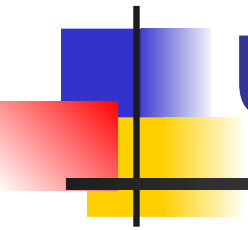
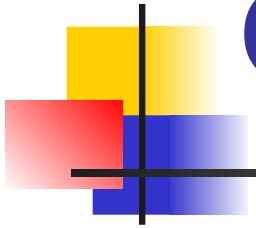
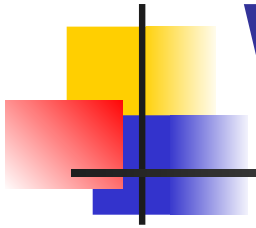
Lecture 2: R Basics(Data analysis using R)





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?

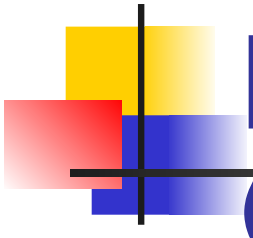
It's free!

It runs on a variety of platforms including Windows, Unix and MacOS.

It provides an unparalleled platform for programming new statistical methods in an easy and straightforward manner.

It contains advanced statistical routines not yet available in other packages.

It has state-of-the-art graphics capabilities.



R has a Steep Learning Curve (steeper for those that knew SAS or other software before)

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

R has a Learning Curve

(steeper for those that knew SAS or other software before)

The **second** reason is more transient. As users of statistical packages, we tend to run one controlled procedure for each type of analysis. Think of PROC GLM in SAS. We can carefully set up the run with all the parameters and options that we need. When we run the procedure, the resulting output may be a hundred pages long. We then sift through this output pulling out what we need and discarding the rest.

R paradigm is different

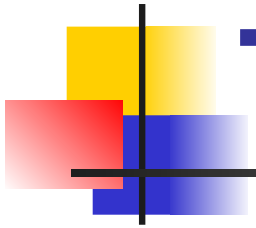
Rather than setting up a complete analysis at once, the process is highly interactive. You run a command (say fit a model), take the results and process it through another command (say a set of diagnostic plots), take those results and process it through another command (say cross-validation), etc. The cycle may include transforming the data, and looping back through the whole process again. You stop when you feel that you have fully analyzed the data.

How to download?

- Google it using R or CRAN

(Comprehensive R Archive Network)

- http://www.r-project.org



Tutorials

Each of the following tutorials are in PDF format.

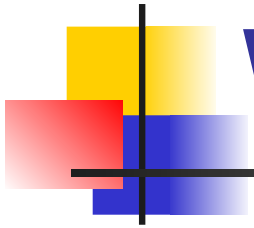
- P. Kuhnert & B. Venables,
  An Introduction to R: Software for Statistical Modeling & Computing

- 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 (c/o Rob Kabacoff) **MOST of these notes follow this web page format**
- Introduction to R by Vincent Zoonekynd
- R Cookbook
- Data Manipulation Reference

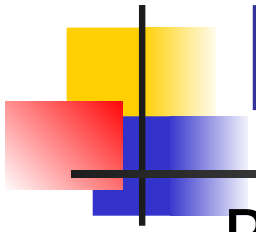


Web links

- R time series tutorial
- R Concepts and Data Types presentation by Deepayan Sarkar
- Interpreting Output From lm()
- The R Wiki
- An Introduction to R
- Import / Export Manual
- R Reference Cards

Web links

- 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)



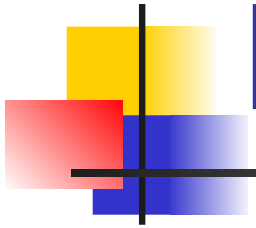
R Overview

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.



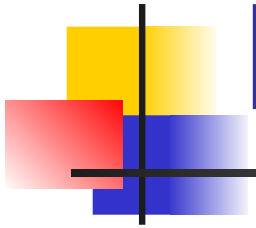
R Overview

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, dataframes, and lists.

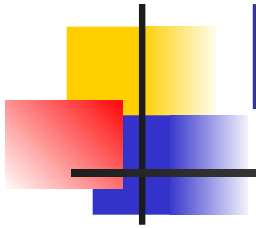
To quit R, use

>q()



R Overview

- 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



R Overview

- 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.

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.

Your First R Session

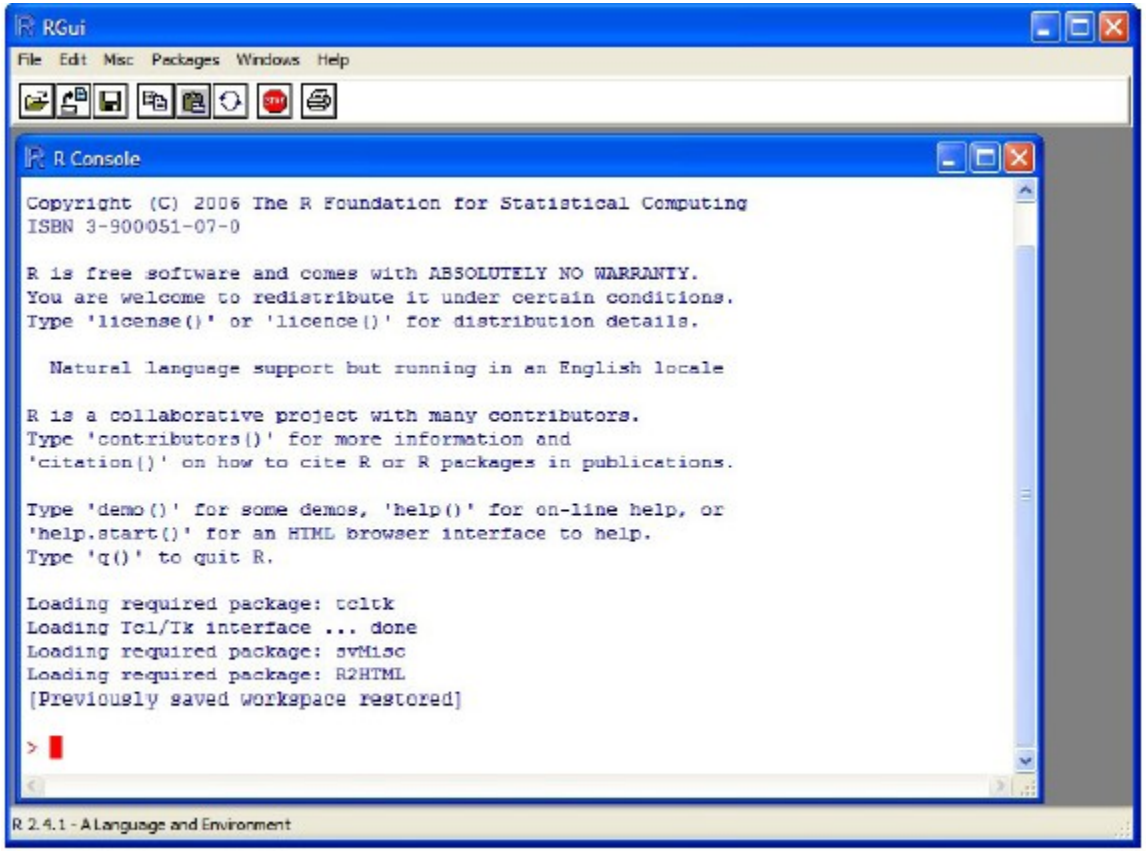


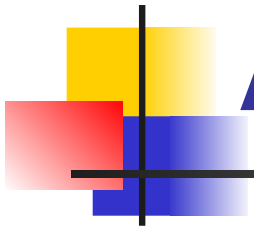
Figure 1.1: The R system on Windows

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 (=).

R Introduction

- 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 and aclosing). 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"
```

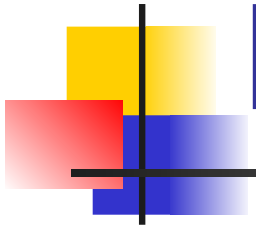
R Introduction

- 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")
```

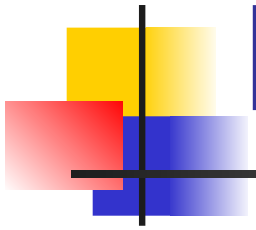


R Warning !

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

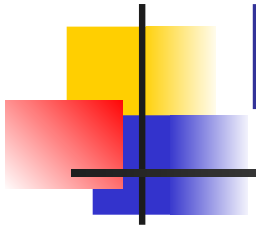
R Introduction

```
> 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
```



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.



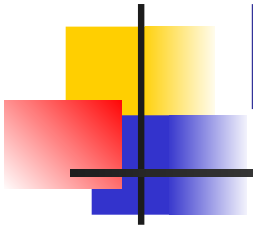
R Workspace

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")
```



R Workspace

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 le, 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 Workspace

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")

R Workspace

#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

R Workspace

save your command history
savehistory(file="*myfile*") # default is ".Rhistory"

recall your command history
loadhistory(file="*myfile*") # default is ".Rhistory“

R Help

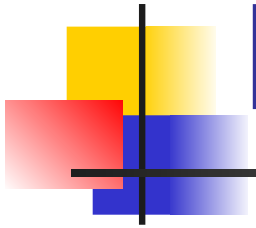
Once **R** is installed, there is a comprehensive built-in help system. At the program's command prompt you can use any of the following:

```
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
```

R Help

```
# 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
```



R Datasets

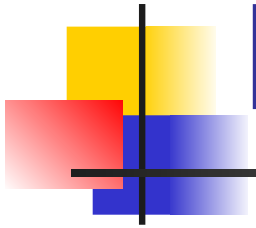
R comes with a number of sample datasets that you can experiment with. Type

> 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.



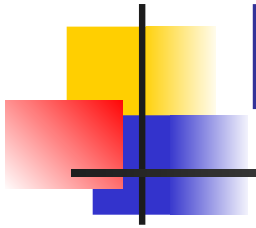
R Packages

- 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
```

R Packages

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

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.

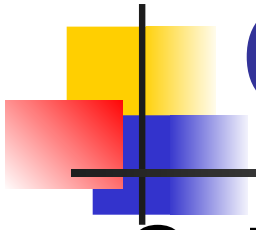
Source Codes

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")
```



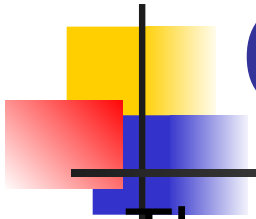
Output

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()
```



Output

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)
```

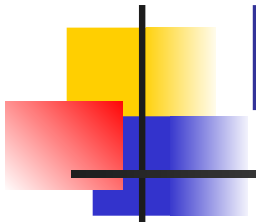
Graphs

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.

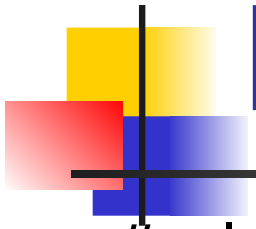
Reusing Results

```
# 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)
```