

Introduction to R

Introduction to R

- 1.1 Installing R
- 1.2 Command Line Interface
- 1.3 The Assignment operator
- 1.4 Commenting
- 1.5 Defining Variables
- 1.6 Help Functions
- 1.7 The `help.start()` command
- 1.8 Basic Maths Operations
- 1.9 Basic R Editor

Introduction to R

- 1.10 Built-In Data Sets
- 1.11 The `summary()` command
- 1.12 Working directories
- 1.13 Coming Unstuck
- 1.14 Quitting the R environment
- 1.15 Data Objects
- 1.16 Listing all items in a workspace
- 1.17 Removing items
- 1.18 Saving and Loading R Data Objects

Introduction to R (Continued)

- 2.1 Three particularly useful commands
- 2.2 Changing GUI options
- 2.3 Colours
- 2.4 Use of the Semi-Colon Operator
- 2.5 The `apropos()` Function
- 2.6 History
- 2.7 The `sessionInfo()` Function
- 2.8 Time and date functions
- 2.9 Logical States
- 2.10 Missing Data
- 2.11 Files in the Working Directory

Section 3 - Inspecting a Data Set

3.1 Dimensions of a data set

3.2 The `summary()` command

3.3 Structure of a Data Object

Introduction to R

Source: R project website (<http://www.r-project.org>)

- ▶ R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues.
- ▶ R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

What is R?

- ▶ R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.
- ▶ One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed.
- ▶ Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.
- ▶ R is available as Free Software under the terms of the Free Software Foundations GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

R is a programming environment that

- ▶ uses a well-developed but simple programming language
- ▶ allows for rapid development of new tools according to user demand
- ▶ these tools are distributed as packages, which any user can download to customize the R environment.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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

[Previously saved workspace restored]

> |

```

>
>
> # Rabbit Data Set in MASS Package
> summary(Rabbit)

```

BPchange	Dose	Run	Treatment	Animal
Min. : 0.50	Min. : 6.25	C1 : 6	Control:30	R1:12
1st Qu.: 1.65	1st Qu.: 12.50	C2 : 6	MDL :30	R2:12
Median : 4.75	Median : 37.50	C3 : 6		R3:12
Mean :11.22	Mean : 65.62	C4 : 6		R4:12
3rd Qu.:20.50	3rd Qu.:100.00	C5 : 6		R5:12
Max. :37.00	Max. :200.00	M1 : 6		
		(Other):24		

```

> |

```

```
>
>
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length"
[4] "Petal.Width"  "Species"
>
> class(iris)
[1] "data.frame"
>
> mode(iris)
[1] "list"
> |
```

```
> summary(iris)
      Sepal.Length      Sepal.Width      Petal.Length      Petal.Width
Min.      :4.300      Min.      :2.000      Min.      :1.000      Min.      :0.100
1st Qu.:5.100      1st Qu.:2.800      1st Qu.:1.600      1st Qu.:0.300
Median :5.800      Median :3.000      Median :4.350      Median :1.300
Mean   :5.843      Mean   :3.057      Mean   :3.758      Mean   :1.199
3rd Qu.:6.400      3rd Qu.:3.300      3rd Qu.:5.100      3rd Qu.:1.800
Max.   :7.900      Max.   :4.400      Max.   :6.900      Max.   :2.500

      Species
setosa      :50
versicolor:50
virginica   :50
```

Figure:

Comprehensive R Archive Network

- ▶ Base R and most R packages are available for download from the Comprehensive R Archive Network (CRAN)
cran.r-project.org.
- ▶ Base R comes with a number of basic data management, analysis, and graphical tools
- ▶ R's power and flexibility, however, lie in its array of packages (currently more than 6,000!)

1.1 Installing R

- ▶ R is very easily installed by downloading it from the CRAN website. Installation usually takes about 2 minutes.
- ▶ When Installation of R is complete, the distinctive R Icon will appear on your desktop. To start R, simply click this Icon.
- ▶ It is common to re-install R once a year or so. The current version of R, version 3.1.2 was released quite recently.



R Console



```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"  
Copyright (C) 2014 The R Foundation for Statistical Computing  
Platform: i386-w64-mingw32/i386 (32-bit)
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.
```

```
  Natural language support but running in an English locale
```

```
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.
```

1.2 Command Line Interface

- ▶ When you start R, the command line interface window will appear on screen. This is one of many windows in the R environment, others including graphical output windows, or script editors.
- ▶ R code can be entered into the command line directly.
- ▶ Alternatively code can be saved to a script, which can be run inside a session using the `source()` function.

1.3 The Assignment operator

- ▶ The assignment operator is used to assign names to particular values.
- ▶ Historically the assignment operator was) a “<-”.
- ▶ The assignment operator can also be the equals sign “=”. (This is valid as of R version 1.4.0.)
- ▶ Both will be used, although, you should learn one and stick with it. Many long term R users prefer the arrow approach.

1.3 The Assignment operator

You can also use `<-` as an assignment operator, reversing the usual assignment positions. (This is actually really useful). Commands are separated either by a semi colon or by a newline.

```
>  
> a <- 6  
> b = 5  
> a + b -> c  
> c  
[1] 11  
>  
> e=7;f<-5;g=sqrt(c)  
>  
> # Broke a Rule!!  
> |
```

Figure:

1.3 The Assignment operator

The Up and Down Keys

- ▶ Before we continue, try using the up and down keys, and see what happens.
- ▶ Previously typed commands are re-presented, and can be re-executed.

1.3 The Assignment operator

objects

- ▶ R stores both data and output from data analysis (as well as everything else) in **objects**.
- ▶ The variables we have created so far are objects.
- ▶ A list of all objects in the current session can be obtained with `ls()`.

1.3.1 Reserved Words - Bad names for Objects

- ▶ Some names are used by the system, e.g. T, F, q, c etc .
- ▶ Avoid using these.
- ▶ Also avoid using command names like **mean** and **sum**

```
>  
>  
> as.character(X)  
[1] "4" "5" "6" "7"  
> as.factor(X)  
[1] 4 5 6 7  
Levels: 4 5 6 7  
> |
```

```
> X <- c(4,5,6,7)
> mode(X)
[1] "numeric"
> class(X)
[1] "numeric"
>
> Y <- c(4L,5L,6L,7L)
> mode(Y)
[1] "numeric"
> class(Y)
[1] "integer"
> |
```

1.4 Commenting

For the sake of readability, it is good practice to comment code. The `#` character at the beginning of a line signifies a comment, which is not executed. Lines of hashtags can be used to identify the beginning and end of code segments

```
<
> # This is a comment
> #####
> # Start of Segment 1
> #####
> #####
> |
```

Figure:

1.5 Defining and Naming Variables

- ▶ A convention is to use define a variable name with a capital letter (R is case sensitive).
- ▶ This reduces the chance of overwriting in-build R functions, which are usually written in lowercase letters.
- ▶ Commonly used variable names such as x,y and z (in lower case letters) are not reserved.

```
camelCase  
variableName  
AlsoCamelCase
```

File Edit Windows

R Console

```

>
>
> # Rabb
> summar # Such
BPck
Min. A <- 6
1st Qu.
Median B <- 7
Mean
3rd Qu. A + B
Max.
> |

```

Untitled - R Editor

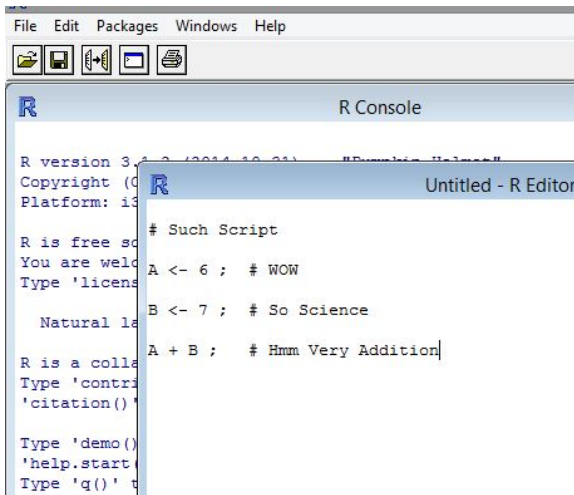
```

# Such
A <- 6
B <- 7
A + B

```

R data sets

Cities
Quarterly Approval Ratings of Presidents
Vapor Pressure of Mercury as Function of Temperature
Locations of Earthquakes off Fiji
Random Numbers from Conway

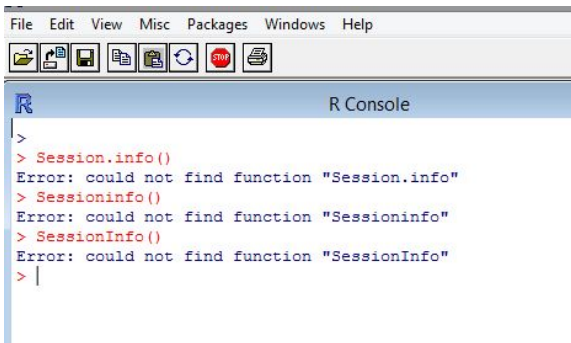


1.6 Help Functions

- ▶ Help files for R functions are accessed by preceding the name of the function with ? (e.g. ?sort).
- ▶ Alternatively you can use the command `help()` (e.g. `help(sqrt)`)

1.6 Help Functions

- ▶ A HTML document appears on screen with information on the function typed in.
- ▶ As well as the list of arguments that the particular function accepts, and how to specify them, there is example code at the bottom of the file.
- ▶ These code segments are often invaluable in learning how to master the function.



The screenshot shows the R Console window with the following menu bar: File, Edit, View, Misc, Packages, Windows, Help. Below the menu bar is a toolbar with icons for file operations (open, save, print, etc.) and a red stop button. The console area has a blue header with the R logo and the text "R Console". The console output shows the following sequence of commands and errors:

```
>  
> Session.info()  
Error: could not find function "Session.info"  
> Sessioninfo()  
Error: could not find function "Sessioninfo"  
> SessionInfo()  
Error: could not find function "SessionInfo"  
> |
```

```
>  
> apropos("essio")  
[1] ".__C__expression"      "as.expression"  
[3] "as.expression.default" "expression"  
[5] "is.expression"         "sessionInfo"  
[7] "setSessionTimeLimit"  
> |
```

1.7 The `help.start()` command

As mentioned by the text on the main console, the `help.start()` command can be used to access detailed help documentation that comes as part of the installation.

1.8 Basic Maths Operations

The most commonly used mathematical operators are all supported by R.

Here are a few examples:

$5 + 3 * 5$	Note the order of operations.
<code>log (10)</code>	Natural logarithm with base $e=2.718282$
<code>log(8,2)</code>	Log to the base 2
4^2	4 raised to the second power
$7/2$	Division
<code>factorial(4)</code>	Factorial of Four
<code>sqrt (25)</code>	Square root
<code>abs (3-7)</code>	Absolute value of 3-7
<code>pi</code>	The mysterious number
<code>exp(2)</code>	exponential function

R can be used for many mathematical operations, including

- ▶ Set Theory
- ▶ Trigonometry
- ▶ Complex Numbers
- ▶ Binomial Coefficients

Set Theory is always useful to know. We will not go into any of the others in great detail today.

1.9 Basic R Editor

R has an inbuilt script editor. We will use it for this class, but there are plenty of top quality Integrated Development Environments out there. (Read up about RStudio for example). For a while, we will use the in-built script editor. Although it is advisable to start using Rstudio or something similar.

To start a new script, or open an existing script simply go to File and click the appropriate options. A new dialogue box will appear. You can write and edit code using this editor. To pass the code for compiling , press the run line or selection option (The third icon on the menu).

Another way to read code is to use the `edit()` function , which operates directly from the command line. To see how the code defining an object `X` was written (or more importantly, could have been written) simply type `edit(X)`. This command has some useful applications that we will see later on.

Script Files

- ▶ Scripts are saved as .R files.
- ▶ These scripts can be called directly using the `source()` command.

1.10 Built-In Data Sets

Inbuilt Data Sets

Several data sets , intended as learning tools, are automatically installed when R is installed. Many more are installed within packages to complement learning to use those packages. One of these is the famous Iris data set, which is used in many data mining exercises.

- ▶ iris
- ▶ mtcars
- ▶ Nile

R

R Console

>

>

> data()

> |

R

R data sets

Data sets in package 'datasets':

AirPassengers

Monthly Airline Passenger
Numbers 1949-1960

BJsales

Sales Data with Leading
Indicator

BJsales.lead (BJsales)

Sales Data with Leading
Indicator

BOD

Biochemical Oxygen Demand

<

□

>

<

□

>

<

≡

>

<

≡

>

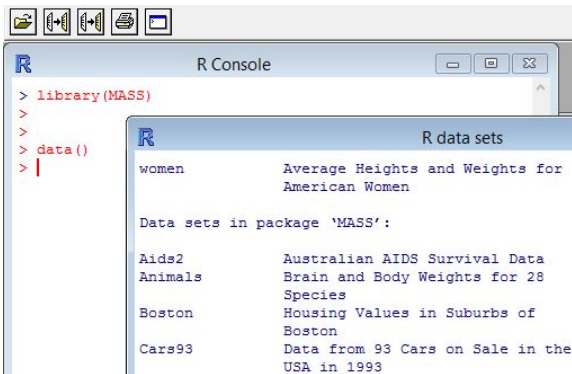
≡

↺

↻

↺

↻



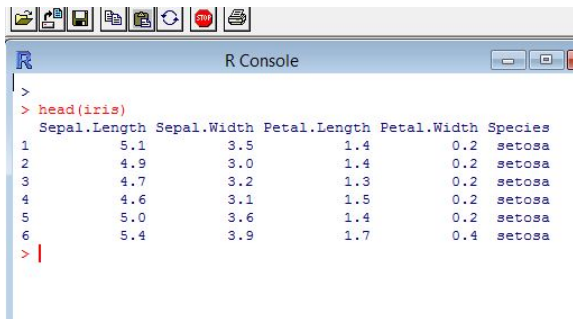
The image shows the R Studio interface. At the top, there is a toolbar with icons for file operations (new, open, save, print, etc.). Below the toolbar, the main window is divided into two panes. The left pane is the 'R Console', which contains the following text:

```
> library(MASS)
>
>
> data()
> |
```

The right pane is titled 'R data sets' and displays a list of data sets available in the 'MASS' package. The list is as follows:

Dataset Name	Description
women	Average Heights and Weights for American Women
Data sets in package 'MASS':	
Aids2	Australian AIDS Survival Data
Animals	Brain and Body Weights for 28 Species
Boston	Housing Values in Suburbs of Boston
Cars93	Data from 93 Cars on Sale in the USA in 1993

- ▶ To see what data sets are available, simply type `data()`.
- ▶ To load a data set, simply type in the name of the data set. Some data sets are very large.
- ▶ To just see the first few (or last) rows, we use the `head()` function or alternatively the `tail()` function.
- ▶ The default number of rows of these commands is 6. Other numbers can be specified.



The image shows a screenshot of an R Console window. The window has a title bar with the R logo and the text "R Console". Below the title bar is a toolbar with icons for file operations (open, save, print, etc.) and a red stop button. The main area of the window displays the following text:

```
>  
> head(iris)  
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
1          5.1           3.5          1.4           0.2  setosa  
2          4.9           3.0          1.4           0.2  setosa  
3          4.7           3.2          1.3           0.2  setosa  
4          4.6           3.1          1.5           0.2  setosa  
5          5.0           3.6          1.4           0.2  setosa  
6          5.4           3.9          1.7           0.4  setosa  
> |
```

The output is a table with 6 rows and 6 columns. The columns are labeled "Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", and "Species". The rows are numbered 1 through 6. The "Species" column contains the word "setosa" for all rows. The "Petal.Width" column contains values 0.2 for rows 1-5 and 0.4 for row 6.

```
✓  
> tail(iris)  
      Sepal.Length Sepal.Width Petal.Length Petal.Width  Sp  
145          6.7         3.3         5.7         2.5 virg  
146          6.7         3.0         5.2         2.3 virg  
147          6.3         2.5         5.0         1.9 virg  
148          6.5         3.0         5.2         2.0 virg  
149          6.2         3.4         5.4         2.3 virg  
150          5.9         3.0         5.1         1.8 virg  
> |
```

- ▶ In many situations, it is useful to call a particular data set using the `attach()` command. This will save having to specify the data sets over repeated operations.
- ▶ The file can then be detached using the `detach()` command.

1.11 The summary() command

The R command `summary()` is a generic function used to produce result summaries of the results of various objects, from simple vectors to the output of complex model fitting functions. The function invokes particular methods which depend on the class of the first argument.

```
summary(Nile)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
456.0	798.5	893.5	919.4	1032.0	1370.0

λ summary(Indometh) Subject time conc 1:11 Min. :0.250 Min.
:0.0500 4:11 1st Qu.:0.750 1st Qu.:0.1100 2:11 Median :2.000
Median :0.3400 5:11 Mean :2.886 Mean :0.5918 6:11 3rd
Qu.:5.000 3rd Qu.:0.8325 3:11 Max. :8.000 Max. :2.7200

1.12 Working directories You can change your working directory by using the appropriate options on the File menu. To determine the current working directory, you can use the `getwd()` command. To change the working directory, we would use the `setwd()` command. This is quite important as objects will be imported and exported to and from the specified directory.


```
> getwd() [1] "C:/Users/Kevin" > >  
setwd("C:/Users/Kevin/Documents") > > getwd() [1]  
"C:/Users/Kevin/Documents"
```

1.13 Coming Unstuck

- ▶ If you are having trouble with a piece of code that is currently compiling , all you have to do is press ESC, just like many other computing environments.

1.14 Quitting the R environment

As the front page text indicates, all you have to do to quite the workspace is to type in `q()`. You will then be prompted to save your work.

1.15 Data Objects

As mentioned previously, R saves data as **objects**. Examples of data objects are

- ▶ Vectors
- ▶ Lists
- ▶ Dataframes
- ▶ Matrices

The simple objects we have created previously are simply single element vectors.

1.16 Listing all items in a workspace

To list all items in an R environment, we use the `ls()` function. This provides a list of all data objects accessible. Another useful command is `objects()`.

```
>
> ls()
[1] "a"          "alt"        "bp"         "Charvec"    "co2"
[6] "dyp"        "limits"     "lp"         "msleep"     "mtcars"
[11] "Numvec"     "p"          "sumd"       "X"          "X1"
[16] "X2"         "Y"
> |
```

Figure:

1.17 Removing items

- ▶ Sometimes it is desirable to save a subset of your workspace instead of the entire workspace.
- ▶ One option is to use the `rm()` function to remove unwanted objects right before exiting your R session; another possibility is to use the `save()` function.

1.18 Saving and Loading R Data Objects

In situations where a good deal of processing must be used on a raw dataset in order to prepare it for analysis, it may be prudent to save the R objects you create in their internal binary form. One attractive feature of this scheme is that the objects created can be read by R programs running on different computer architectures than the one on which they were created, making it very easy to move your data between different computers. Each time an R session is completed, you are prompted to save the workspace image, which is a binary file called `.RData` in the working directory.

Whenever R encounters such a file in the working directory at the beginning of a session, it automatically loads it making all your saved objects available again. So one method for saving your work is to always save your workspace image at the end of an R session. If you would like to save your workspace image at some other time during your R session, you can use the `save.image()` function, which, when called with no arguments, will also save the current workspace to a file called `.RData` in the working directory.

2.1 Three particularly useful commands

- ▶ `help()`
- ▶ `summary()`
- ▶ `help.start()`

2.2 Changing GUI options

- ▶ We can change the GUI options using the GUI preferences option on the Edit menu.
- ▶ (Important when teaching R)
- ▶ A demonstration will be done in class.

2.3 Colours

- ▶ R supported a massive number of colours.
- ▶ Type in `colours()` (or `colors()`) to see what colours are supported.

```
>
> colours()
[1] "white"
[3] "antiquewhite"
[5] "antiquewhite2"
[7] "antiquewhite4"
[9] "aquamarine1"
[11] "aquamarine3"
[13] "azure"
[15] "azure2"
[17] "azure4"
[19] "bisque"
[21] "bisque2"

"aliceblue"
"antiquewhite1"
"antiquewhite3"
"aquamarine"
"aquamarine2"
"aquamarine4"
"azure1"
"azure3"
"beige"
"bisque1"
"bisque3"
```

Figure:

2.4 Use of the Semi-Colon Operator

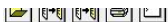
- ▶ The semi-colon operator at the end of each line of code is not necessary in general, but using it overcomes errors due to copying and pasting from document soft copies.
- ▶ It is also useful for compacting multiple short statements onto a single line.
- ▶ In other programming languages, such as Octave, using the semicolon in this way has a distinct purpose.

2.5 The apropos() Function

This function is very useful for determining what functions are available for a particular topic, although the process requires a great deal of trial and error. Suppose we are looking for a command to compute the correlation coefficient. We would use a very short string (e.g. cor) that would plausibly be part of useful function names.

2.6 History

- ▶ The command `history()` is used to obtain the last 25 commands used by R.
- ▶ 25 is the default number, you can specify another number.

**R**

R Console

```
>  
>  
> history()  
> |
```

R

R History

```
Dynamite plots  
=====  
class: small-code  
``{r}  
dyp <- ggplot(sumd, aes(x=conservation, y=awake)) + geom  
``  
``{r echo=FALSE}  
dyp <- dyp +  
labs(x="Conservation", y="Awake")+  
theme(axis.title=element_text(face="bold", size=24, lin  
axis.text=element_text(size=20, color="black"))  
``  
``{r fig=TRUE, fig.align="center", fig.width=12, fig.h  
dyp  
``
```


2.7 The `sessionInfo()` Function

To get a description of the version of R and its attached packages used in the current session, we can use the `sessionInfo()` function

```

> sessionInfo()
R version 3.1.2 (2014-10-31)
Platform: i386-w64-mingw32/i386 (32-bit)

locale:
[1] LC_COLLATE=English_Ireland.1252  LC_CTYPE=English_Ireland.
[3] LC_MONETARY=English_Ireland.1252 LC_NUMERIC=C
[5] LC_TIME=English_Ireland.1252

attached base packages:
[1] stats      graphics  grDevices  utils      datasets  methods
[7] base

loaded via a namespace (and not attached):
[1] colorspace_1.2-4 digest_0.6.8      ggplot2_1.0.0
[4] grid_3.1.2      gtable_0.1.2     MASS_7.3-35
[7] ...

```

Figure:

2.8 Time and date functions

The commands `Sys.time()` and `Sys.Date()` returns the systems idea of the current date with and without time. We can perform some simple algebraic calculations to compute time differences (i.e. to find out how long some code took to compile).

System Time

```
.  
> X1=Sys.time()  
> #wait a few seconds  
>  
> X2=Sys.time()  
>  
> X2-X1  
Time difference of 12.52164 secs  
> X2  
[1] "2015-03-06 19:11:41 GMT"  
> X1  
[1] "2015-03-06 19:11:28 GMT"  
> |
```

Figure:

2.9 Logical States

- ▶ Logical states TRUE and FALSE are simply specified as such, all in capital letters.
- ▶ The shortcuts T and F are also recognized

2.10 Missing Data

- ▶ In some cases the entire contents of a vector may not be known. For example, missing data from a particular data set.
- ▶ A place can be reserved for this by assigning it the special value NA. NA is a logical constant of length 1 which contains a missing value indicator.
- ▶ NA stands for Not Available.

2.11 Files in the Working Directory

It is possible to call an R script from the working directory, using the `source()` function.

```
source("myfunctions.r")  
source("mydata.r")
```

We can also send code put directly to a file in the working directory, using the `sink()` command. The first time the command is used, the name of the created file is specified. Subsequent commands print output directly to this file, until the command is used again to cease the operation.

```
>  
> sink("IrisSum.txt")  
> summary(iris)  
> sink()  
>  
> |
```

Figure:

Section 3 : Inspecting a Data Set

Section 3 Inspecting a Data Set

Summary of useful commands

- ▶ `dim()` and `length()`
- ▶ `nrow()` and `ncol()`
- ▶ `names()`
- ▶ `summary()`
- ▶ `tail()`
- ▶ `head()`
- ▶ `describe()` (from the **psych** package)

3.1 Dimensions of a data set

- ▶ We have remarked that some data sets are very large.
- ▶ This is perhaps a good place to consider summary information about data objects.
- ▶ For a simple vector, a useful command to determine the length (remark: sample size) is the function `length()`.

```
Y=4:18 ; length(Y) [1] 15
```

For more complex data sets (and data frames which we will see later) , we have several tools for assessing the size of data.

```
>  
>  
> dim(iris) # dimensions of data set  
[1] 150    5  
>  
> nrow(iris) # number of rows  
[1] 150  
>  
>  
> ncol(iris) # number of columns  
[1] 5  
> |
```

Figure:

Column (Variable) names and Row names

- ▶ We can also determine the row names and column names using the functions `rownames()` and `colnames()`.
- ▶ If there are no specific row or column names, the command will just return the indices.

```
>  
> colnames(iris)  
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length"  
[4] "Petal.Width"  "Species"  
> |
```

Figure:

3.2 The `summary()` command

- ▶ The command `summary()` is one of the most useful commands in R.
- ▶ It is a generic function used to produce result summaries of the results of various functions.
- ▶ The function invokes particular methods which depend on the class of the first argument.
- ▶ In other words, R picks out the most suitable type of summary for that data.

```
> summary(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300
Median :5.800	Median :3.000	Median :4.350	Median :1.300
Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.199
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500


```
Species
setosa      :50
versicolor:50
virginica   :50
```

`summary()` is particularly useful for manipulating data from more complex data objects.

3.3 Structure of a Data Object

The structure, class and storage mode of an object can be determined using the following commands. Try out a few.

- ▶ `str()`
- ▶ `class()`
- ▶ `mode()`


```
>  
>  
> class(Nile)  
[1] "ts"  
>  
> mode(Nile)  
[1] "numeric"  
>  
> |
```

Figure:

```
> a<-6
> mode(a)
[1] "numeric"
> class(a)
[1] "numeric"
> str(a)
  num 6
>
> mode(iris)
[1] "list"
> class(iris)
[1] "data.frame"
> |
```

Section 4 : Packages

Packages

- ▶ A Package in R is a file containing a collection of objects which have some common purpose.
- ▶ Packages enhance the capabilities and scope for research in a certain field.
- ▶ For example, the package MASS contains objects associated with the Venables and Ripley's *“Modern Applied Statistics with S.”*

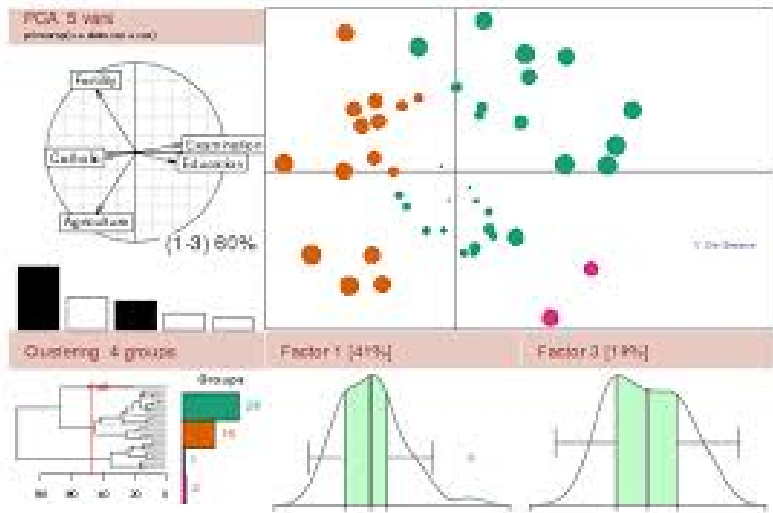


Figure: Comprehensive R Archive Network

R Packages

- ▶ “10 R packages I wish I knew about earlier” - Drew Conway (Yhat.com, February 2013)
- ▶ “The HadleyVerse” - Hadley Wickham
 - ▶ ggplot2, dplyr, reshape, lubridate, stringr
 - ▶ With Romain Francois, Dianne Cook and Garret Golemund.
- ▶ Dr Brendan Haplin (UL): lme4 ,TraMineR, Gelman’s arm, MASS, foreign.
- ▶ Shiny - Web Applications with R

R Packages

Some examples of packages are Actuar, written for actuarial science, and QRMLib, which complements the Quantitative Risk Management. The command `library()` lists all the available packages.

To load a particular package, for example MASS, we would write `library(MASS)`

Packages

- ▶ The CRAN package repository features 6107 available packages.
- ▶ Packages contain various functions and data sets for numerous purposes, e.g. **ggplot2** package, **AER** package, **survival** package, etc.
- ▶ Some packages are part of the basic installation. Others can be downloaded from CRAN.
- ▶ To access all of the functions and data sets in a particular package, it must be loaded into the workspace.
- ▶ For example, to load the **ggplot2** package:

```
install.packages(ggplot2) library(ggplot2)
```


4.2 Using and Installing packages

- ▶ Many packages come with R. To use them in an R session, you need to load the package, as previously demonstrated.
- ▶ Some packages are not automatically installed when you install R but they need to be downloaded and installed individually.
- ▶ We must first install them using the `install.packages()` function, which typically downloads the package from CRAN and installs it for use. (follow the instructions as indicated).

```
install.packages("ggplot2") install.packages("qcc")  
install.packages("sqldf") install.packages("RMongo")  
install.packages("randomforest")
```

4.2.1 Version of R

Many packages will require you to have the most recent version of R and also other packages. It is a good idea to update regularly.

Section 5 : Data Creation, Data Entry, Data Import and Export

5.1 The c() command

To create a simple data set, known as a vector, we use the c() command to create the vector.

```
|>  
>  
> Y=c(1,2,4,8,16 ) #create a data vector  
> Y  
[1] 1 2 4 8 16  
> |
```

Figure:

Vectors

```
> ### Vector of Numeric Values  
> Numvec = c(10,13,15,19,25);  
> ## Vector of Character Values  
> Charvec = c("LouLou","Oscar","Rasher");  
>  
> ## Vector of Logical Values  
> Charvec = c(TRUE,TRUE,FALSE,TRUE);  
> |
```

Figure:

Vectors

Vectors can be bound together either by row or by column.

```
>  
> X=1:3; Y=4:6  
> cbind(X,Y)  
      X Y  
[1,] 1 4  
[2,] 2 5  
[3,] 3 6  
> rbind(X,Y)  
      [,1] [,2] [,3]  
X        1    2    3  
Y        4    5    6  
> |
```

5.2 The `scan()` command

- ▶ The `scan()` function is a useful method of inputting data quickly.
- ▶ You can use to quickly copy and paste values into the R environment. It is best used in the manner as described in the following example.
- ▶ Create a variable `X` and use the `scan()` function to populate it with values.
- ▶ Type in a value, and then press return. Once you have entered all the values, press return again to return to normal operation.


```
>  
>  
> X=scan()  
1: 5  
2: 6  
3: 7  
4: 84  
5: 12  
6:  
Read 5 items  
> |
```

5.2.1 Characters with the scan() command

The scan() command can also be used for inputting character data.

```
R  
|  
>  
>  
> Y=scan(what=" ")  
1: Oscar  
2: LouLou  
3: Charlie  
4: Anita  
5: Maria  
6:  
Read 5 items  
> |
```

5.3 Using the data editor

5.4 Spreadsheet Interface

R provides a spreadsheet interface for editing the values of existing data sets. We use the command `data.entry()`, and name of the data object as the argument. (Also try out the `fix()` command)