Data Importing/Exporting

Outline

- **Reading the list of files in a folder**
- **Importing Data from different sources**
- **R add-on packages**
- **Accessing built-in data sets**
- **Loading data from other R packages**

Reading a list of files and folders

- list.files function produce a character vector of the names of files named folder
- list.files(path) provides the directory information and all folders and files.
- We can select the specific file types.
- list.dirs produces the directory information about the folder and files in the given folder

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

Things to note before importing the data in R

- If you work with spreadsheets, the first row is usually reserved for the header, while the first column is used to identify the sampling unit;
- Avoid names, values or fields with blank spaces, otherwise each word will be interpreted as a separate variable, resulting in errors that are related to the number of elements per line in your data set;
- If you want to concatenate words, inserting a . in between two words instead of a space;
- Short names are prefered over longer names;
- Try to avoid using names that contain symbols such as `?, $,%, ^, &, *, (, ),-,#, <,>, /, |`
- Delete any comments that you have made in your Excel file to avoid extra columns or NAs to be added to your file; and
- Make sure that any missing values in your data set are indicated with `NA`.

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)
```

```
read.table("http://data.princeton.edu/wws509/datasets/effort.dat")
```

```
> 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
```

Note: We can read .csv files directly as

```
>read.csv("http://people.sc.fsu.edu/~jburkardt/datasets/stats/movie_budgets.csv")
```

Import Commands

Type of text file	R Command
Comma delimited (.CSV)	`read.table(<filename>, header=T, sep=',')`
TAB delimited (.TXT)	`read.table(<filename>, header=T, sep='\t')`
Blank space (.TXT)	`read.table(<filename>, header=T, sep='')`

Importing using `fread()`:

- `fread()` is a powerful data import function that is similar to read.table() but faster. It is part of the data.table package, which you will need to be installed. You should only have to give fread() the name of the file you want to import, and fread() will try to work out the appropriate way to import the data.
- In case you have a file with a separator character that is different from a tab, a comma or a semicolon, you can always use the *read.delim*() and *read.delim2*() functions. These are variants of the read.table() function, just like the read.csv() function

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.

```
> data(package="UsingR")
> data(blood, package="UsingR")
> blood
```

installr package- Useful to update the R and its packages

The installr package offers a set of R functions for the installation and updating of software (currently, only on Windows OS), with a special focus on R itself. To update R, you can simply run the following code:

```
if(!require(installr)) {
install.packages("installr"); require(installr)}
updateR()
```

Alternatively,

```
install.packages("installr)
library(installr)
```

Now you can go to the toolbar menu click installr and then update R or R packages.

Importing Data from other formats

- Importing Excel Files With The XLConnect Package

```
> library(XLConnect)
> wb <- loadWorkbook("<file name and extension>")
> data<-readWorksheet(wb,sheet=1, header=" ")
```

Note that you need to add the sheet argument to specify which sheet you want to load into R.

- Importing Excel files with xlsx Package

```
> library(xlsx)
> data<-readWorksheet(wb,sheet=1)
```

  - SPSS File:

```
library(foreign)
mydata <-read.spss(c:/datafile.spss)
```

  - STATA files have .dta extension

```
library(foreign)
mydata <-read.dta("<Path to file>")
```

  - Systat Files into R

```
library(foreign)
mydata <-read.systat("<Path to file>")
```

Reading SAS and Minitab files into R

Install the sas7bdat package. Load it, and then invoke the read.sas7bdat()

```
library(sas7bdat)
mySASData <- read.sas7bdat("example.sas7bdat")
```

You can also use the foreign library to load in SAS data in R. First step export the data from SAS as a SAS XPORT file. After that, we can simply use the following function:

```
> mydata <-read.xport(c:/datafile.xpt)
```

For Minitab files

```
library(foreign)
myMTPData <- read.mtp("example2.mtp")
```

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. R has built in function to print months.
month.abb
month.name

Dates in R

R provides several options for dealing with date and date/time data. The built in "as.Date" function handles dates (without times); the contributed library "chron" handles dates and times, but does not control for time zones; and the "POSIXct" and "POSIXlt" classes allow for dates and times with control for time zones. The general rule for date/time data in R is to use the simplest technique possible. Thus, for date only data, "as.Date" will usually be the best choice. If you need to handle dates and times, without timezone information, the "chron" library is a good choice; the POSIX classes are especially useful when timezone manipulation is important.

Except for the POSIXlt class, dates are stored internally as the number of days or seconds from some reference date. Thus dates in R will generally have a numeric mode, and the class function can be used to find the way they are actually being stored. The POSIXlt class stores date/time values as a list of components (hour, min, sec, mon, etc.) making it easy to extract these parts.

Dates in R

The as.Date function allows a variety of input formats through the format= argument. The default format is a four digit year, followed by a month, then a day, separated by either dashes or slashes. **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())

```
> as.Date('1920-6-15')
[1] "1920-06-15"
> as.Date('1990/02/17')
[1] "1990-02-17"
> as.Date('1/15/2001',format='%m/%d/%Y')
[1] "2001-01-15"
> as.Date('April 26, 2001',format='%B %d, %Y')
[1] "2001-04-26"
```

Dates in R

Commonly used codes in R

Code	Meaning	Code	Meaning
%a	Abbreviated weekday	%A	Full weekday
%b	Abbreviated month	%B	Full month
%c	Locale-specific date and time	%d	Decimal date
%H	Decimal hours (24 hour)	%I	Decimal hours (12 hour)
%j	Decimal day of the year	%m	Decimal month
%M	Decimal minute	%p	Locale-specific AM/PM
%S	Decimal second	%U	Decimal week of the year (starting on Sunday)
%w	Decimal Weekday (0=Sunday)	%W	Decimal week of the year (starting on Monday
%x	Locale-specific Date	%X	Locale-specific Time
%y	2-digit year	%Y	4-digit year
%z	Offset from GMT	%Z	Time zone (character)

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 . Note 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")
```

```
>as.Date(78468, origin = "1800-01-01",format="%m/%d/%Y")
```