Data Importing/Exporting

August 29,2013

Outline

- **Importing Data from different sources**
- **R add-on packages**
- **Accessing built-in data sets**
- **Loading data from other R packages**

Saving Data Frame

The data is first collected, and may be pre-processed using software, be it a spreadsheet or statistical software. Each piece of software has its own storage format; the simplest option is to convert data in a format common to all software(.csv or .txt)

- The function **read.table()** is the easiest way to import data into R. The preferred raw data format is either a tab delimited or a comma-separate file (CSV).
- The simplest and recommended way to import Excel files is to do a Save As in Excel and save the file as a tab delimited(.txt) or .CSV file and then import this file in to R.

Common Error When Importing Data

Sometimes, if the reading doesn't function correctly, the error may stem from the text file itself. Below is a list of some of the most common problems

- Column separator incorrectly specified
- Decimal point incorrectly specified
- Tabulation replacing a space
- Row and column name includes an apostrophe
- Problem of open speech marks

Robust Reading of Data

In some cases, reading data fails so we may use more robust procedure which can be used to identify the problem

- Use scan function
- Change the decimal separator from "," to "."
- Convert matrix to data-frame
- Convert the factors into numerics

Read a file directly from the web

We can import data files into R for analysis directly from a website. Example: Suppose we would like to import the census 2010 data from STATS. STATS is a dataset directory which contains example datasets.

```
> site<-"http://people.sc.fsu.edu/~jburkardt/datasets/census/census_2010.txt"
> data <- read.table(site, header=F)
```

```
> site<-"http://people.sc.fsu.edu/~jburkardt/datasets/stats/height_female_baby.csv"
> data <- read.table(site, header=T, sep=",")
```

```
>site<-"http://people.sc.fsu.edu/~jburkardt/datasets/stats/movie_budgets.csv"
>data <- read.table(site, header=T, sep=",")
>head(data, n=10)# print first 10 rows of the data
>tail(data, n=10) # print last 10 rows of the data
```

Interactive data frame entry in R

R also has a built-in spreadsheet that you can use to enter data frames interactively, bypassing the need for Excel altogether. To access it, you must use the data.frame() and edit() commands in R.
Example: Suppose we want to create a data frame with three variables: name , sex, and gpa. The name and sex variable is specified to be a character variable, while the gpa a numeric:

```
>mydata<-data.frame(name=character(0),sex=character(0), gpa=numeric(0))
> mydata <- edit(mydata)
```

R add-on packages

The base installation of R includes a number of packages in its library. These are collections of both functions and data, and will accommodate most of what we will need for this course. However, there are many additional add-on packages that reside at CRAN. These comprise add-ons that perform more very specialized tasks, contain user-contributed datasets, or are even designed to accompany published textbooks.
The stats, graphics, datasets methods, base ,(among several others) are automatically loaded at the beginning of a session. You can check **sessionInfo()** to see what packages are currently attached.

R Library

If you type **library()**, you will get a pop-up list of all other R packages currently installed on your computer.
If you want to ”attach” an add-on package, find the name in the list and type
>**library(packagename)**
Under Windows or other systems where the R console has menus across the top, choose Install Package(s)... from the Packages menu and follow the instructions. You must have a live Internet connection to do so.

Accessing built-in datasets

Several datasets are supplied with R (in package datasets), and others are available in packages (including the recommended packages supplied with R). To see the list of datasets currently available use

>**data()**

>data(AirPassangers) # Monthly Airline Passenger Numbers 1949-1960

> AirPassengers

Accessing data from a particular package

To access data from a particular package, use the package argument, for example

```
> data(package="rpart")
> data(Puromycin, package="datasets")
>Puromycin
```

If a package has been attached by library, its datasets are automatically included in the search.

Dates in R

The measurement of time is highly unique. Successive years start on different days of the week. There are months with different numbers of days. Leap years have an extra day in February. Occasional years have an additional 'leap second' added to them because friction from the tides is slowing down the rotation of the earth from when the standard time was set on the basis of the tropical year in 1900.

Sys.time() prints the date in the longest time scale

We can extract the date from **Sys.time()** using **substr** like this:

```
> substr(as.character(Sys.time()),1,10)
> substr(as.character(Sys.time()),12,19)
```

Note that **unclass** prints the number of seconds since 1 January 1970.

> **unclass(Sys.time())**

Date

There are two basic classes of date/times. Class "POSIXct" represents the (signed) number of seconds since the beginning of 1970 (in the UTC timezone) as a numeric vector. Class "POSIXlt" is a named list of vectors closer to human-readable forms, representing seconds, minutes, hours, days, months and years.
Note that **date()** prints the date and current time
> **date()**
Note that it prints hour in 24 hour-clock. We can convert **Sys.time** to an object that inherits from class POSIXlt
> date< −**as.POSIXlt(Sys.time())**

Date

We can use the element name operator $ to extract parts of the date and time from this object using the following names: sec, min, hour, mday, mon, year, wday, yday and isdst . Not that mday (=day number within the month), wday (day of the week starting at 0= Sunday), yday (day of the year after 1 January =0) and isdst which means 'is daylight savings time in operation?' with logical 1 for TRUE or 0 for FALSE).

```
> date=as.POSIXlt(Sys.time())
> date$wday
[1] 4
> date$yday
[1] 240
> date$isdst
[1] 1
```

Try **unlist(unclass(date))**

Calculations with dates and times

We can do the following calculations with dates and times:

- time + number
- time - number
- time1 - time2
- time1 'logical operation' time2

where the logical operations are one of $==, !=, <, <=,' >'$ *or* $>=$. Calculate the number of days between two dates, 22 September 2003 and 20 September 2005:

```
> date1<-as.POSIXlt("2003-09-22")
> date2<-as.POSIXlt("2005-09-20")
> date2-date1
Time difference of 729 days
```

The **difftime** function can be used as below

```
> difftime("2005-09-20","2003-09-22")
Time difference of 729 days
```

Calculation with dates

For differences in hours include the times (colon-separated) and write

```
>difftime("2005-10-21 6:14:21","2005-10-21 5:12:32")
Time difference of 1.030278 hours
```

Alternatively, we can subtract one date-time object from another directly:

```
>ISOdate(2005,10,21)-ISOdate(2003,8,15)
Time difference of 798 days
```

Converting Dates in R readable format

We can 'strip a date' out of a character string using the strptime function. There are functions to convert between character representations and objects of classes POSIXlt and POSIXct representing calendar dates and times.
Suppose we have a list of dates as in the file below:

```
>dates <- c("27/02/2004", "27/02/2005", "14/01/2003",
"28/06/2005", "01/01/1999")
>newdate<- strptime(dates,format="%d/%m/%Y")#Uppercase Y
> newdate
[1] "2004-02-27" "2005-02-27" "2003-01-14" "2005-06-28"
 "1999-01-01"
```

We can use **as.Date** to convert the data in date format.

```
date<-as.Date(bdate, format="%m/%d/%Y")
```