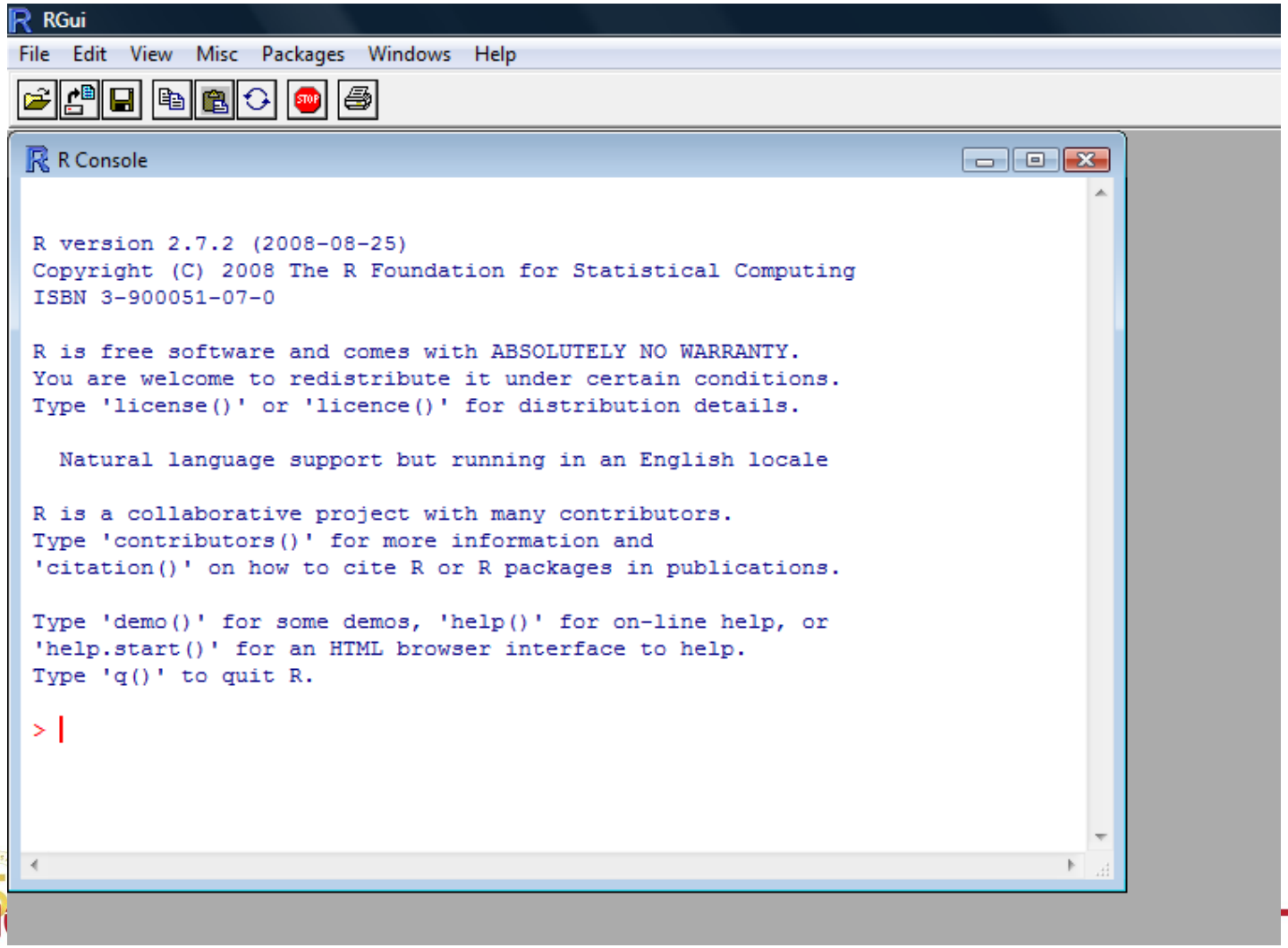
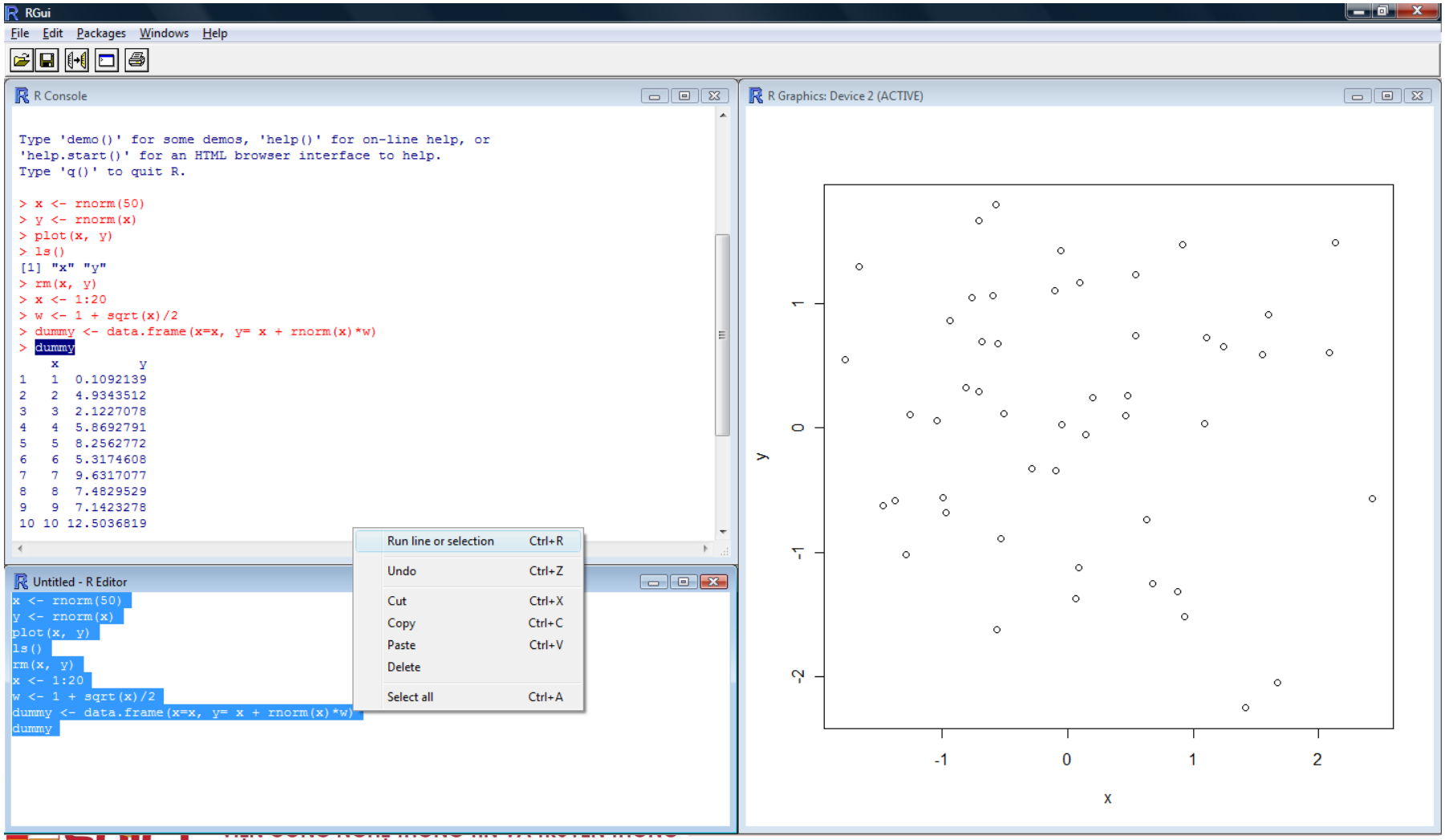


Figure 1.1: The R system on Windows

R Operating Environment



R Operating Environment



Start with R

>help. start()

>help(function name)

>?function name

Example

>?lm

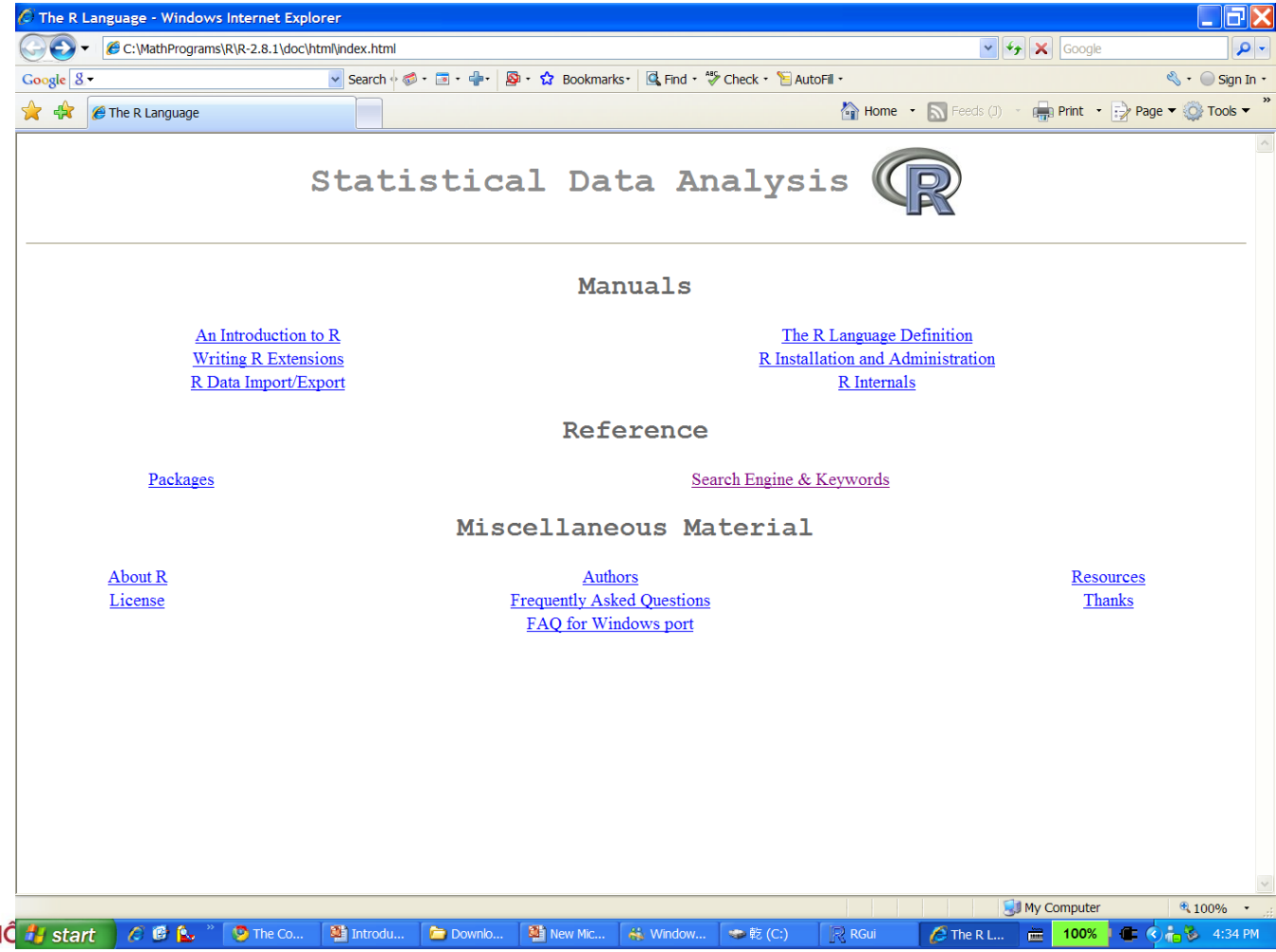
>??object

>help.search(“title”)

Example

>help.search(“test”)

>??colsum



Download and Install Package

R Introduction

- Results of calculations can be stored in objects using the assignment operators:
  - An arrow (<-) formed by a smaller than character and a hyphen without a space!
  - The equal character (=).
- These objects can then be used in other calculations. To print the object just enter the name of the object. There are some restrictions when giving an object a name:
  - Object names cannot contain `strange' symbols like !, +, -, #.
  - A dot (.) and an underscore ( ) are allowed, also a name starting with a dot.
  - Object names can contain a number but cannot start with a number.
  - R is case sensitive, X and x are two different objects, as well as temp and temP.

An example

```
> # An example
> x <- c(1:10)
> x[(x>8) | (x<5)]
> # yields 1 2 3 4 9 10
> # How it works
> x <- c(1:10)
> x
>1 2 3 4 5 6 7 8 9 10
> x > 8
> F F F F F F F F T T
> x < 5
> T T T T F F F F F F
> x > 8 | x < 5
> T T T T F F F F T T
> x[c(T,T,T,T,F,F,F,F,T,T)]
> 1 2 3 4 9 10
```

R Introduction

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

```
> ls()
[1] "x" "y"
```

- So to run the function ls we need to enter the name followed by an opening (and a closing). Entering only ls will just print the object, you will see the underlying R code of the the function ls. Most functions in R accept certain arguments. For example, one of the arguments of the function ls is pattern. To list all objects starting with the letter x:

```
> x2 = 9
> y2 = 10
> ls(pattern="x")
[1] "x" "x2"
```

R Introduction

- 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, x2)
```

- Lets create two small vectors with data and a scatterplot.

```
z2 <- c(1,2,3,4,5,6)
z3 <- c(6,8,3,5,7,1)
plot(z2,z3)
title("My first scatterplot")
```

- R is a case sensitive language.
  - FOO, Foo, and foo are three different objects

R Introduction

```
> x = sin(9)/75
> y = log(x) + x^2
> x
[1] 0.005494913
> y
[1] -5.203902
> m <- matrix(c(1,2,4,1), ncol=2)
> m
> [,1] [,2]
[1,] 1 4
[2,] 2 1
> solve(m)
[,1] [,2]
[1,] -0.1428571 0.5714286
[2,] 0.2857143 -0.1428571
```

What's R?

- Programming language and software environment for data manipulation, calculation and graphical display.

- Originally created by Ross Ihaka and Robert Gentleman at University of Auckland, and now developed by the R Development Core Team.

Where to get R?

- http://www.r-project.org/
- Latest Release: R 2.8.1, on Dec 22, 2008

Why use R?

- ✓IT IS FREE
- ✓Pre-compiled binary versions are provided for Microsoft Windows, Mac OS X, and several other Linux/Unix-like operating systems
- ✓Open source code available freely available on GNU General Public License
- ✓For computationally-intensive tasks, C, C++ and Fortran code can be linked and called at run time
- ✓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 directly at the computer or on hardcopy

R basics

- R basics
  - Data frame, lists, matrices
  - I/O (read.table)
  - Graphical procedures
- How to apply R for statistical problem?
- How to program your R function?
- Statistics basics

- R website: http://www.r-project.org/ (check out its documentation!)

Working with R

- Most packages deal with statistics and data analysis.
- You can run R on different platforms
- Knowing where you are
  - *getwd()* Get Working Directory
  - *setwd()* Set Working Directory
  - *list.files()* List the Files in a Directory/Folder
- Getting quick help with help(), demo(), example()
  - help(plot)
  - demo(nlm) #Nonlinear least-squares
  - example() #example("smooth", package="stats", lib.loc=.Library)

RGui - [R Console]

File Edit Misc Packages Windows Help

```
R : Copyright 2003, The R Development Core Team
Version 1.7.0  (2003-04-16)

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.

Type `demo()' for some demos, `help()' for on-line help, or
`help.start()' for a HTML browser interface to help.
Type `q()' to quit R.

>
```

R 1.7.0 - A Language and Environment

Start | StatisticsWithRPartI.ppt | RGui - [R Console] | vumc | 11:04 PM

RGui - [R Console]

File Edit Misc Packages Windows Help

```
R : Copyright 2003, The R Development Core Team
Version 1.7.0  (2003-04-16)

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.

Type `demo()' for some demos, `help()' for on-line help, or
`help.start()' for a HTML browser interface to help.
Type `q()' to quit R.

> library("MASS")
>
```

R 1.7.0 - A Language and Environment

Start | StatisticsWithRPartI.ppt | RGui - [R Console] | vumc | 11:09 PM

RGui

File Edit Windows

R Console

```
R : Copyright 2003, The R Devel
Version 1.7.0  (2003-04-16)

R is free software and comes wi
You are welcome to redistribute
Type `license()' or `licence()'

R is a collaborative project wi
Type `contributors()' for more

Type `demo()' for some demos, `
`help.start()' for a HTML brows
Type `q()' to quit R.

> library("MASS")
> data()
>
```

R data sets

```
volcano                         Topographic Information on Auckland's Maunga W$
warpbreaks                      The Number of Breaks in Yarn during Weaving
women                           Average Heights and Weights for American Women

Data sets in package 'MASS':

abbey                           Determinations of Nickel Content
accdeaths                       Accidental Deaths in the US 1973-1978
Aids2                           Australian AIDS Survival Data
Animals                         Brain and Body Weights for 28 Species
anorexia                        Anorexia Data on Weight Change
austres                         Quarterly Time Series of the Number of Austral$
bacteria                        Presence of Bacteria after Drug Treatments
beav1                           Body Temperature Series of Beaver 1
beav2                           Body Temperature Series of Beaver 2
biopsy                          Biopsy Data on Breast Cancer Patients
birthwt                         Risk Factors Associated with Low Infant Birth $
Boston                          Housing Values in Suburbs of Boston
cabbages                        Data from a cabbage field trial
caith                           Colours of Eyes and Hair of People in Caithness
Cars93                          Data from 93 Cars on Sale in the USA in 1993
cats                            Anatomical Data from Domestic Cats
cement                          Heat Evolved by Setting Cements
chem                            Copper in Wholemeal Flour
coop                            Co-operative Trial in Analytical Chemistry
cpus                            Performance of Computer CPUs
crabs                           Morphological Measurements on Leptograpsus Cra$
Cushings                        Diagnostic Tests on Patients with Cushing's Sy$
DDT                             DDT in Kale
deaths                          Monthly Deaths from Lung Diseases in the UK
```

Start StatisticsWithRPartI.ppt RGui vumc 11:10 PM

RGui - [R Console]

File Edit Misc Packages Windows Help

```
R : Copyright 2003, The R Development Core Team
Version 1.7.0  (2003-04-16)

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.

Type `demo()' for some demos, `help()' for on-line help, or
`help.start()' for a HTML browser interface to help.
Type `q()' to quit R.

> library("MASS")
> data()
> data(Cars93)
> Cars93
    Manufacturer          Model    Type Min.Price Price Max.Price MPG.city
1          Acura        Integra   Small      12.9  15.9      18.8       25
2          Acura         Legend Midsize      29.2  33.9      38.7       18
3           Audi             90 Compact      25.9  29.1      32.3       20
4           Audi            100 Midsize      30.8  37.7      44.6       19
5            BMW           535i Midsize      23.7  30.0      36.2       22
6          Buick        Century Midsize      14.2  15.7      17.3       22
7          Buick        LeSabre   Large      19.9  20.8      21.7       19
8          Buick     Roadmaster   Large      22.6  23.7      24.9       16
9          Buick        Riviera Midsize      26.3  26.3      26.3       19
10      Cadillac        DeVille   Large      33.0  34.7      36.3       16
11      Cadillac        Seville Midsize      37.5  40.1      42.7       16
12     Chevrolet       Cavalier Compact       8.5  13.4      18.3       25
13     Chevrolet        Corsica Compact      11.4  11.4      11.4       25
14     Chevrolet         Camaro  Sporty      13.4  15.1      16.8       19
15     Chevrolet         Lumina Midsize      13.4  15.9      18.4       21
16     Chevrolet     Lumina_APV     Van      14.7  16.3      18.0       18
17     Chevrolet          Astro     Van      14.7  16.6      18.6       15
18     Chevrolet        Caprice   Large      18.0  18.8      19.6       17
```

R 1.7.0 - A Language and Environment

Start | StatisticsWithRPartI.ppt | RGui - [R Console] | vumc | 11:11 PM

Packages in R environment

- Basic packages
  - "package:methods" "package:stats" "package:graphics“ "package:utils" "package:base"
- Contributed packages
- Bioconductor
  - an open source and open development software project for the analysis and comprehension of genomic data
- You can see what packages loaded now by the command search()
- Install a new package?
  - install.packages("Rcmdr", dependencies=TRUE)

Download and Install Package

- All R functions and datasets are stored in packages. Only when a package is loaded are its contents available. This is down both for efficiency, and to aid package developers.

- To see which packages are installed at your site, issue the command
- >library(boot)

- Users connected to the Internet can use install.packages() and update.packages() to install and update packages.

- To see packages currently loaded, use search().

R Basics

- objects
- naming convention
- assignment
- functions
- R environment
  - workspace
  - history

Objects

- names
- types of objects: vector, factor, array, matrix, data.frame, ts, list
- attributes
  - mode: numeric, character, complex, logical
  - length: number of elements in object
- creation
  - assign a value
  - create a blank object

Naming Convention

- must start with a letter (A-Z or a-z)
- can contain letters, digits (0-9), and/or periods “ . ”
- case-sensitive
  - `mydata` different from `MyData`
- do not use use underscore “_”

R Conflicting objects

- It is not recommended to do, but R allows the user to give an object a name that already exists. If you are not sure if a name already exists, just enter the name in the R console and see if R can find it. R will look for the object in all the libraries (packages) that are currently attached to the R system. R will not warn you when you use an existing name.

```
> mean = 10
> mean
[1] 10
```

- The object mean already exists in the base package, but is now masked by your object mean. To get a list of all masked objects use the function conflicts.

```
>
[1] "body<-" "mean"
```

R Conflicting objects

The object mean already exists in the base package, but is now masked by your object mean. To get a list of all masked objects use the function conflicts.

```
> conflicts()
[1] "body<-" "mean“
```

You can safely remove the object mean with the function rm() without risking deletion of the mean function.

Calling rm() removes only objects in your working environment by default.

Source Codes

you can have input come from a script file (a file containing **R** commands) and direct output to a variety of destinations.

Input

The **source( )** function runs a script in the current session. If the filename does not include a path, the file is taken from the current working directory.

```
# input a script
source("myfile")
```

Assignment

- “<-” used to indicate assignment
  - `x<-c(1,2,3,4,5,6,7)`
  - `x<-c(1:7)`
  - `x<-1:4`

- *note: as of version 1.4 “=“ is also a valid assignment operator*

Functions

- actions can be performed on objects using functions (note: a function is itself an object)
- have arguments and options, often there are defaults
- provide a result
- parentheses () are used to specify that a function is being called

R Datasets

R comes with a number of sample datasets that you can experiment with. Type

> data( )

to see the available datasets. The results will depend on which packages you have loaded. Type

help(*datasetname***)**

for details on a sample dataset.

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. There is a lively R user community and many R packages have been written and made available on CRAN for other users. 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.

R Packages

- When you download R, already a number (around 30) of packages are downloaded as well. To use a function in an R package, that package has to be attached to the system. When you start R not all of the downloaded packages are attached, only seven packages are attached to the system by default. You can use the function search to see a list of packages that are currently attached to the system, this list is also called the search path.

```
> search()
[1] ".GlobalEnv" "package:stats" "package:graphics"
[4] "package:grDevices" "package:datasets" "package:utils"
[7] "package:methods" "Autoloads" "package:base"
```

R Packages

To attach another package to the system you can use the menu or the library function. Via the menu:

Select the `Packages' menu and select `Load package...', a list of available packages on your system will be displayed. Select one and click `OK', the package is now attached to your current R session. Via the library function:

```
> library(MASS)
> shoes
$A
[1] 13.2 8.2 10.9 14.3 10.7 6.6 9.5 10.8 8.8 13.3
$B
[1] 14.0 8.8 11.2 14.2 11.8 6.4 9.8 11.3 9.3 13.6
```

R Packages

- The function library can also be used to list all the available libraries on your system with a short description. Run the function without any arguments

```
> library()
Packages in library 'C:/PROGRA~1/R/R-25~1.0/library':
base                    The R Base Package
Boot            Bootstrap R (S-Plus) Functions  (Canty)
class                   Functions for Classification
cluster                 Cluster Analysis Extended Rousseeuw  et al.
codetools               Code Analysis Tools for R
datasets                The R Datasets Package
DBI                     R Database Interface
foreign                 Read Data Stored by Minitab, S, SAS,
                SPSS, Stata, Systat, dBase, ...
graphics                The R Graphics Package
```

```
install = function() {
install.packages(c("moments","graphics","Rcmdr","hexbin"),
repos="http://lib.stat.cmu.edu/R/CRAN")
}
install()
```

R Workspace & History

Workspace

- during an R session, all objects are stored in a temporary, working memory
- list objects
  - `ls()`
- remove objects
  - `rm()`
- objects that you want to access later must be saved in a “workspace”
  - from the menu bar: File->save workspace
  - from the command line:
    `save(x,file=“MyData.Rdata”)`

RGui - [R Console]

File Edit Misc Packages Windows Help

```
59       Mercedes-Benz 300E
60            Mercury Capri
61           Mercury Cougar
62         Mitsubishi Mirage
63       Mitsubishi Diamante
64            Nissan Sentra
65            Nissan Altima
66             Nissan Quest
67            Nissan Maxima
68       Oldsmobile Achieva
69 Oldsmobile Cutlass_Ciera
70    Oldsmobile Silhouette
71  Oldsmobile Eighty-Eight
72           Plymouth Laser
73           Pontiac LeMans
74          Pontiac Sunbird
75         Pontiac Firebird
76       Pontiac Grand_Prix
77       Pontiac Bonneville
78                 Saab 900
79               Saturn SL
80             Subaru Justy
81            Subaru Loyale
82            Subaru Legacy
83             Suzuki Swift
84            Toyota Tercel
85            Toyota Celica
86             Toyota Camry
87            Toyota Previa
88           Volkswagen Fox
89       Volkswagen Eurovan
90        Volkswagen Passat
91       Volkswagen Corrado
92                Volvo 240
93                Volvo 850
> ls()
[1] "Cars93"
>
```

R 1.7.0 - A Language and Environment

Start | StatisticsWithRPartI.ppt | RGui - [R Console] | vumc | 11:12 PM

History

- command line history
- can be saved, loaded, or displayed
  - `savehistory(file="MyData.Rhistory)`
  - `loadhistory(file="MyData.Rhistory)`
  - `history(max.show=Inf)`

- during a session you can use the arrow keys to review the command history

RGui

File Edit Windows

R Console

```
73            Pontiac LeMans
74           Pontiac Sunbird
75          Pontiac Firebird
76        Pontiac Grand_Prix
77        Pontiac Bonneville
78                  Saab 900
79                 Saturn SL
80              Subaru Justy
81             Subaru Loyale
82             Subaru Legacy
83              Suzuki Swift
84             Toyota Tercel
85             Toyota Celica
86              Toyota Camry
87             Toyota Previa
88            Volkswagen Fox
89        Volkswagen Eurovan
90         Volkswagen Passat
91        Volkswagen Corrado
92                 Volvo 240
93                 Volvo 850
> ls()
[1] "Cars93"
> history(max.show=Inf)
>
```

R History

```
library("MASS")
data()
data(Cars93)
Cars93
ls()
history(max.show=Inf)
```

Start | StatisticsWithRPartI.ppt | RGui | vumc | 11:14 PM

R Workspace

Objects that you create during an R session are hold in memory, the collection of objects that you currently have is called the workspace. This workspace is not saved on disk unless you tell R to do so. This means that your objects are lost when you close R and not save the objects, or worse when R or your system crashes on you during a session.

When you close the RGui or the R console window, the system will ask if you want to save the workspace image. If you select to save the workspace image then all the objects in your current R session are saved in a file .RData. This is a binary file located in the working directory of R, which is by default the installation directory of R.

R Workspace

- During your R session you can also explicitly save the workspace image. Go to the `File‘ menu and then select `Save Workspace...', or use the save.image function.

```
## save to the current working directory
save.image()
## just checking what the current working directory is
getwd()
## save to a specific file and location
save.image("C:\\Program Files\\R\\R-2.5.0\\bin\\.RData")
```

If you have saved a workspace image and you start R the next time, it will restore the workspace. So all your previously saved objects are available again. You can also explicitly load a saved workspace, that could be the workspace image of someone else. Go the `File' menu and select `Load workspace...'.

R Workspace

Commands are entered interactively at the **R** user prompt. **Up** and **down arrow keys** scroll through your command history.

You will probably want to keep different projects in different physical directories.

R gets confused if you use a path in your code like

c:\mydocuments\myfile.txt

This is because R sees "\" as an escape character. Instead, use

c:\\my documents\\myfile.txt

or

c:/mydocuments/myfile.txt

R Workspace

```
getwd() # print the current working directory

ls()  # list the objects in the current workspace
setwd(mydirectory)   # change to mydirectory
setwd("c:/docs/mydir")
#view and set options for the session
    help(options) # learn about available options
    options() # view current option settings
    options(digits=3) # number of digits to print on output
# work with your previous commands
    history() # display last 25 commands
    history(max.show=Inf) # display all previous commands
# save your command history
    savehistory(file="myfile") # default is ".Rhistory"
# recall your command history
    loadhistory(file="myfile") # default is ".Rhistory“
```

R Help

Once **R** is installed, there is a comprehensive built-in help system. At the program's command prompt you can use any of the following:

help.start() # general help
help(*foo*) # help about function *foo*
?*foo* # same thing
apropos("*foo*") # list all function containing string foo
example(*foo*) # show an example of function *foo*

# search for foo in help manuals and archived mailing lists
RSiteSearch("*foo*")

get vignettes on using installed packages
vignette() # show available vingettes
vignette("*foo*") # show specific vignette

Data Input

Data Types

Two most common object types for statistics:

matrix
data frame

Outline

- Data Types
- Importing Data
- Keyboard Input
- Database Input
- Exporting Data
- Viewing Data
- Variable Labels
- Value Labels
- Missing Data
- Date Values

R basic data types

- vector, array, list, matrix, data frame
  - **list**: an ordered collection of data of *arbitrary types.*
  - **vector**: an ordered collection of data of the same type.
  - **matrix**: all elements of a matrix have the same mode, i.e. all numeric, or all character. Thus a matrix is a more restricted structure than a data frame
  - **array**: The generalization from a matrix (2 dimensions) to allow > 2 dimensions gives an array. A matrix is a 2D array.
  - **data frame**: A data frame is a generalization of a matrix, in which different columns may have different modes. All elements of any column must however have the same mode, i.e. all numeric or all factor, or all character.

Vectors

```
a <- c(1,2,5.3,6,-2,4) # numeric vector
b <- c("one","two","three") # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
#logical vector
```

Refer to elements of a vector using subscripts.

```
a[c(2,4)] # 2nd and 4th elements of vector
```

Matrix

- a matrix is a vector with an additional attribute (dim) that defines the number of columns and rows
- only one mode (numeric, character, complex, or logical) allowed
- can be created using `matrix()`

```
x<-matrix(data=0,nr=2,nc=2)
        or
x<-matrix(0,2,2)
```

Matrices

All columns in a matrix must have the same mode(numeric, character, etc.) and the same length.

The general format is

mymatrix <- **matrix(***vector*, **nrow=***r*, **ncol=***c*, **byrow=***FALSE*,**dimnames=list(***char_vector_rownames*, *char_vector_colnames***))**

byrow=TRUE indicates that the matrix should be filled by rows. **byrow=FALSE** indicates that the matrix should be filled by columns (the default). **dimnames** provides optional labels for the columns and rows.

```
# generates 5 x 4 numeric matrix
y<-matrix(1:20, nrow=5,ncol=4)
# another example
cells <- c(1,26,24,68)
rnames <- c("R1", "R2")
cnames <- c("C1", "C2")
mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,
dimnames=list(rnames, cnames))
#Identify rows, columns or elements using subscripts.
x[,4] # 4th column of matrix
x[3,] # 3rd row of matrix
x[2:4,1:3] # rows 2,3,4 of columns 1,2,3
```

Arrays

Arrays are similar to matrices but can have more than two dimensions. See **help(array)** for details.

Lists

An ordered collection of objects (components). A list allows you to gather a variety of (possibly unrelated) objects under one name.

```
# example of a list with 4 components -
    # a string, a numeric vector, a matrix, and a scaler
    w <- list(name="Fred", mynumbers=a, mymatrix=y, age=5.3)

    # example of a list containing two lists
    v <- c(list1,list2)
```

Identify elements of a list using the [[]] convention.

```
mylist[[2]] # 2nd component of the list
```

Data frames

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).

```
d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
mydata <- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed") #variable names
```

There are a variety of ways to identify the elements of a dataframe .

```
myframe[3:5] # columns 3,4,5 of dataframe
myframe[c("ID","Age")] # columns ID and Age from dataframe
myframe$X1 # variable x1 in the dataframe
```

Data Frame

- several modes allowed within a single data frame
- can be created using `data.frame()`

```
L<-LETTERS[1:4] #A B C D
x<-1:4          #1 2 3 4
data.frame(x,L) #create data frame
```

- `attach()` and `detach()`
  - the database is attached to the R search path so that the database is searched by R when it is evaluating a variable.
  - objects in the database can be accessed by simply giving their names

Data Elements

- select only one element
  - `x[2]`
- select range of elements
  - `x[1:3]`
- select all but one element
  - `x[-3]`
- slicing: including only part of the object
  - `x[c(1,2,5)]`
- select elements based on logical operator
  - `x(x>3)`

Data frame

- A data frame is an object with rows and columns (a bit like a 2D matrix)
  - The rows contain different observations from your study, or measurements from your experiment
  - The columns contain the values of different variables
- All the values of the same variable must go in the same column!
  - If you had an experiment with three treatments (control, pre-heated and pre-chilled), and four measurements per treatment

Data Frame

How to define a data frame:

```
> accountants <- data.frame(home=statef, loot=incomes, shot=incomef)
```

The components of a data frame could be vectors (numerical, character or logical), matrix, or other data frames.

attach( ) and detach( )

These two functions are used to make variables in the list temporarily visible. Thus, we can use variable name directly instead of using ListName$VarName.

Factors

Tell **R** that a variable is **nominal** by making it a factor. The factor stores the nominal values as a vector of integers in the range [1... k] (where k is the number of unique values in the nominal variable), and an internal vector of character strings (the original values) mapped to these integers.

```
# variable gender with 20 "male" entries and
# 30 "female" entries
    gender <- c(rep("male",20), rep("female", 30))
    gender <- factor(gender)
# stores gender as 20 1s and 30 2s and associates
# 1=female, 2=male internally (alphabetically)
# R now treats gender as a nominal variable
    summary(gender)
```

What’s the correct data frame?

Control	Pre-heated	Pre-chilled
6.1	6.3	7.1
..	..	..
..	..	..
..	..	..

Response	Treatment
6.1	Control
5.9	Control
..	..
..	..
6.2	Pre-heated

Factors

- Factors: classification variables

```
> trt <- factor(rep(c("Control", "Treated"), c(3, 4)))
> str(trt)
 Factor w/ 2 levels "Control","Treated": 1 1 1 2 2 2 2
> summary(trt)                    # summary gives a frequency table
Control Treated
      3       4
```

- If the levels of a factor are numeric (e.g. the treatments are labelled“1”, “2”, and “3”) it is important to ensure that the data are actually stored as a factor and not as numeric data. Always check this by using summary.

Assigning values to variables

- R uses ‘gets’ <- rather than the more familier ‘equals’ = sign
  - x <- 12.6 #assign value to a numerical variable
  - y <- c(3, 7, 9, 11) #vector
  - a <- 1:6 # : means a series of integers between
  - b <- seq(0.5, 0, -0.1)

Vector and Assignment

The simplest data structure in R: vector

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

" <- " is an assignment operator. In most context, " = " is the same.

The further assignment

```
> y <- c(x, 0, x)
```

consisting of 11 entries.

> x <- 1:30 x <- seq(1, 30)

```
> seq(-5, 5, by=.2) -> s3
```

```
> s4 <- seq(length=51, from=-5, by=.2)
```

equal to `c(-5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0)`

```
> s5 <- rep(x, times=5)
> s6 <- rep(x, each=5)
```

Basic Commands

+, - , *, /, sqrt, sum

length, sort, max, min

NA, NaN (eg. 0/0), Inf, -Inf # Notice: **R is case sensitive!**

/*/ #matrix multiplication

mean(x) =sum(x)/length(x) # sample mean

var(x) =`sum((x-mean(x))^2)/(length(x)-1)`

sample variance

Character vector: use double quote

```
> labs <- paste(c("X","Y"), 1:10, sep="")
```

Data Import, Export and Viewing

Importing Data

- `read.table()`
  - reads in data from an external file
- `data.entry()`
  - create object first, then enter data
- `c()`
  - concatenate
- `scan()`
  - prompted data entry
- R has ODBC for connecting to other programs

Data entry & editing

- start editor and save changes
  - `data.entry(x)`
- start editor, changes not saved
  - `de(x)`
- start text editor
  - `edit(x)`

Data input from a file

- Learning how to read the data into R is amongst the most important topics you need to master

- From the file
  - read.table()

Obtain parts of your data

- Subscripts with vectors

```
y[3]  #third element
y[3:7]  #from third to 7th elements
y[c(3, 5, 6, 9)]  #3rd, 5th, 6th, and 9th elements
y[-1]  #drop the first element; dropping using negative ingegers
y[y > 6] #all the elements that are > 6
```

- Subscripts with matrices, arrays, and dataframe

```
A <- array(1:30, c(5, 3, 2)) #3D array
A[,2:3,]
A[2:4,2:3,]
worms <- read.table("worms.txt", header=T, row.names=1)
worms[,1:3] #all the rows, columns 1 to 3
```

- Subscripts with lists

```
cars <- list(c("Toyota", "Nissan"), c(1500, 1750), c("blue", "red", "black")
cars[[3]] # c("blue", "red", "black")
cars[[3]][2] # "red"  note: not cars[3][2]
```

Using logic conditions to get subsets

```
> library(lattice)
> barley[1:7,]
     yield   variety year            site
1 27.00000 Manchuria 1931 University Farm
2 48.86667 Manchuria 1931          Waseca
3 27.43334 Manchuria 1931          Morris
4 39.93333 Manchuria 1931       Crookston
5 32.96667 Manchuria 1931    Grand Rapids
6 28.96667 Manchuria 1931          Duluth
7 43.06666   Glabron 1931 University Farm

> Duluth1932 <- barley[barley$year=="1932" &
  barley$site=="Duluth", c("variety","yield")]
```

Importing Data

Importing data into **R** is fairly simple.

For Stata and Systat, use the **foreign** package.

For SPSS and SAS I would recommend the **Hmisc** package for ease and functionality.

See the **Quick-R** section on **packages**, for information on obtaining and installing the these packages.

Example of importing data are provided below.

From A Comma Delimited Text File

```
# first row contains variable names, comma is separator
# assign the variable id to row names
# note the / instead of \ on mswindows systems

mydata <- read.table("c:/mydata.csv", header=TRUE, sep=",",
row.names="id")
```

From Excel

The best way to read an Excel file is to export it to a comma delimited file and import it using the method above.

On windows systems you can use the **RODBC** package to access Excel files. The first row should contain variable/column names.

```
# first row contains variable names
# we will read in workSheet mysheet
library(RODBC)
channel <- odbcConnectExcel("c:/myexel.xls")
mydata <- sqlFetch(channel, "mysheet")
odbcClose(channel)
```

From SAS

- # save SAS dataset in trasport format
  libname out xport 'c:/mydata.xpt';
  data out.mydata;
  set sasuser.mydata;
  run;
- library(foreign)

#bsl=read.xport(“mydata.xpt")

Keyboard Input

Usually you will obtain a dataframe by importing it from **SAS**, **SPSS**, **Excel**, **Stata**, a database, or an ASCII file. To create it interactively, you can do something like the following.

```
# create a dataframe from scratch
age <- c(25, 30, 56)
gender <- c("male", "female", "male")
weight <- c(160, 110, 220)
mydata <- data.frame(age,gender,weight)
```

Keyboard Input

You can also use **R**'s built in spreadsheet to enter the data interactively, as in the following example.

```
# enter data using editor
mydata <- data.frame(age=numeric(0), gender=character(0),
weight=numeric(0))
mydata <- edit(mydata)
# note that without the assignment in the line above,
# the edits are not saved!
```

Exporting Data

There are numerous methods for exporting **R** objects into other formats . For SPSS, SAS and Stata. you will need to load the **foreign** packages. For Excel, you will need the **xlsReadWrite** package.

To A Tab Delimited Text File

```
write.table(mydata, "c:/mydata.txt", sep="\t")
```

To an Excel Spreadsheet

```
library(xlsReadWrite)
    write.xls(mydata, "c:/mydata.xls")
```

To SAS

```
library(foreign)
    write.foreign(mydata, "c:/mydata.txt",
    "c:/mydata.sas",   package="SAS")
```

Save object/data

- Every R object can be stored into and restored from a file with the commands “save” and “load”.
  - > save(x, file=“x.Rdata”)
  - > load(“x.Rdata”)

- Importing and exporting data with rectangular tables in the form of tab-delimited text files.
  - > write.table(x, file=“x.txt”, sep=“\t”)

Import and Export Data

File types that can be imported into R:

.data, .txt, .xls, .xlsx, .html, .xml, etc.

Example of importing/exporting text files into R:

```
> data<-read.table("C:/……/data.txt", header=TRUE, sep="\t")
> write.table(x, "C:/……/data.txt", header=TRUE, sep="\t")
```

Other data import commands: scan() ……

For data import/export:

http://cran.r-project.org/doc/manuals/R-data.html

Viewing Data

There are a number of functions for listing the contents of an object or dataset.

```
# list objects in the working environment
ls()
# list the variables in mydata
names(mydata)
# list the structure of mydata
str(mydata)
# list levels of factor v1 in mydata
levels(mydata$v1)
# dimensions of an object
dim(object)
```

Viewing Data

There are a number of functions for listing the contents of an object or dataset.

```
# class of an object (numeric, matrix, dataframe, etc)
class(object)
# print mydata
mydata
# print first 10 rows of mydata
head(mydata, n=10)
# print last 5 rows of mydata
tail(mydata, n=5)
```

Variable Labels

R's ability to handle variable labels is somewhat unsatisfying.

If you use the **Hmisc** package, you can take advantage of some labeling features.

```
library(Hmisc)
label(mydata$myvar) <- "Variable label for variable myvar"
describe(mydata)
```

Unfortunately the label is only in effect for functions provided by the **Hmisc** package, such as **describe()**. Your other option is to use the variable label as the variable name and then refer to the variable by position index.

```
names(mydata)[3] <- "This is the label for variable 3"
mydata[3] # list the variable
```

Value Labels

To understand value labels in **R**, you need to understand the data structure factor.

You can use the factor function to create your own value lables.

```
# variable v1 is coded 1, 2 or 3
# we want to attach value labels 1=red, 2=blue,3=green
mydata$v1 <- factor(mydata$v1,
levels = c(1,2,3),
labels = c("red", "blue", "green"))
# variable y is coded 1, 3 or 5
# we want to attach value labels 1=Low, 3=Medium, 5=High
mydata$v1 <- ordered(mydata$y,
levels = c(1,3, 5),
labels = c("Low", "Medium", "High"))
```

Use the **factor()** function for **nominal data** and the **ordered()** function for **ordinal data**. **R** statistical and graphic functions will then treat the data appropriately.

Note: factor and ordered are used the same way, with the same arguments. The former creates factors and the later creates ordered factors.

Missing Data

In **R**, 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). Unlike SAS, **R** uses the same symbol for character and numeric data.

Testing for Missing Values

```
is.na(x) # returns TRUE of x is missing
y <- c(1,2,3,NA)
is.na(y) # returns a vector (F F F T)
```

Recoding Values to Missing

```
# recode 99 to missing for variable v1
# select rows where v1 is 99 and recode column v1
mydata[mydata$v1==99,"v1"] <- NA
```

Excluding Missing Values from Analyses

Arithmetic functions on missing values yield missing values.

```
x <- c(1,2,NA,3)
mean(x)          # returns NA
mean(x, na.rm=TRUE) # returns 2
```

Missing Data

The function **complete.cases()** returns a logical vector indicating which cases are complete.

```
# list rows of data that have missing values
mydata[!complete.cases(mydata),]
```

The function **na.omit()** returns the object with listwise deletion of missing values.

```
# create new dataset without missing data
newdata <- na.omit(mydata)
```

Advanced Handling of Missing Data

Most modeling functions in **R** offer options for dealing with missing values. You can go beyond pairwise of listwise deletion of missing values through methods such as multiple imputation. Good implementations that can be accessed through **R** include **Amelia II**, **Mice**, and **mitools**.

Date Values

Dates are represented as the number of days since 1970-01-01, with negative values for earlier dates.

```
# use as.Date( ) to convert strings to dates
mydates <- as.Date(c("2007-06-22", "2004-02-13"))
# number of days between 6/22/07 and 2/13/04
    days <- mydates[1] - mydates[2]
```

Sys.Date( ) returns today's date.

Date() returns the current date and time.

Date Values

The following symbols can be used with the format( ) function to print dates.

```
# print today's date
today <- Sys.Date()
format(today, format="%B %d %Y")
    "June 20 2007"
```

Symbol	Meaning	Example
%d	day as a number (0-31)	01-31
%a	abbreviated weekday	Mon
%A	unabbreviated weekday	Monday
%m	month (00-12)	00-12
%b	abbreviated month	Jan
%B	unabbreviated month	January
%y	2-digit year	07
%Y	4-digit year	2007

Output

Output

The **sink( )** function defines the direction of the output.

```
# direct output to a file
    sink("myfile", append=FALSE, split=FALSE)

    # return output to the terminal
    sink()
```

The **append** option controls whether output overwrites or adds to a file.

The **split** option determines if output is also sent to the screen as well as the output file.

Here are some examples of the **sink()** function.

```
# output directed to output.txt in c:\projects directory.
# output overwrites existing file. no output to terminal.
sink("myfile.txt", append=TRUE, split=TRUE)
```

Control Statements

The language has available a conditional construction of the form

```
> if (expr_1) expr_2 else expr_3
```

Please try "||" "&&" , "|" "&" to see the difference;

There is also a `for` loop construction which has the form

```
> for (name in expr_1) expr_2
```

Also check functions "repeat", "while".

Write R function

- A function definition looks like

```
funcdemo <- function(x, y)
{
        z <- x + y
        return (z)
}
```

Control flow

- if(cond) expr
- if(cond) cons.expr else alt.expr
- for(var in seq) expr
  - for (i in 1:n)
- while(cond) expr
- repeat expr
- break & next

```
for(i in 1:10) {
   print(i*i)
}

i<-1
while(i<=10) {
   print(i*i)
   i<-i+sqrt(i)
}
```

Statistical functions

`rnorm, dnorm, pnorm, qnorm`	Normal distribution random sample, density, cdf and quantiles
`lm, glm, anova`	Model fitting
`loess, lowess`	Smooth curve fitting
`sample`	Resampling (bootstrap, permutation)
`.Random.seed`	Random number generation
`mean, median`	Location statistics
`var, cor, cov, mad, range`	Scale statistics
`svd, qr, chol, eigen`	Linear algebra

Graphical procedures in R

- High-level plotting functions create a new plot on the graphics device, possibly with axes, labels, titles and so on.
- Low-level plotting functions add more information to an existing plot, such as extra points, lines and labels.
- Interactive graphics functions allow you interactively add information to, or extract information from, an existing plot, using a pointing device such as a mouse.

Graphic Plot

- install.packages("gplots")
  - Gplots provides heatmap2 (providing color key)
- Plot Types: Line Charts, Bar Charts, Histograms, Pie Charts, Dot Charts, etc.

- Format:
  >PLOT-TYPE(PLOT-DATA, DETAILS)

- PLOT-TYPE: plot, plot.xy, barplot, pie, dotchart, etc.
- PLOT-DATA: Data, Data$XXX, as.matrix(Data), etc.
- Details: axes, col, pch, lty, ylim, type, xlab, ylab, etc.

- For graphics plot:
- *http://www.harding.edu/fmccown/R/*

plot(x, y)

`plot(xy)` If x and y are vectors, `plot(x, y)` produces a scatterplot of y against x. The same effect can be produced by supplying one argument (second form) as either a list containing two elements x and y or a two-column matrix.

`plot(x)` If x is a time series, this produces a time-series plot. If x is a numeric vector, it produces a plot of the values in the vector against their index in the vector. If x is a complex vector, it produces a plot of imaginary versus real parts of the vector elements.

qqnorm(x)

`qqline(x)`
`qqplot(x, y)`

Distribution-comparison plots. The first form plots the numeric vector `x` against the expected Normal order scores (a normal scores plot) and the second adds a straight line to such a plot by drawing a line through the distribution and data quartiles. The third form plots the quantiles of `x` against those of `y` to compare their respective distributions.

hist(x)

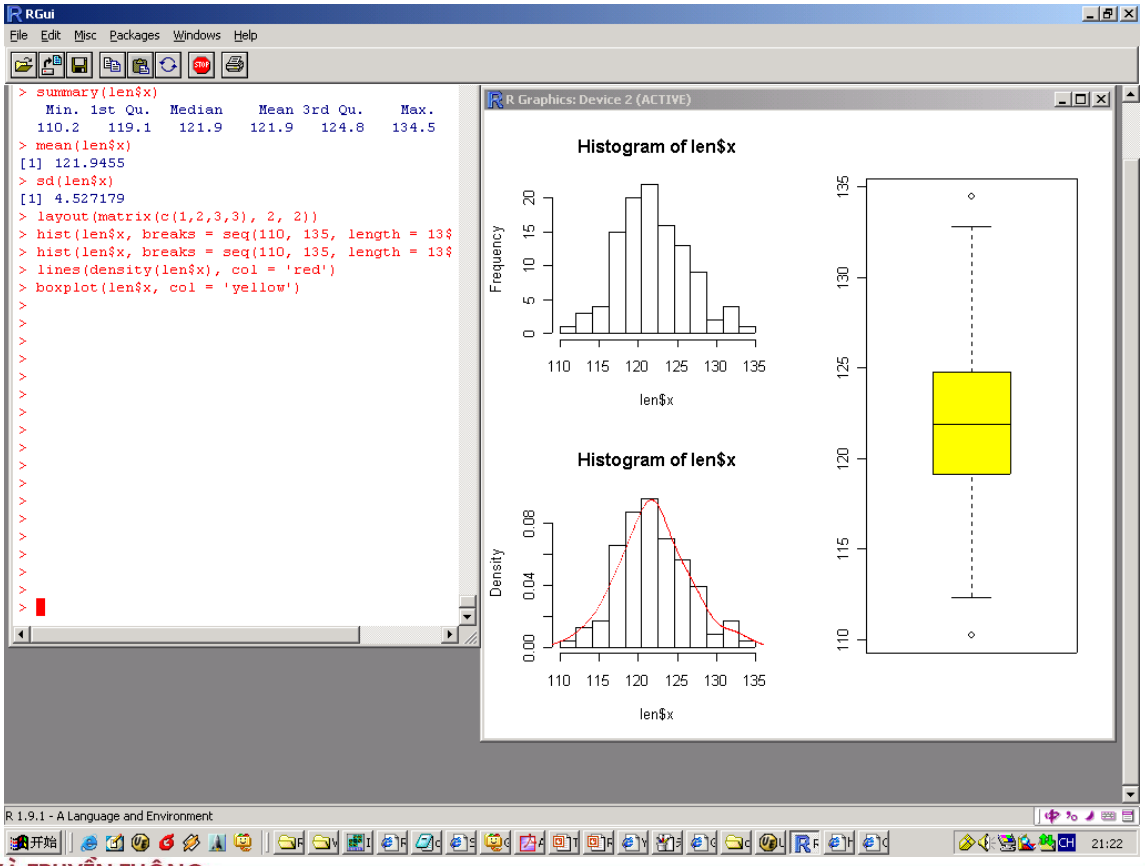
hist(x, nclass=*n*)

hist(x, breaks=*b*, ...)

Summary of your data

- Commands
  - summary()
  - mean()
  - var(), sd()
  - min(), max()

  - hist()
  - boxplot()



Microarray data analysis

Pre-processing

CEL,
affy
vsn

.gpr,
.
marray
limma
vsn

exprSet

Differential expression

siggenes
genefilter
limma
multtest

Graphs & networks

graph
RBGL
Rgraphviz

Cluster analysis

CRAN
class
cluster
MASS
mva

Prediction

CRAN
class
e1071
ipred
LogitBoost
MASS
nnet
randomForest
rpart

Annotation

annotate
annaffy
+ metadata packages

Graphics

geneplotter
hexbin
+ CRAN

Graphs

To redirect graphic output use one of the following functions. Use **dev.off( )** to return output to the terminal.

Function	Output to
pdf("mygraph.pdf")	pdf file
win.metafile("mygraph.wmf")	windows metafile
png("mygraph.png")	png file
jpeg("mygraph.jpg")	jpeg file
bmp("mygraph.bmp")	bmp file
postscript("mygraph.ps")	postscript file

Redirecting Graphs

```
# example - output graph to jpeg file
    jpeg("c:/mygraphs/myplot.jpg")
    plot(x)
    dev.off()
```

Reusing Results

One of the most useful design features of **R** is that the output of analyses can easily be saved and used as input to additional analyses.

```
# Example 1
lm(mpg~wt, data=mtcars)
```

This will run a simple linear regression of miles per gallon on car weight using the dataframe mtcars. Results are sent to the screen. Nothing is saved.

Reusing Results

```
# Example 2
fit <- lm(mpg~wt, data=mtcars)
```

This time, the same regression is performed but the results are saved under the name fit. No output is sent to the screen. However, you now can manipulate the results.

```
str(fit) # view the contents/structure of "fit“
```

The assignment has actually created a list called "fit" that contains a wide range of information (including the predicted values, residuals, coefficients, and more.

Reusing Results

```
# plot residuals by fitted values
plot(fit$residuals, fit$fitted.values)
```

To see what a function returns, look at the **value** section of the online help for that function. Here we would look at **help(lm)**.

The results can also be used by a wide range of other functions.

```
# produce diagnostic plots
plot(fit)
```

Useful Functions

```
length(object) # number of elements or components
str(object)    # structure of an object
class(object)  # class or type of an object
names(object)  # names
c(object,object,...)  # combine objects into a vector
cbind(object, object, ...) # combine objects as columns
rbind(object, object, ...) # combine objects as rows
ls()       # list current objects
rm(object) # delete an object
newobject <- edit(object) # edit copy and save a newobject
fix(object)               # edit in place
```

Data Manipulation

Outline

- Creating New Variable
- Operators
- Built-in functions
- Control Structures
- User Defined Functions
- Sorting Data
- Merging Data
- Aggregating Data
- Reshaping Data
- Sub-setting Data
- Data Type Conversions

Introduction

Once you have access to your data, you will want to massage it into useful form. This includes creating new variables (including recoding and renaming existing variables), sorting and merging datasets, aggregating data, reshaping data, and subsetting datasets (including selecting observations that meet criteria, randomly sampling observation, and dropping or keeping variables).

Introduction

Each of these activities usually involve the use of **R**'s built-in operators (arithmetic and logical) and functions (numeric, character, and statistical). Additionally, you may need to use control structures (if-then, for, while, switch) in your programs and/or create your own functions. Finally you may need to convert variables or datasets from one type to another (e.g. numeric to character or matrix to dataframe).

Creating new variables

- Use the assignment operator **<-** to create new variables. A wide array of operators and functions are available here.
- # Three examples for doing the same computations

  ```
  mydata$sum <- mydata$x1 + mydata$x2
  mydata$mean <- (mydata$x1 + mydata$x2)/2

  attach(mydata)
  mydata$sum <- x1 + x2
  mydata$mean <- (x1 + x2)/2
  detach(mydata)
  ```
- ```
  mydata <- transform( mydata,
  sum = x1 + x2,
  mean = (x1 + x2)/2
  )
  ```
  ```

Creating new variables

Recoding variables

- In order to recode data, you will probably use one or more of R's control structures.
- # create 2 age categories
  mydata$agecat <- ifelse(mydata$age > 70,
  c("older"), c("younger"))
  # another example: create 3 age categories
  attach(mydata)
  mydata$agecat[age > 75] <- "Elder"
  mydata$agecat[age > 45 & age <= 75] <- "Middle Aged"
  mydata$agecat[age <= 45] <- "Young"
  detach(mydata)

Creating new variables

Recoding variables

- In order to recode data, you will probably use one or more of R's control structures.
- # create 2 age categories
  mydata$agecat <- ifelse(mydata$age > 70, c("older"), c("younger"))

  # another example: create 3 age categories
  attach(mydata)
  mydata$agecat[age > 75] <- "Elder"
  mydata$agecat[age > 45 & age <= 75] <- "Middle Aged"
  mydata$agecat[age <= 45] <- "Young"
  detach(mydata)

Creating new variables

Renaming variables

- You can rename variables programmatically or interactively.
- # rename interactively
  fix(mydata) # results are saved on close

  # rename programmatically
  library(reshape)
  mydata <- rename(mydata, c(oldname="newname"))

  # you can re-enter all the variable names in order
  # changing the ones you need to change. The limitation
  # is that you need to enter all of them!
  names(mydata) <- c("x1","age","y", "ses")

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if x is TRUE

Control Structures

- **R** has the standard control structures you would expect. **expr** can be multiple (compound) statements by enclosing them in braces { }. It is more efficient to use built-in functions rather than control structures whenever possible.

Control Structures

- **if-else**
- if (*cond*) *expr*
  if (*cond*) *expr1* else *expr2*
- **for**
- for (*var* in *seq*) *expr*
- **while**
- while (*cond*) *expr*
- **switch**
- switch(*expr*, ...)
- **ifelse**
- ifelse(*test*,*yes*,*no*)

Control Structures

- ```
  # transpose of a matrix
  # a poor alternative to built-in t() function

  mytrans <- function(x) {
    if (!is.matrix(x)) {
      warning("argument is not a matrix: returning NA")
      return(NA_real_)
    }
    y <- matrix(1, nrow=ncol(x), ncol=nrow(x))
    for (i in 1:nrow(x)) {
      for (j in 1:ncol(x)) {
        y[j,i] <- x[i,j]
      }
    }
  return(y)
  }
  ```
  ```

Control Structures

- # try it
  z <- matrix(1:10, nrow=5, ncol=2)
  tz <- mytrans(z)

R built-in functions

Almost everything in **R** is done through functions. Here I'm only referring to numeric and character functions that are commonly used in creating or recoding variables.

Note that while the examples on this page apply functions to individual variables, many can be applied to vectors and matrices as well.

Numeric Functions

Function	Description
abs(*x*)	absolute value
sqrt(*x*)	square root
ceiling(*x*)	ceiling(3.475) is 4
floor(*x*)	floor(3.475) is 3
trunc(*x*)	trunc(5.99) is 5
round(*x*, digits=*n*)	round(3.475, digits=2) is 3.48
signif(*x*, digits=*n*)	signif(3.475, digits=2) is 3.5
cos(*x*), sin(*x*), tan(*x*)	also acos(*x*), cosh(*x*), acosh(*x*), etc.
log(*x*)	natural logarithm
log10(*x*)	common logarithm
exp(*x*)	e^*x*

Character Functions

Function	Description
substr(*x***, start=***n1***, stop=***n2***)**	Extract or replace substrings in a character vector. x <- "abcdef" substr(x, 2, 4) is "bcd" substr(x, 2, 4) <- "22222" is "a222ef"
grep(*pattern***,** *x* **,** **ignore.case**=FALSE**, fixed**=FALSE**)**	Search for *pattern* in *x*. If fixed =FALSE then *pattern* is a regular expression. If fixed=TRUE then *pattern* is a text string. Returns matching indices. grep("A", c("b","A","c"), fixed=TRUE) returns 2
sub(*pattern***,** *replacement***,** *x***,** **ignore.case** =FALSE**, fixed**=FALSE**)**	Find *pattern* in *x* and replace with *replacement* text. If fixed=FALSE then *pattern* is a regular expression. If fixed = T then *pattern* is a text string. sub("\\s",".","Hello There") returns "Hello.There"
strsplit(*x***,** *split***)**	Split the elements of character vector *x* at *split*. strsplit("abc", "") returns 3 element vector "a","b","c"
paste(..., sep="")	Concatenate strings after using *sep* string to seperate them. paste("x",1:3,sep="") returns c("x1","x2" "x3") paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3") paste("Today is", date())
toupper(*x***)**	Uppercase
tolower(*x***)**	Lowercase

Stat/Prob Functions

- The following table describes functions related to probaility distributions. For random number generators below, you can use set.seed(1234) or some other integer to create reproducible pseudo-random numbers.

Function	Description

dnorm(*x*)	normal density function (by default m=0 sd=1)
	# plot standard normal curve
	x <- pretty(c(-3,3), 30)
	y <- dnorm(x)
	plot(x, y, type='l', xlab="Normal Deviate", ylab="Density", yaxs="i")

pnorm(*q*)	cumulative normal probability for q (area under the normal curve to the right of q) pnorm(1.96) is 0.975
qnorm(*p*)	normal quantile. value at the p percentile of normal distribution qnorm(.9) is 1.28 # 90th percentile
rnorm(*n*, m=0,sd=1)	n random normal deviates with mean m and standard deviation sd. #50 random normal variates with mean=50, sd=10 x <- rnorm(50, m=50, sd=10)

dbinom(*x*, *size*, *prob*)
pbinom(*q*, *size*, *prob*)
qbinom(*p*, *size*, *prob*)
rbinom(*n*, *size*, *prob*)
	binomial distribution where size is the sample size
	and prob is the probability of a heads (pi)
	# prob of 0 to 5 heads of fair coin out of 10 flips
	dbinom(0:5, 10, .5)
	# prob of 5 or less heads of fair coin out of 10 flips
	pbinom(5, 10, .5)

dpois(*x*, *lamda*)
ppois(*q*, *lamda*)
qpois(*p*, *lamda*)
rpois(*n*, *lamda*)
	poisson distribution with m=std=lamda
	#probability of 0,1, or 2 events with lamda=4
	dpois(0:2, 4)
	# probability of at least 3 events with lamda=4
	1- ppois(2,4)

dunif(*x*, min=0, max=1) **punif(*q*, min=0, max=1)** **qunif(*p*, min=0, max=1)** **runif(*n*, min=0, max=1)**	uniform distribution, follows the same pattern as the normal distribution above. #10 uniform random variates x <- runif(10)

Function	Description
mean(*x***, trim=**0**, na.rm=**FALSE**)**	mean of object x # trimmed mean, removing any missing values and # 5 percent of highest and lowest scores mx <- mean(x,trim=.05,na.rm=TRUE)
sd(*x***)**	standard deviation of object(x). also look at var(x) for variance and mad(x) for median absolute deviation.
median(*x***)**	median
quantile(*x***, *probs*)**	quantiles where x is the numeric vector whose quantiles are desired and probs is a numeric vector with probabilities in [0,1]. # 30th and 84th percentiles of x y <- quantile(x, c(.3,.84))
range(*x***)**	range
sum(*x***)**	sum
diff(*x***, lag=***1***)**	lagged differences, with lag indicating which lag to use
min(*x***)**	minimum
max(*x***)**	maximum
scale(*x***, center=**TRUE**, scale=**TRUE**)**	column center or standardize a matrix.

Other Useful Functions

Function	Description
seq(*from* **,** *to***,** *by***)**	generate a sequence indices <- seq(1,10,2) #indices is c(1, 3, 5, 7, 9)
rep(*x***,** *ntimes***)**	repeat *x n* times y <- rep(1:3, 2) # y is c(1, 2, 3, 1, 2, 3)
cut(*x***,** *n***)**	divide continuous variable in factor with *n* levels y <- cut(x, 5)

Sorting

- To sort a dataframe in R, use the **order( )** function. By default, sorting is ASCENDING. Prepend the sorting variable by a minus sign to indicate DESCENDING order. Here are some examples.
- # sorting examples using the mtcars dataset
  data(mtcars)
  # sort by mpg
  newdata = mtcars[order(mtcars$mpg),]
  # sort by mpg and cyl
  newdata <- mtcars[order(mtcars$mpg, mtcars$cyl),]
  #sort by mpg (ascending) and cyl (descending)
  newdata <- mtcars[order(mtcars$mpg, -mtcars$cyl),]

Merging

To merge two dataframes (datasets) horizontally, use the **merge** function. In most cases, you join two dataframes by one or more common key variables (i.e., an inner join).

```
# merge two dataframes by ID
total <- merge(dataframeA,dataframeB,by="ID")
# merge two dataframes by ID and Country
total <-
merge(dataframeA,dataframeB,by=c("ID","Country"))
```

Merging

ADDING ROWS

To join two dataframes (datasets) vertically, use the **rbind** function. The two dataframes **must** have the same variables, but they do not have to be in the same order.

total <- rbind(dataframeA, dataframeB)

If dataframeA has variables that dataframeB does not, then either:

Delete the extra variables in dataframeA or

Create the additional variables in dataframeB and set them to NA (missing)

before joining them with rbind.

Aggregating

- **It is relatively easy to collapse data in R using one or more BY variables and a defined function.**
- # aggregate dataframe mtcars by cyl and vs, returning means
  # for numeric variables
  attach(mtcars)
  aggdata <-aggregate(mtcars, by=list(cyl),
  FUN=mean, na.rm=TRUE)
  print(aggdata)
- OR use apply

Aggregating

- When using the aggregate() function, the by variables must be in a list (even if there is only one). The function can be built-in or user provided.
- See also:
- summarize() in the Hmisc package
- summaryBy() in the doBy package

Data Type Conversion

- **Type conversions in R work as you would expect. For example, adding a character string to a numeric vector converts all the elements in the vector to character.**
- **Use is.*foo* to test for data type *foo*. Returns TRUE or FALSE**
  Use as.*foo* to explicitly convert it.
- **is.numeric(), is.character(), is.vector(), is.matrix(), is.data.frame()**
  as.numeric(), as.character(), as.vector(), as.matrix(), as.data.frame)

Exploratory Data Analysis with R

Why we need statistics

- Everything varies
  - If you measure the same thing twice you will get two different answers
  - Heterogeneity is universal: spatial heterogeneity & temporal heterogeneity
  - We need to distinguish between variation that is scientifically interesting, and variation that just reflects background heterogeneity
- Significance (“statistically significant”)
  - A result is unlikely to have occurred by chance
  - A result is unlikely to have occurred by chance if the null hypothesis was true
  - Null hypothesis says that “nothing’s happening”, and the alternative says “something is happening”; null hypothesis has to be a falsifiable hypothesis

Given a sequence, what can we ask?

```
AGCTTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
TTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
TATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACC
ATTACCACCACCATCACCATTACCACAGGTAACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAG
CCCGCACCTGACAGTGCGGGCTTTTTTTTTCGACCAAAGGTAACGAGGTAACAACCATGCGAGTGTTGAA
GTTCGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCGATATTCTGGAAAGCAATGCC
AGGCAGGGGCAGGTGGCCACCGTCCTCTCTGCCCCCGCCAAAATCACCAACCACCTGGTGGCGATGATTG
AAAAAACCATTAGCGGCCAGGATGCTTTACCCAATATCAGCGATGCCGAACGTATTTTTGCCGAACTTTT
GACGGGACTCGCCGCCGCCCAGCCGGGGTTCCCGCTGGCGCAATTGAAAACTTTCGTCGATCAGGAATTT
GCCCAAATAAAACATGTCCTGCATGGCATTAGTTTGTTGGGGCAGTGCCCGGATAGCATCAACGCTGCGC
TGATTTGCCGTGGCGAGAAAATGTCGATCGCCATTATGGCCGGCGTATTAGAAGCGCGCGGTCACAACGT
```

- *What sort of statistics should be used to describe this sequence?*
- *Can we determine what sort of organism this sequence came from based on sequence content?*
- *Do parameters describing this sequence differ (significantly) from those describing bulk DNA in that organism?*
- *What sort of sequence might this be: protein coding? Transposable elements?*

Ref: Computational Genome Analysis, Deonier et al.

Goals

- Understand basic concepts
  - Exploratory data analysis (EDA)
    - Getting to know your data
    - Formulating hypotheses worth testing (boxplots, histograms, scatter plots, QQ-plot)
  - Confirmatory data analysis
    - Making decisions using experimental data; hypothesis testing (p-values, confidence intervals etc)
- Get to know the R statistical language

EDA techniques

- Mostly graphical (a clear picture is worth a thousand words!)
- Plotting the raw data (histograms, scatter plots, etc.)
- Plotting simple statistics such as means, standard deviations, medians, box plots, etc

Knowing your data

- Types of your data
- Central tendency
  - Mode: The data values that occur most frequently are called the **mode** (drawing a histogram of the data)
  - Arithmetic mean: $\bar{a}=\Sigma a / n$
    - sum(a) / length(a)
  - Median: the “middle value” in the data set
    - sort(y)[ceiling(length(y)/2)]
- Variance
  - Degrees of freedom (d.f.)

Probability distribution

law	function
Gaussian (normal)	`rnorm(n, mean=0, sd=1)`
exponential	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor (F)	`rf(n, df1, df2)`
Pearson (χ^2)	`rchisq(n, df)`
binomial	`rbinom(n, size, prob)`
geometric	`rgeom(n, prob)`
hypergeometric	`rhyper(nn, m, n, k)`
logistic	`rlogis(n, location=0, scale=1)`
lognormal	`rlnorm(n, meanlog=0, sdlog=1)`
negative binomial	`rnbinom(n, size, prob)`
uniform	`runif(n, min=0, max=1)`
Wilcoxon's statistics	`rwilcox(nn, m, n), rsignrank(nn, n)`

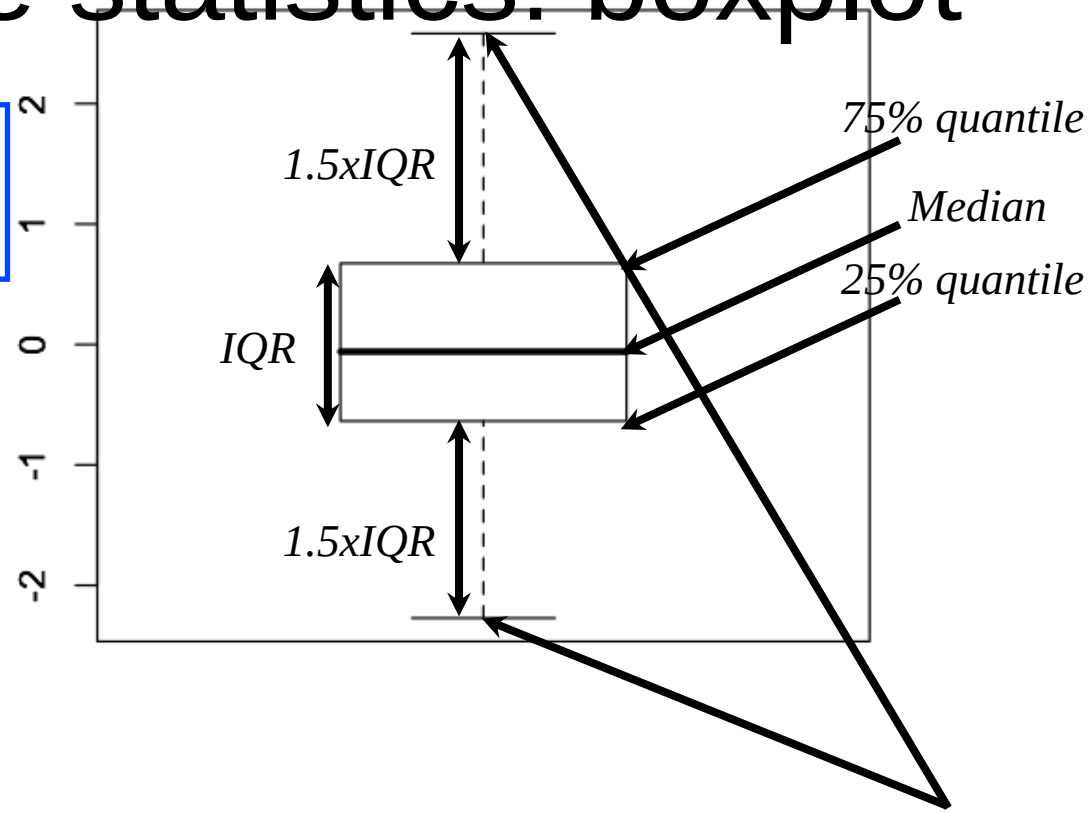
Descriptive statistics

summary statistics for ‘quantifying’ a data set (mean, median, variance, standard deviation)

```
set.seed(100)
x<-rnorm(100, mean=0, sd=1)
mean(x)
median(x)
IQR(x)
var(x)
summary(x)
```

Descriptive statistics: boxplot

```
set.seed(100)
x<-rnorm(100, mean=0, sd=1)
boxplot(x)
```



IQR (interquantile range)= 75% quantile -25% quantile

Everything above or below are considered outliers

P-quantile

<u>(Theoritical) Quantiles</u>: The p-quantile is the value with the property that there is a probability p of getting a value less than or equal to it.

<u>Empirical Quantiles</u>: The p-quantile is the value with the property that p% of the observations are less than or equal to it.

Empirical quartiles can easily be obtained in R.

```
set.seed(100)
x<-rnorm(100, mean=0, sd=1)
quantile(x)
```

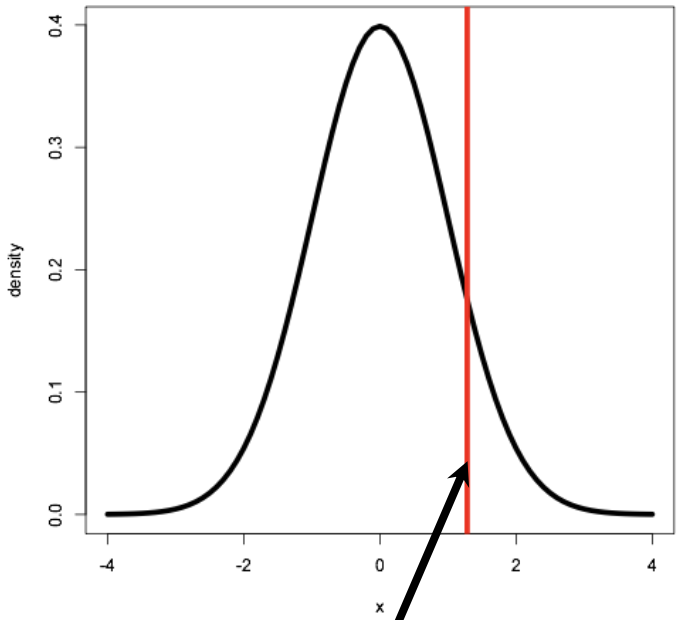
```
       0%       25%       50%       75%      100%
-2.2719255 -0.6088466 -0.0594199  0.6558911  2.5819589
```

The 50% quantile -> median

More on quantiles

quartile (divided into 4 groups)
decile (10 groups)
percentage (100 groups).

```
q90<-qnorm(.90, mean = 0, sd = 1)
#q90 -> 1.28
x<-seq(-4,4,.1)
f<-dnorm(x, mean=0, sd=1)
plot(x,f,xlab="x",ylab="density",type="l",lwd=5)
abline(v=q90,col=2,lwd=5)
```



90% of the prob. (area under the curve) is on the left of red vertical line.

QQ-plot

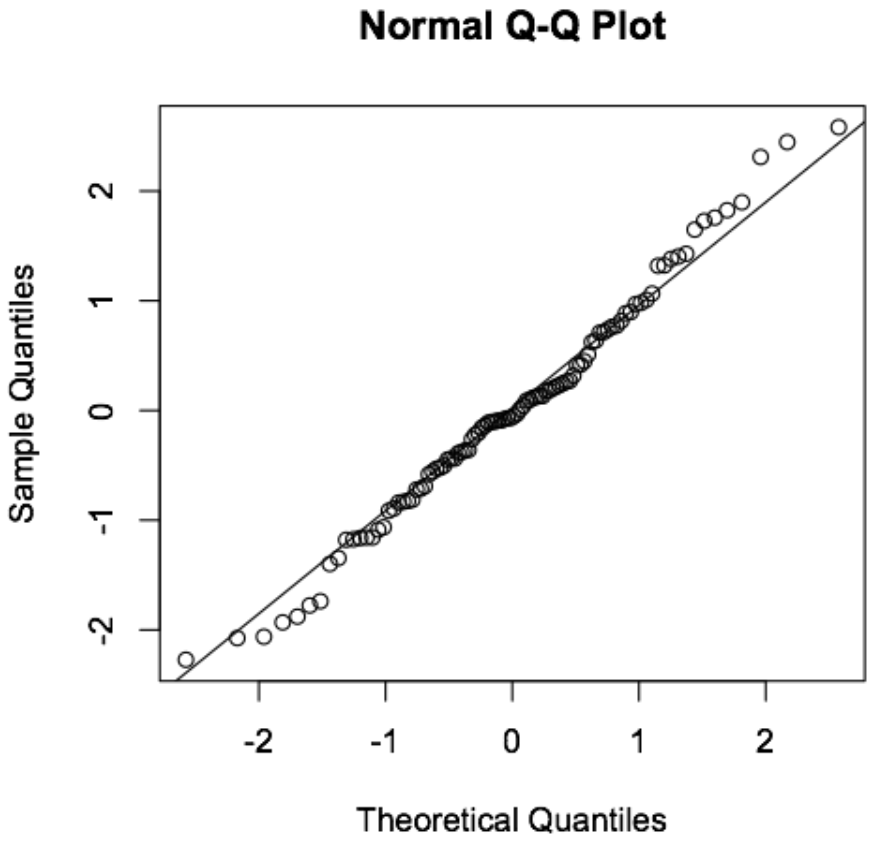
- *Many statistical methods make some assumption about the distribution of the data (e.g. normal).*

- *The quantile-quantile plot provides a way to visually verify such assumptions.*

- *The QQ-plot shows the theoretical quantiles versus the empirical quantiles. If the distribution assumed (theoretical one) is indeed the correct one, we should observe a straight line.*

QQ-plot

```
set.seed(100)
x<-rnorm(100, mean=0, sd=1)
qqnorm(x)
qqline(x)
```



Statistical tests

	Data type			
Goal	Measurement (from **Gaussian** Population)	Rank, Score, or Measurement (from **Non-Gaussian** Population)	Binomial (Two Possible Outcomes)	Survival time
Describe one group	Mean, SD	Median, interquartile range	Proportion	Kaplan Meier survival curve
Compare one group to a hypothetical value	One-sample t test	Wilcoxon test	Chi-square or Binomial test	
Compare two unpaired groups	Unpaired t test	Mann-Whitney test	Fisher's test (chi-square for large samples)	Log-rank test or Mantel-Haenszel
Compare two paired groups	Paired t test	Wilcoxon test	McNemar's test	Conditional proportional hazards regression

http://www.graphpad.com/www/Book/Choose.htm

P value

- A p value is an estimate of the probability of a particular result, or a result more extreme than the result observed, could have occurred by chance, if the null hypothesis were true. The null hypothesis says ’nothing is happing’.
- For example, if you are comparing two means, the null hypothesis is that the means of the two samples are the same.
- The p value is a measure of the credibility of the null hypothesis. If something is very unlikely to happen, we say that it is statistically significant.

Multiple testing problem and q value

- When we set a p-value threshold of, for example, 0.05, we are saying that there is a 5% chance that the result is a false positive.
- While 5% is acceptable for one test, *if we do lots of tests on the data, then this 5% can result in a large number of false positives*. (e.g., 200 tests result in 10 false positives by chance alone). This is known as the *multiple testing problem*.
- Multiple testing correlations adjust p-values derived from statistical testing to control the occurrence of false positives (i.e., the false discovery rate). The *q value is a measure of significance in terms of the false discovery rate (FDR)* rather than the false positive rate.
- Bonferroni correction (too conservative)

Parametric/nonparametric tests

- Choose a parametric test if you are sure that your data are sampled from a population that follows a Gaussian distribution (at least approximately) (e.g., t test, Fisher test).
- You should definitely select a nonparametric test in three situations:
  - The outcome is a rank or a score and the population is clearly not Gaussian
  - Some values are "off the scale," that is, too high or too low to measure
  - The data ire measurements, and you are sure that the population is not distributed in a Gaussian manner
- large data sets present no problems: The central limit theorem ensures that parametric tests work well with large samples even if the population is non-Gaussian.

Statistical modeling

- It is not “the data is fitted to a model”; rather, it is “the model is fitted to the data”
- To determine a minimal adequate model from the large set of potential models that might be used to describe the given set of data
- The object is to determine the values of the parameters in a specific model that lead to the best fit of the model to the data.
- We define the “best” model in terms of maximum likelihood
  - Given the data
  - And given our choice of modle
  - What values of the parameters of that model make the observed data most likely?

(Generalized) linear models

- The model formulae look very like equations but there are important differences

  y = a + bx (formula: y ~ x)

  y = a + bx + cz (formula: y ~ x + z)

- Fitting linear models

  fm2 <- lm(y ~ x1 + x2, data = production)

- Generalized Linear Models

  glm(y ~ z, family = poisson)

  glm(y ~ z, family = binomial)

Useful R/BioConductor packages

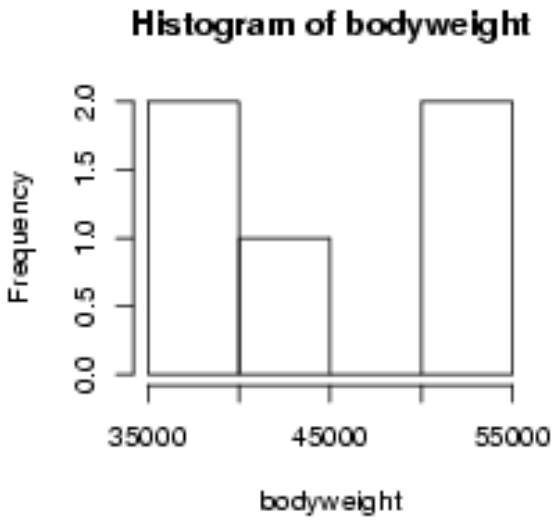
Marray, limma	Spotted cDNA array analysis
affy	Affymetrix array analysis
vsn	Variance stabilization
annotate	Link microarray data to metadata on the web
ctest	Statistical tests
genefilter, limma, multtest, siggenes	Gene filtering (e.g.: differential expression)
mva, cluster, clust	Clustering
class, rpart, nnet	Classification

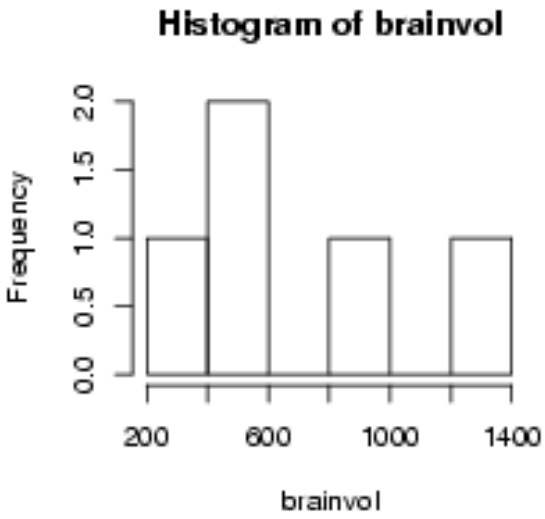
Example 1: Primate's body weight & brain volume

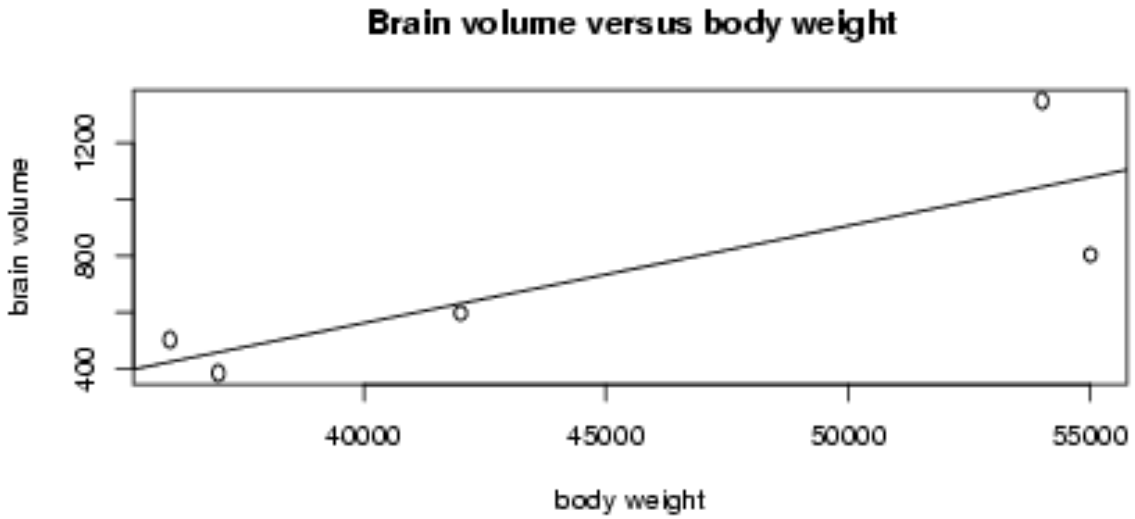
primate.dat

	bodyweight	brainvol
H.sapiens	54000	1350
H.erectus	55000	804
H.habilis	42000	597
A.robustus	36000	502
A.afarensis	37000	384

- *Summary of the data (bodyweight, and brainvol)*
- *Correlation between bodyweight and brainvol*
- *Linear fitting*
- *Plotting*

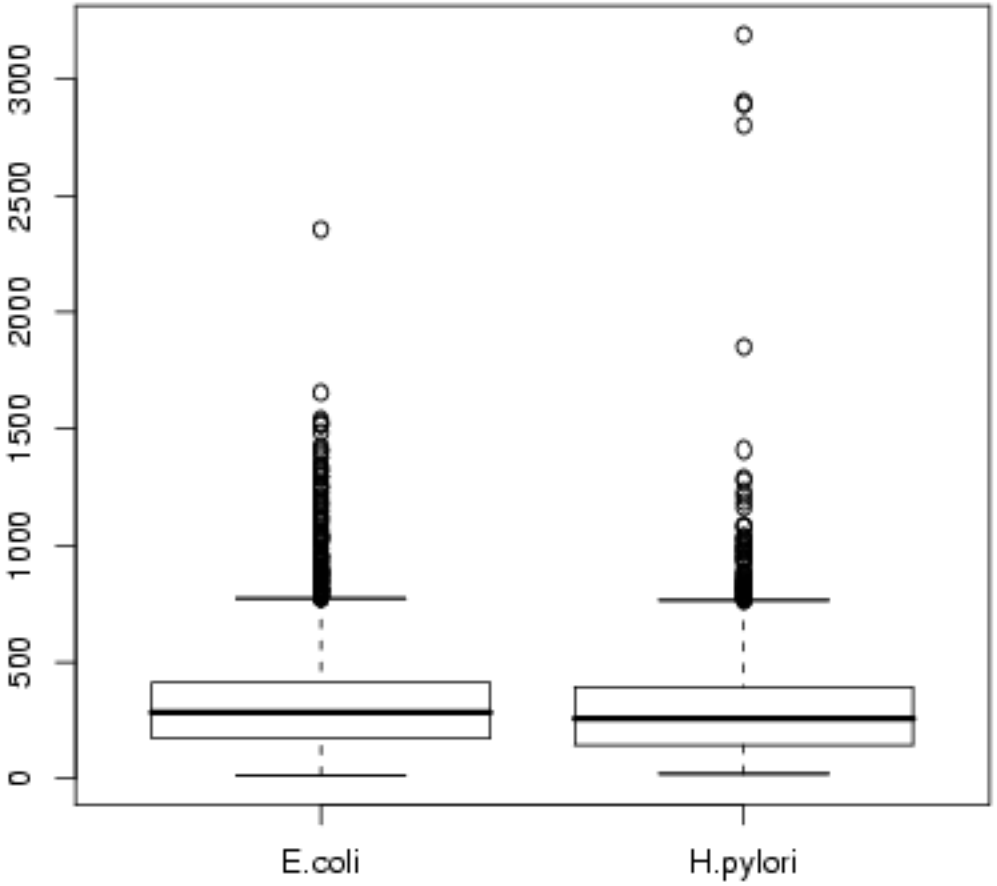

Histogram of bodyweight
Frequency
0.0
0.5
1.0
1.5
2.0
35000
45000
55000
bodyweight


Histogram of brainvol
Frequency
0.0
0.5
1.0
1.5
2.0
200
600
1000
1400
brainvol


Brain volume versus body weight
brain volume
400
800
1200
40000
45000
50000
55000
body weight

Example 2: Gene length

- Do the protein-coding genes in *E.coli* and *H.pylori* Genomes have statistically different gene lengths?



Bioconductor: Biostrings

- Install:
  - source("http://bioconductor.org/biocLite.R")
  - biocLite("Biostrings")
- Example 1: alignment of two DNA sequences
  - library(Biostrings)
  - s1 <- DNAString("GGGCCC"); s2 <- DNAString("GGGTTCCC")
  - aln <- pairwiseAlignment(s1, s2, type="global")
- Example 2: alignment of two protein sequences
  - s1 <- AAString("STSAMVWENV")
  - s2 <- AAString("STTAMMEDV")
  - pairwiseAlignment(s1, s2, type="global", substitutionMatrix="BLOSUM62", gapOpening=-11, gapExtension=-1)

Sample 1: T-test

- Generate two datasets X and Y;
- Do the Shapiro-Wilk normality test;
- Do the t-test
  - Alternative: two sided; less; greater;
  - Paired or not;
  - Confidence interval.

Sample 2: Linear Regression

- A comparison of GM monthly returns & SP500 monthly returns. GM and SP500 monthly return data during the period of Jan. 2002 to Jun. 2007 are taken. Plotted in R, they will be analyzed and compared.

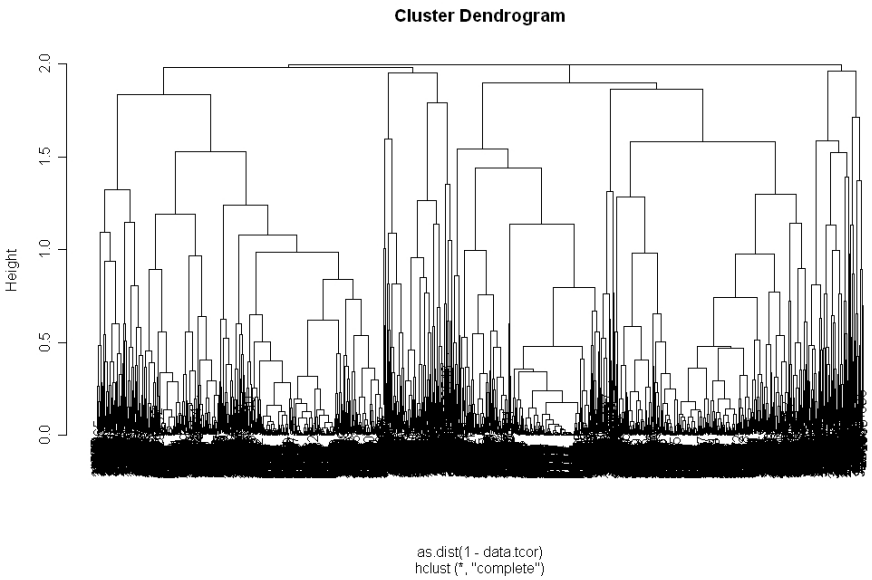
- Data from: http://www.stanford.edu/~xing/statfinbook/data.html

Sample 2: Linear Regression

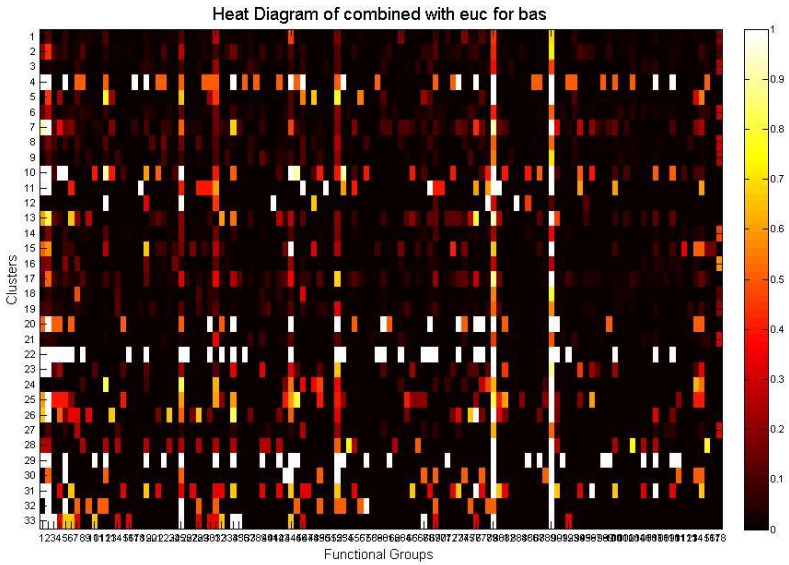
```
GM<-read.table("C:/GM.txt", header=TRUE, sep="")
SP<-read.table("C:/SP.txt", header=TRUE, sep="")
plot(GM)
lines(GM$logret, lty=1) #connect the plots with solid line
lines(SP$logret, type="o", lty=1, pch="+", col="red")
#connect with different mark and color
x<-1:66  # x here is the time step
GML<-lm(GM$logret~x) # linear regression coresponding to time
SPL<-lm(SP$logret~x)
abline(coef(GML), lwd=3)  #abline gives the reg line
abline(coef(SPL), col="red", lwd=3)
  #lwd gives the line width
```

Sample 3:
Neuron Data Study

Hierarchical Clustering

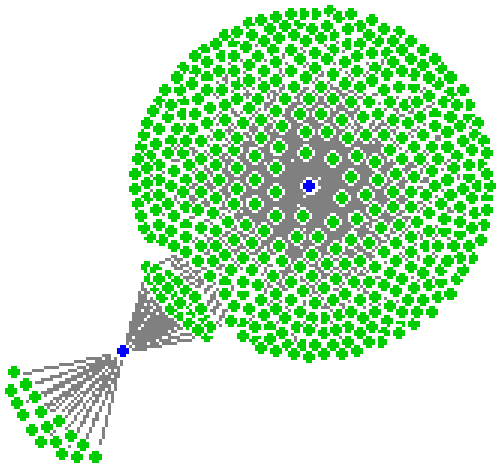


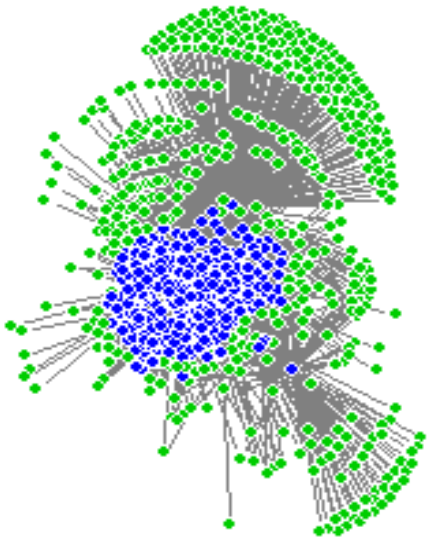
Heat diagram



Sample 3: Neuron Data Study

Partial Correlation Network among genes in PKJ and BAS cells





VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!

soict.hust.edu.vn/ fb.com/groups/soict