

Time Series and Forecasting 161.342

Introduction to R and Basic Time Series Functions

What is R?

R is a computing environment which incorporates the S programming language originally developed by Chambers *et al* at the Bell Laboratories in the USA. The S language was specifically designed to create a flexible environment for the analysis of data, and consequently has become one of the most popular tools with statisticians and data analysts. The environment is similar to the commercial package S-Plus, although R is a more efficient implementation in terms of computing speed.

Downloading R

You can download R for personal use free of charge via the web site:

<http://cran.r-project.org/>

If you decide to download R for your personal use, then first go to the pre-compiled (binary) links at the above site and follow the appropriate instructions for the operating system you are using (e.g. Windows, Mac or Linux). You need to choose *base* followed by *setup* and follow the instructions given.

R comes with no warranty or licensed support (because it is free). However, you can subscribe to an active mail base for asking questions or seeking advice on problems related to R. To do this, send an email to Majordomo@stat.math.ethz.ch with the command: `subscribe r-help` in the message.

Getting Help in R

Start R (in Windows select the executable program from the `start' menu). When you quit R, e.g. by typing `q()`, you will be asked whether you want to save the *workspace image*: in general click *no*. To get help on any command type `?name`, where *name* is the name of the command you need help with. Of course this assumes you know the name of the command you need help with! The only command you know so far is `q()`: so try typing: `q()`. After you've looked at the help on `q` close the help window.

You can scroll through all commands that you've used by using the *up* and *down* arrows on your keyboard. This can be useful if you want to execute the same command again or if you want to edit a previous command you've typed. Try pressing the *up* and *down* arrows now to see what happens.

You can also get help via a hyperlink *html* system. Click on *help* on the menu, followed by *help html* or type `help.start()` at the command prompt (try it now). You will get a range of html links to choose from. Some useful ones for you to try are:

- *An Introduction to R* - a useful introductory text
- *HTML help* - as above
- *Packages* - a list of libraries that can be loaded into R with the `library()` command.
- *Search* - useful if you know what you want to do, but don't know the right command or functions (try typing *boxplot* in the search engine to see what it returns).

Entering Data

There are three main ways of getting data into R:

1. Typing the data in using the keyboard: Enter the data within the brackets of `c()`
2. Reading in a single variable, in no particular format: Use `scan()`
3. Reading in a table of one or more variables: Use `read.table()`

As an example of 1 above, create a vector of the amount spent in a supermarket by 4 customers. Type the following and hit *return*.

```
> spent <- c(3.11, 8.88, 9.26, 10.81)
```

Notes:

- `>` is the R command prompt (you don't type this!)
- An assignment in R is written as `<-` or `=`
- The function `c()` stands for *concatenate* and places the data into a *vector* called *spent*
- Comments in R are preceded by the hash symbol `#` - everything after `#` is ignored by R. However, you should not ignore comments, because they sometimes tell you something useful or include instructions!

Type `spent` to view the elements of the vector:

```
> spent
[1] 3.11 8.88 9.26 10.81
```

Note you can access the elements of a vector by referring to its index in *square* brackets, e.g.

```
> spent [3]
[1] 9.26
```

gives the amount spent by the third customer. Note: this is useful because it means we do not have to scroll through copious output to get the information we want.

As an example of 2 above, we will scan unsorted data into the vector *spent* and then sort the data into ascending order. The data are in a file called `spent.dat` on the H-drive. The data can be read using `scan` with `file.choose()` to open a window for selecting the data file, or you can just type in the name of the data file in

quotation marks: `"/path/spent.dat"` (you will need to know and enter the path where the data has been saved). Read the data into R now using `scan`:

```
# Remember comments follow # so first two line below do nothing
#`file.choose' creates a window for selecting files
# data will be stored in the vector `spent'

> spent <- scan(file.choose())
Read 50 items
```

List the 50 observations:

```
> spent
 [1] 17.39 39.16 42.97 41.02 28.38 19.50 10.81 19.27  9.26 25.13
 [2] 20.16 18.36
 [13] 24.47 93.34 48.65 19.54 59.07 17.00 32.03 52.75 22.22 44.67
 [14] 18.43 27.65
 [25] 28.06 26.26 13.78 54.80 61.22 46.69  8.88 34.98 23.04 15.62
 [26] 15.23 50.39
 [37] 86.37 12.69 36.37 45.40 85.76 26.24 24.58 44.08 70.32  3.11
 [38] 20.59 28.08
 [49] 82.70 38.64
```

These observations are unordered. Put them into ascending order:

```
> spent <- sort(spent)

> spent
 [1]  3.11  8.88  9.26 10.81 12.69 13.78 15.23 15.62 17.00 17.39
 [2] 18.36 18.43
 [13] 19.27 19.50 19.54 20.16 20.59 22.22 23.04 24.47 24.58 25.13
 [14] 26.24 26.26
 [25] 27.65 28.06 28.08 28.38 32.03 34.98 36.37 38.64 39.16 41.02
 [26] 42.97 44.08
 [37] 44.67 45.40 46.69 48.65 50.39 52.75 54.80 59.07 61.22 70.32
 [38] 82.70 85.76
 [49] 86.37 93.34
```

Note: the same output could be obtained by typing `sort(spent)`, but this would leave the vector `spent` unchanged (i.e. unsorted).

Basic Statistical Functions

Once the data is in a vector, there are plenty of functions for finding useful summary statistics, e.g. the *mean*, *median*, *standard deviation*, and *variance*. Work through the examples below, checking your output against mine. Remember that comments are given after the hash `#` symbol - don't waste your time typing in my comments!

```
> mean(spent)
 [1] 34.7022
> sd(spent)
 [1] 21.6974
> var(spent)
 [1] 470.7771
> median(spent)
 [1] 27.855
```

```
> quantile(spent, 0.5) # this, of course, is the same as the median
 50%
27.855
```

For the next command below use the UP arrow and edit the 0.5 to 0.25

```
> quantile(spent, 0.25) # the lower quartile = 0.25 quantile = 25th
percentile
 25%
19.3275
> quantile(spent, 0.75)
 75%
45.2175
> IQR(spent) # note the capitals: R is CASE sensitive!
[1] 25.89
> quantile(spent, 0.75) - quantile(spent, 0.25)
 75%
25.89
> max(spent)
[1] 93.34
> min(spent)
[1] 3.11
```

The following gives some useful summary statistics:

```
> summary(spent)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 3.11  19.33   27.85   34.70  45.22   93.34

> hist(spent) # a graphics window should open with the picture
```

The number of *bins* in the histogram can be changed using the break parameter (abbreviated to br):

```
> hist(spent, br=10)
> boxplot(spent)
> qqnorm(spent) # Normal quantile plot: approx. straight if data
'normal'
```

Time Series Plots

R contains many functions for the analysis of financial time series. A large number of these functions are in a library called `tseries`. Therefore, in general, once you're in R, type: `library(tseries)` to load this library of functions - do it now.

Let us look at some time series data. There are many data sets within R itself which can be loaded with `data(nameofdata)`. For example, the number of passengers per month for an airline company. Follow the instructions below.

```
> data(AirPassengers)
```

To list the current R workspace use `ls()`:

```
> ls()
[1] "AirPassengers"
```

To find out what "AirPassengers" is use class

```
> class(AirPassengers)
[1] "ts"
```

So AirPassengers is of class `ts` (which stands for *time series*). Objects of class `ts` have lots of useful functions that can be applied, e.g.

```
> frequency(AirPassengers)
[1] 12
> start(AirPassengers)
[1] 1949 1
> end(AirPassengers)
[1] 1960 12
```

To just list all the data in `AirPassengers` use:

```
> AirPassengers
      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
1950 115 126 141 135 125 149 170 170 158 133 114 140
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234 264 302 293 259 229 203 229
1955 242 233 267 269 270 315 364 347 312 274 237 278
1956 284 277 317 313 318 374 413 405 355 306 271 306
1957 315 301 356 348 355 422 465 467 404 347 305 336
1958 340 318 362 348 363 435 491 505 404 359 310 337
1959 360 342 406 396 420 472 548 559 463 407 362 405
1960 417 391 419 461 472 535 622 606 508 461 390 432
```

The seasons the data belong to (i.e. *indicator variables*) can be created using `cycle`:

```
> cycle(AirPassengers)
      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949   1   2   3   4   5   6   7   8   9  10  11  12
1950   1   2   3   4   5   6   7   8   9  10  11  12
1951   1   2   3   4   5   6   7   8   9  10  11  12
1952   1   2   3   4   5   6   7   8   9  10  11  12
1953   1   2   3   4   5   6   7   8   9  10  11  12
1954   1   2   3   4   5   6   7   8   9  10  11  12
1955   1   2   3   4   5   6   7   8   9  10  11  12
1956   1   2   3   4   5   6   7   8   9  10  11  12
1957   1   2   3   4   5   6   7   8   9  10  11  12
1958   1   2   3   4   5   6   7   8   9  10  11  12
1959   1   2   3   4   5   6   7   8   9  10  11  12
1960   1   2   3   4   5   6   7   8   9  10  11  12
```

To get the annual totals use `aggregate`:

```
> aggregate(AirPassengers)
Time Series:
Start = 1949
End = 1960
Frequency = 1
[1] 1520 1676 2042 2364 2700 2867 3408 3939 4421 4572 5140 5714
```

Objects of class `ts` can also be plotted in various ways:

```
> plot(AirPassengers)
```

Use the UP ARROW to recall last command and then edit to:

```
> plot(log(AirPassengers))
```

Why might taking logs of this data help in subsequent analysis?

```
> acf(log(AirPassengers)) # notice peaks at 1 year cycle: Why?
> spectrum(log(AirPassengers)) # notice the peaks at multiples of 1
```

The data can be broken down into components (trend, seasonal effect, and remainder) using `stl`:

```
> plot(stl(AirPassengers, 'periodic'))
```

Note that functions are *embedded* within brackets so that `stl` acts on the object `AirPassengers` and then `plot` acts on the result of this. 'periodic' is a parameter value within the function `stl`, which tells R the data are *periodic* or *seasonal*.

Alternatively, to remove the seasonal effects, plot the annual totals:

```
> plot(aggregate(AirPassengers))
```

All the plots can be copied and pasted into a document to create a technical report.

Further exercise: Read in the chocolate-beer-electricity data (`cbe.dat`) from the H-drive using `cbe <- read.table(file.choose(), head=T)`. Look at the first 4 rows of the data: the data are in a *data frame*, which is like a *matrix* with elements $x[i, j]$ for the i -th row and j -th column. To view the first four rows, type `cbe[1:4,]`. To view the electricity data in column 3, type `cbe[, 3]`. In general, when *no* values are specified, *all* data are provided - so no row value gave produced all the electricity data). Plot the electricity data and the acf of the electricity data.

Now choose either the beer or chocolate data (depending on your taste!). The data was imported as a data frame object, which is standard in R. However, we would like the data to be a time series object. Therefore we need to coerce the data to class `ts`. For this you need to know when the data started and the cycle frequency. From the lectures, the data was monthly data (i.e. frequency 12) for the period January 1958 to December 1990. Thus, the command is given by:

```
> beer.ts <- ts(cbe[,2], start=1958, freq=12)
```

(I chose beer, but you may prefer to choose chocolate, in which case use an appropriate name and specify column 1 in the above). Now plot the data, the acf and spectrum of the data, and the aggregated data. For the aggregated data you should obtain a plot which smoothes out the seasonal variation.