Examples in R for the Novice

Last modified: 23/11/2006 (Frank Masci)

Contents of this Document:
- *Goals*
- *Getting Started with 'R Help'*
- *Summary of FAQs and Examples presented below*
- *Answers to FAQs and Example Code*
- *References*

Goals

Rather than present you with another R manual, the focus here is to give some simple examples to get you started. There are a plethora of manuals and books out there if you need the details (e.g. how to change the font in the legend of a plot). A foundation is important before building a program tailored for your own needs.

It's important to note that this is for beginners, or for those wishing to come up to speed with the language. There is really no limit with what you can do with R and its functionality is growing as you read. The documents and websites referenced below are by no means exhaustive. They serve as good starting points with other documents referenced therein.

We envisage the number of examples to grow as people in the section continue to use R for their research and discover new tidbits and tricks that can be shared. This goes without saying since we are all 'Good Corporate Citizens'.

If you wish to include an example below, please ensure:
1. you provide a brief one-liner in the "Summary of FAQs and Examples..." section and your name, e.g. " *Contributor:John Knucklehead* ".
2. it is simple as possible (but not too simple) and has a well defined objective;
3. is interspersed with detailed comments;
4. doesn't already exist here in some variation, unless the functionality is vastly superior, or its objective vastly different.
5. you use the `Courier New, size 8 font` to ensure formatting is preserved when copying and pasting the code into a text file for execution in R, or, directly into the R GUI;
6. you update the **"Last modified:"** field above with the date and your name.

Getting Started with 'R Help'

You can install the R programming interface from the ABS software portal. R is 'open source' so supervisor approval is not needed, i.e. there are no licenses and no limit to the number of simultaneous users. The default installation comes with a standard set of libraries and packages. You can import more of these to suit

your needs in future (see examples below). Having started the R 'Graphical User Interface' (GUI), you can play around for yourself by browsing the help files. Here are a few useful commands on how to access help documentation and browse the packages available on your system:

To see which packages are installed:
> library()

To see which packages are currently loaded by default on first starting R:
> search()

To see the directories from which R packages are being searched for on your system:
> .libPaths()

To obtain a listing of all functions ('subroutines') in a package (e.g. the 'MASS' package):
> library(help=MASS)

To load a particular package (e.g. the 'MASS' package) for use in your session from a standard installation directory (a standard installation directory is one listed using the `".libPaths()"` *command above.):*
> library(MASS)

To load a particular package (e.g. the 'sx11' package) for use in your session from a non-standard installation directory:
> library(sx11,lib.loc="s:\\splus\\library6")

To view the help documentation and usage examples for a specific function (e.g. the 'rlm' function in the 'MASS' package):
> help(rlm)

Note that you must first load the package containing the function you wish to access, even if just accessing its help file.

To search all packages and functions that contain a particular word of interest in their help file, e.g. search for "dir":
> help.search("dir")

To globally reset the current working directory from/to which input/output files should be read/written during execution:
> setwd("S:\\data\\mydirectory")

To import a function that does not exist in a standard package: e.g. "myfunction.r" located in some directory "user" into a program or R session:
> source("s:\\user\\myfunction.r");

*To customize your default directories for library and package searches, copy the contents of the TSA template file "**S:\R\rw2001\etc\Rprofile**" to your local R installation copy: "**C:\Program Files\R\rw2001\etc\Rprofile**". The ".libPaths" setting in this file already specifies TSA's default package directories. You can add more directories if you like. Furthermore, you can customize other R settings in this file, e.g. you may want to redefine your local working directory (see "setwd" command above), or, you can explicitly "source" R files containing functions you commonly use. These will be imported each time you invoke R.*

Summary of FAQs and Examples presented below

1. *What's the best way of executing R code? Contributor: Frank Masci*

2. *How do I read in data (in columns x, y, z...) from a text file, perform a linear regression between two quantities, create some plots, and write some diagnostics to an output text file? Contributor: Frank Masci.*

3. *How do I read in time series from text files (e.g. in SEASABS format), store them into time series objects, perform some operations (e.g. compute movements), and quickly plot them? Contributor: Frank Masci.*

4. *How do I reformat/restructure and manipulate data in the form of matrices, lists and vectors? Contributor: Nick von Sanden.*

5. *How do I.....*

Answers to FAQs and Example Code

1. *What's the best way of executing R code?*

To execute any of the example codes below, you can either:

1. copy and paste the code into the R GUI, ensuring the directory locations of all input files are correct.
2. better: place the code in a text file, e.g. "`mycode.r`" in a private directory: "`s:\\user\\`" and then execute:
   `source("s:\\user\\mycode.r");`
3. if you're comfortable using DOS command prompts: place the code in a text file e.g. "`mycode.r`" in a private directory, then within that same directory, execute on the DOS command prompt:
   `"S:\\R\\rw2001\\bin\\Rterm.exe" -q --no-restore --no-save < mycode.r`
   If you want diagnostic outputs to be redirected to a log file, e.g. "`outlog.txt`" then simply append "`> outlog.txt`" to this execution line.

2. *How do I read in data (in columns x, y, z...) from a text file, perform a linear regression between two quantities, create some plots, and write some diagnostics to an output text file?*

```
# Lines starting with "#" are comments and are not executed.

# The following line tells R not to echo each input line back to the
# screen after it's executed:
options(echo = FALSE);

# Import the package "MASS" since it contains the "rlm" function we
# call below:
library(MASS);

# Following line is optional: set the current working directory from/to
# which input/output files will be read/written. Alternatively, can explicitly
# prepend directory paths to file names specified below.
setwd("P:\\My Documents\\FrankMasci\\ExampleRcode");

#-------------------------------------------------------------
```

```
# Read data from text file: "Time_mvts_abs.txt". You can also prepend
# the directory path. This file contains five columns, which we call:
# times, mvt1, mvt2, absmvt1, absmvt2. The "list" argument is used
# to specify the datatypes: "" => string, 0 => floating point number:

xydata <- scan("Time_mvts_abs.txt", list("",0,0,0,0));

# Assign variables to columns specified in input file:

times <- xydata[[1]]
x <- xydata[[2]];
y <- xydata[[3]];
absx <- xydata[[4]];
absy <- xydata[[5]];

#-------------------------------------------------------------
# Perform robust linear regression using "rlm" function in the "MASS"
# package. By "robust", we mean as outlier resistant as possible.
# Here we regress "y" (column 3 in input file) against "x". Outputs
# from the regression are then saved in separate parameters.
# Note that we also define new variables from the regression output
# such as t-vales and p-values. See the "R-help" entry on "rlm" for
# more details.

lin <- rlm(y ~ x + 1, method="MM");
par <- coefficients(summary(lin));
intercept <- par[1,1];
slope <- par[2,1];
seint <- par[1,2];
seslope <- par[2,2];
tint <- abs((intercept - 0)/seint); ## i.e. test: H0: intercept=0
tslope <- abs((slope - 1)/seslope); ## i.e. test: H0: slope=1
pvalueint <- 2*pt(tint, (length(x)-2), lower.tail = FALSE);
pvalueslope <- 2*pt(tslope, (length(x)-2), lower.tail = FALSE);

#-------------------------------------------------------------
# Write the values stored above to the file: "results.txt".
# If you want to write this output directly to the screen, then
# simply specify file="".

write(paste("\n SLOPE (OUTLIER RESISTENT) =",slope), file="results.txt", append=F);
write(paste(" STANDERROR_SLOPE =",seslope), file="results.txt", append=T);
write(paste(" T-VALUE_SLOPE =",tslope), file="results.txt", append=T);
write(paste(" P-VALUE_SLOPE (H0: slope=1) =",pvalueslope), file="results.txt", append=T);
write(paste(" INTERCEPT (OUTLIER RESISTENT) =",intercept), file="results.txt", append=T);
write(paste(" STANDERROR_INT =",seint), file="results.txt", append=T);
write(paste(" T-VALUE_INT =",tint), file="results.txt", append=T);
write(paste(" P-VALUE_INT (H0: intercept=0) =",pvalueint,"\n"), file="results.txt",
append=T);

#-------------------------------------------------------------
# Create plot 1: simply column 3 versus column 2 of input file.
# See the "R-help" entry on "plot" for the gory details on each argument.
# This example also places a legend on the plot:

# Define the output plotting device, here the plot is saved to a png file.
# Note: png appears to be the best quality for our windows viewing devices:
png(file="mvt1_vs_mvt2.png", width=680, height=450, pointsize=12, bg="white", res=200);
# Define the plotting frame:
plot(x, y, main="Movements at equal times", xlab="% movement in series 1", ylab="% movement
in series 2", pch=" ", cex.lab=1.3, cex.axis=1.2, cex.main=1.4)
# Show the actual data:
points(x, y, col="red", pch=24, bg="red");
# Add lines to this plot defined by (intercept, slope) arguments:
abline(0, 1, col="black");
abline(intercept, slope, col="blue", lty=2);

# Place a legend on the plot. This example is pretty complete.
legend(15, -2,                          # coordinates of desired legend location.

       c("Real values","Linear fit","Line of equality"), # text labels in legend.
       col=c("red","blue","black"),   # line colours.
       text.col="black",              # text colour.
       lty=c(-1,2,1),                 # line styles: "-1" => ignore.
       lwd=c(-1,1,1),                 # line widths.
       pch=c(24, -1, -1),             # symbol types of points.
       pt.bg=c("red","blue","black"), # backgound colours of points.
       merge=TRUE,                    # place points in middle of lines.
       bty='n');                      # don't place a frame around legend.

# close the output plotting device:
```

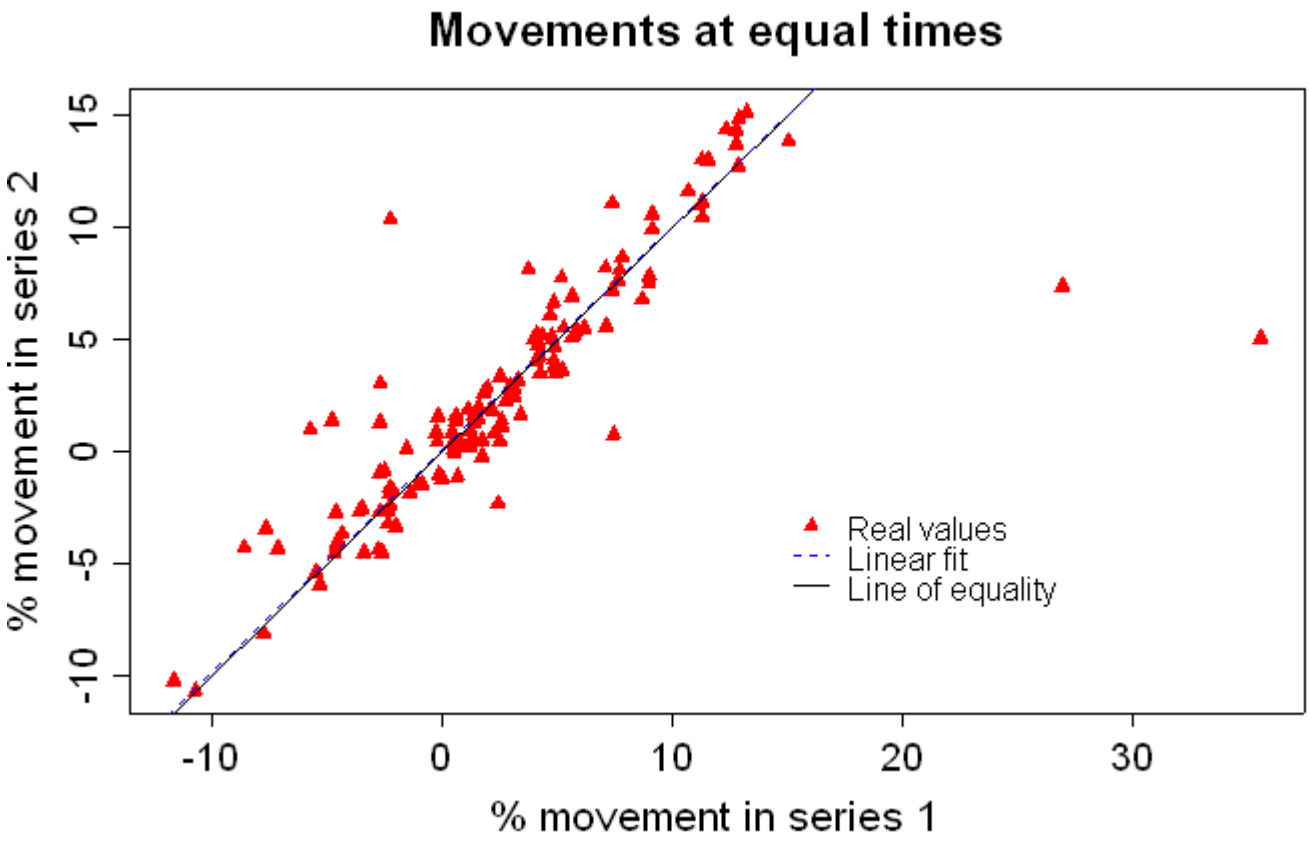
```
dev.off();

#-------------------------------------------------------------
# Create plot 2: columns 2 and 3 versus column 1 (time) from input file.
# This plot attempts to label the x-axis with actual time label strings
# provided in our input file (column 1). These time labels are strings,
# not numerical values and thus require special handling. If we simply
# wanted to plot numerical values, see the default call to "plot" in the
# previous example.

png(file="mvt_vs_time.png", width=680, height=450, pointsize=12, bg="white", res=200);
# In the following, we have set "axes=FALSE" since these will be explicitly defined below:
plot(x, main="Movements versus time", xlab="Quarter / year", ylab="% movement", pch=" ",
frame=TRUE, axes=FALSE, cex.lab=1.3, cex.main=1.4);
# Show the data as joined lines and show a point in red at the end of the series:
lines(x, col="blue",lty=1);
lines(y, col="green",lty=1);
points(length(x), y[length(y)], col="red", pch=16, cex=1.5);
# Add a horizontal line [with intercept,slope = 0,0]:
abline(0, 0, col="black");
# Show the axes with time lable strings. Note that the "9" in the following line
# specifies the sampling interval separating the tick marks on the x-axis:
axis(1, seq(1,length(x),round((length(x)/9),0)),
labels=times[seq(1,length(x),round((length(x)/9),0))], las=1, cex.axis=1.2);
axis(2, cex.axis=1.3);

# close the output plotting device:
dev.off();

#-------------------------END-----------------------------------
```

Example Outputs:

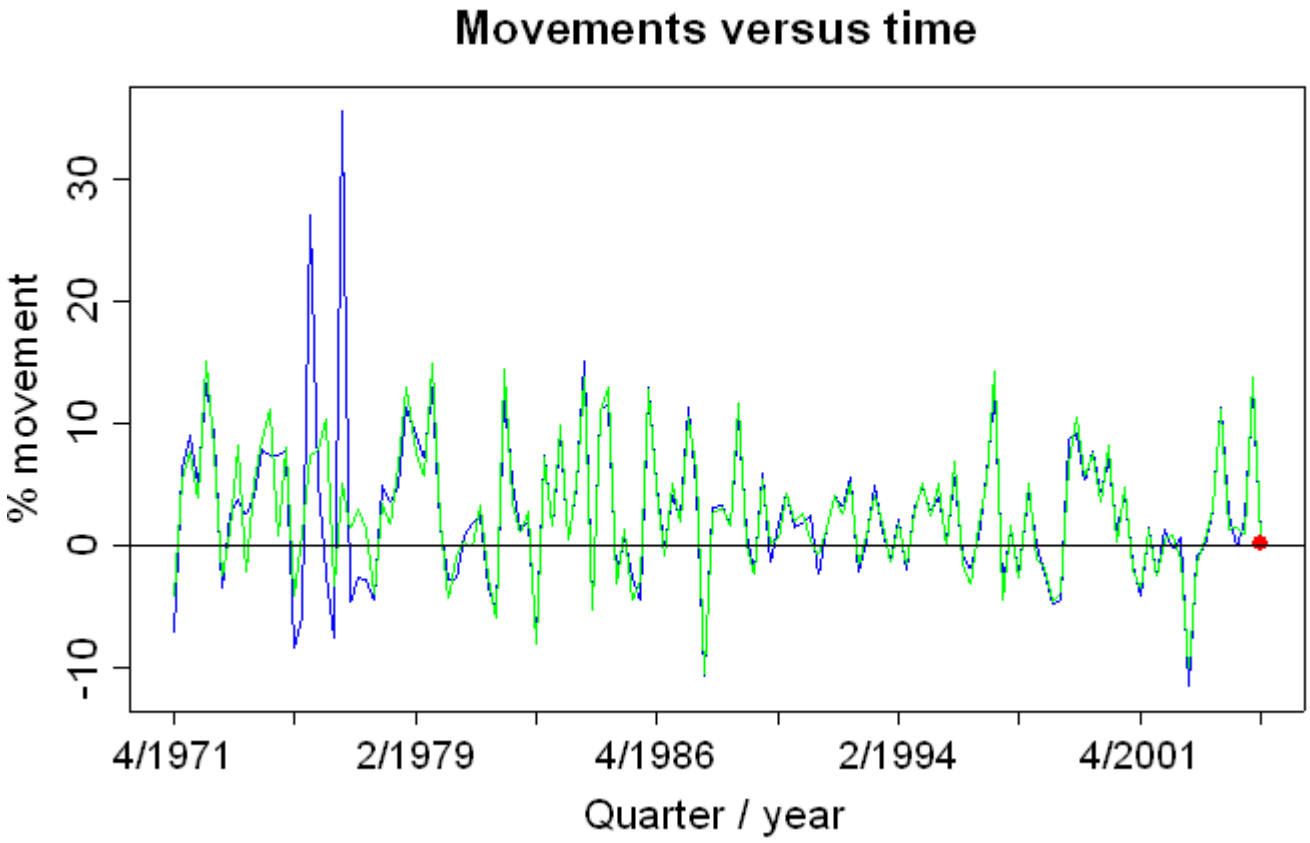
The file "results.txt" looks like:

```
SLOPE (OUTLIER RESISTENT) = 0.994422221122267
STANDERROR_SLOPE = 0.0176829270360894
T-VALUE_SLOPE = 0.315433008706611
P-VALUE_SLOPE (H0: slope=1) = 0.752923623184282
INTERCEPT (OUTLIER RESISTENT) = 0.094800704213533
STANDERROR_INT = 0.121839528118300
T-VALUE_INT = 0.778078392764999
P-VALUE_INT (H0: intercept=0) = 0.437894477162239
```

Plot 1 looks like:



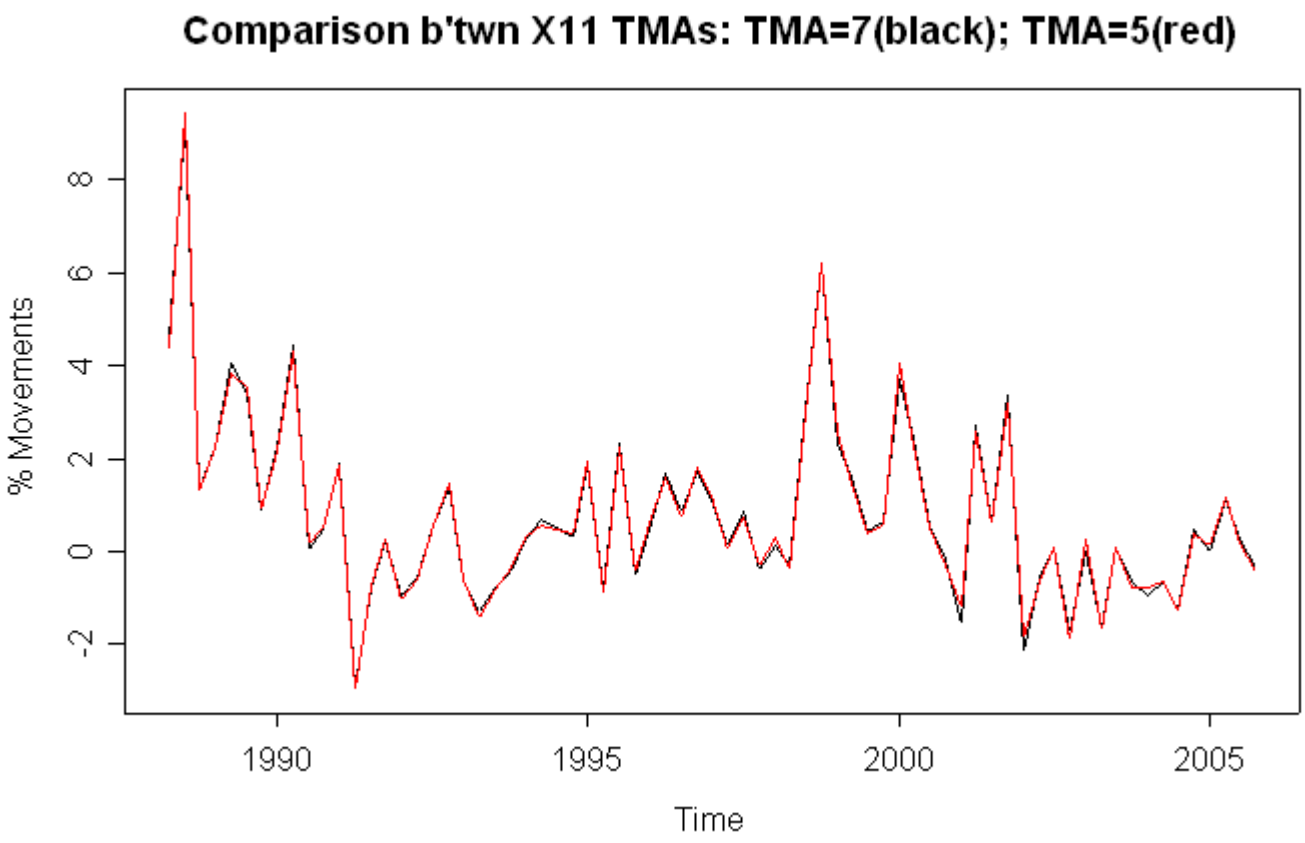
Plot 2 looks like:



3. *How do I read in time series from text files (e.g. in SEASABS format), store them into time series objects, perform some operations (e.g. compute movements), and quickly plot them?*

```
# Lines starting with "#" are comments and are not executed.

# The following line tells R not to echo each input line back to the
# screen after it's executed:
options(echo = FALSE)

# Following line is optional: set the current working directory from/to
# which input/output files will be read/written. Alternatively, can explicitly
# prepend directory paths to file names specified below.
setwd("P:\\My Documents\\FrankMasci\\ExampleRcode");

#---------------------------------------------------------------------
# Read in two series from text files. You can also prepend
# the directory paths. Each file contains three columns:
# "year", "period", "value" as generated by the SEASABS "Save.." option.

x1 <- scan("SA7_8GR.txt", list(0,0,0));
x2 <- scan("SA5_8GR.txt", list(0,0,0));

# Assign variables to column 3 data from each input file:
data1 <- x1[[3]];
data2 <- x2[[3]];

#---------------------------------------------------------------------
# Store series data into time series objects for easier plotting and
# manipulation.

startyear <- 1988; # the startyear in each series. Here it's the same for both.
startperiod <- 1;  # the starting period (month or qtr) in each series.
freq <- 4;         # 4 => quarterly, 12 => monthly
ts1 <- ts( data1, start=c(startyear, startperiod), frequency=freq );
ts2 <- ts( data2, start=c(startyear, startperiod), frequency=freq );

#---------------------------------------------------------------------
# Compute % movements in each time series:

mvt_ts1 <- 100*diff(ts1,1)/ts1[1:length(ts1)-1];
mvt_ts2 <- 100*diff(ts2,1)/ts2[1:length(ts2)-1];

#---------------------------------------------------------------------
# Plot both time series on the same panel. First set up output plotting device.
# Here the plot is saved to a png file. Note: png appears to be the best quality
# for our windows viewing devices:

png(file="plot_movements.png", width=680, height=450, pointsize=12, bg="white", res=200);

# See the "R-help" entry on "ts.plot" for the gory details on each argument and more:
ts.plot(mvt_ts1, mvt_ts2, gpars=list(ylab=" % Movements", col=c(1:2)),
        main="Comparison b'twn X11 TMAs: TMA=7(black); TMA=5(red)");

# close the output plotting device:
dev.off()

#---------------------------END-----------------------------------
```

Example Output:



[4.](#) *How do I reformat/restructure and manipulate data in the form of matrices, lists and vectors?*

```
#Assume we have data loaded
> tmp<-100:3
#Data may be in form of vector, matrix, or list and may contain numeric or character
information. This can be checked with #the mode() and is.numeric/is.character functions.

> mode(tmp)
[1] "numeric"
> is.vector(tmp)
[1] TRUE

#now change to matrix and check dimensions

> tmp3<-matrix(tmp,nrow=7)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]  100   93   86   79   72   65   58   51   44    37    30    23    16     9
[2,]   99   92   85   78   71   64   57   50   43    36    29    22    15     8
[3,]   98   91   84   77   70   63   56   49   42    35    28    21    14     7
[4,]   97   90   83   76   69   62   55   48   41    34    27    20    13     6
[5,]   96   89   82   75   68   61   54   47   40    33    26    19    12     5
[6,]   95   88   81   74   67   60   53   46   39    32    25    18    11     4
[7,]   94   87   80   73   66   59   52   45   38    31    24    17    10     3

#this is equvialent to

>tmp2<-100:94
>for (i in 2:14) {
>tmp2<-cbind(tmp2,(100-(i-1)*7):(100-i*7+1))}
>tmp2

#which column binds vectors to produce the same final matrix with the following dimensions

> dim(tmp2)
[1]  7 14
> is.matrix(tmp2)
[1] TRUE
```

```
>
dimnames(tmp2)[[2]]<-c("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec
","Tot","Mean")
>
dimnames(tmp2)[[1]]<-c("Person1","Person2","Person3","Person4","Person5","Person6","Person7",
)
>
> tmp2
        Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Tot Mean
Person1 100  93  86  79  72  65  58  51  44  37  30  23  16    9
Person2  99  92  85  78  71  64  57  50  43  36  29  22  15    8
Person3  98  91  84  77  70  63  56  49  42  35  28  21  14    7
Person4  97  90  83  76  69  62  55  48  41  34  27  20  13    6
Person5  96  89  82  75  68  61  54  47  40  33  26  19  12    5
Person6  95  88  81  74  67  60  53  46  39  32  25  18  11    4
Person7  94  87  80  73  66  59  52  45  38  31  24  17  10    3

#This then allows selection of rows and columns of the matrix either by name or by
column/row. To select by column or #row we use the square brackets [i,j] to select elements
in columns j and row i. Note that multiple rows/columns can be #selected simultaneously and
leaving an index out will result in select of all elements in that index. Ie

> tmp2[,3]
Person1 Person2 Person3 Person4 Person5 Person6 Person7
     86      85      84      83      82      81      80
> tmp2[3,]
 Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec  Tot Mean
  98   91   84   77   70   63   56   49   42   35   28   21   14    7
> tmp2[2,6]
[1] 64
> tmp2[3,]
 Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec  Tot Mean
  98   91   84   77   70   63   56   49   42   35   28   21   14    7
> tmp2[,11]
Person1 Person2 Person3 Person4 Person5 Person6 Person7
     30      29      28      27      26      25      24
> tmp2[c(1,3,7),]
        Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Tot Mean
Person1 100  93  86  79  72  65  58  51  44  37  30  23  16    9
Person3  98  91  84  77  70  63  56  49  42  35  28  21  14    7
Person7  94  87  80  73  66  59  52  45  38  31  24  17  10    3

#An alternative is to use logical operators. For example to choose all rows for which the
mean column is less than or equal #to 5 we could use

> tmp2[tmp2[,14]<=5,]
        Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Tot Mean
Person5  96  89  82  75  68  61  54  47  40  33  26  19  12    5
Person6  95  88  81  74  67  60  53  46  39  32  25  18  11    4
Person7  94  87  80  73  66  59  52  45  38  31  24  17  10    3

#Note that any characters or non-numeric elements in the matrix will result in the matrix
being non-numeric so that #operations may not be able to proceed.

tmp4<-tmp2
tmp4[3,3]<-"No information available here"
mode(tmp4)

[1] "character"

#We can then combine the above options to replace this element (and any others) and then use
the is.numeric() function to #force the matrix back to numeric. Note this will introduce NA
(not available) into elements which do not convert to numeric

tmp5<-as.numeric(tmp4)
tmp5[!is.na(tmp5)]

#Note that by these commands we have lost one observation - relating to the NA element and
also lost all information on #the dimensions of the matrix - ie converted back to a vector.

#The third most used data object type in R is a list. This is a combination of data objects
that can have any mode or type. #For example a list can contain matrices, vectors and scalars
with differing attributes (one element of the list may be a time #series) and modes (elements
do not all have to be numeric). They are therefore more flexible than data.frames.

#You can create a list with the list command

tmp6<-list(tmp2[,1],tmp2[,2],as.character(tmp2[,3]),tmp2[c(4,5,6),])

#the names() function can be used to check the elements of the list or change the names of
```

```
the elements in the list

names(tmp6)<-c("Element 1","Vector element","Character element","Matrix element")

#Elements in the list can then either be selected by name or by position in the list. Note
that lists use the double square #bracket to specifiy elements [[]]. Note also that the
element of a list may be either another list or a vector so square #brackets can be used
after the double square brackets to select an element of the matrix stored as an element in
the list. #For example

> tmp6$"Character element"
[1] "86" "85" "84" "83" "82" "81" "80"
> tmp6[[3]]
[1] "86" "85" "84" "83" "82" "81" "80"
> tmp6[[3]]==tmp6$"Character element"
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE

> tmp6[[4]][,3]
Person4 Person5 Person6
     83      82      81
```

5. *How do I...*

References

- A very nice introduction to R, for novice and expert alike:

  R_introduction.pdf

- A short course in R as taught in the Mathematics Department at ANU:

  R_overview_ANU.pdf

- Coupled with the above, the following 'reference card' is indispensible:

  R_reference_card.pdf

- For a list of time series specific functions:

  R_timeseries_ref.pdf

- A very basic tutorial and refresher course on R and S-Plus commands:

  SPlusTutorial.pdf

- Also, don't forget the "Comprehensive R-Archive Network" (CRAN) website with links and FAQs contained therein:
  http://cran.r-project.org/

- Also, here's a website containing many tidbits and tricks of the trade (please keep this a secret, it's a good one!):
  http://pj.freefaculty.org/R/Rtips.html#12.1