**Reading Data from File**

The data is first collected, and may be pre-processed using software, be it in 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 or CSV file and then import this file in to R.

Functions for importing data,

***read.table()*** Reads a file in table format and creates a dataframe

***read.csv()*** Same as read.table() where sep=","

***read.fwf()*** Read a table of fixed width formatted data. Data that is not delimited need to specify the width of the fields

The ***read.table*** function reads a file in table format and automatically creates a dataframe from it, with cases corresponding to rows (lines) and variables to columns (fields) in the file.

**Saving the File from Excel**

Once you have made your dataframe in Excel and corrected all the inevitable data-entry and spelling errors, then you need to save the dataframe in a file format that can be read by R. Much the simplest way is to save all your dataframes from Excel as tab-delimited text files: from the ‘Save as type’ options choose ‘Text (Tab delimited)’. There is no need to add a suffix, because Excel will automatically add ‘.txt’ to your file name.

Note that read.table would fail if there were any spaces in any of the variable names in the first row of the dataframe

* The whole path and file name needs to be enclosed in double quotes: “c:\\data.txt”.
* header=T says that the first row contains the variable names.
* Always use double backslash \\ rather than \ or forwardslash / in the file path.
* Replace the blanks with NA in Excel(or use a different separator symbol)
* The title should have single word (state.name instead of state name)
* If you export the file from Excel using commas to separate the fields use sep= “,”
* If you export the file from Excel using space to separate the fields use sep= “ ”
* If you export the file from Excel using tabulation to separate the fields use sep= “\t”

Note when you are trying to read variables that consist of character strings containing blank spaces (as in files containing place names). You can use read.table so long as you export the file from Excel using commas to separate the fields, and you tell read.table that the separators are commas using sep=","

***>example <-read.table("c://data.csv",header=T,sep=",")***

but it is quicker and easier to use read.csv in this case(and there is no need for header=T)

***>example<-read.csv("c://data.csv")***

Instead of typing header=T in read.table functions, we can use

***>example<-read.delim("c://file.txt")***

**Separators and Decimal Points**

The default field separator character in read.table is sep=" ". This separator is a space, which is produced by one or more spaces, one or more tabs \t, one or more newlines \n, or one or more carriage returns.

**From SPSS**

library(Hmisc)  
mydata <- spss.get("c:/mydata.por", use.value.labels=TRUE)

**From SAS**

library(Hmisc)

mydata <- sasxport.get("c:/mydata.xpt")

**From Stata**

library(foreign)

mydata <- read.dta("c:/mydata.dta")

**From systat**

library(foreign)

mydata <- read.systat("c:/mydata.dta")

**Read a file directly from the web**

We can import data files into R for analysis directly from the website.

Example:

> site<-"http://people.sc.fsu.edu/~jburkardt/datasets/census/census\_2010.txt"

> data <- read.table(site, header=F)

**Interactive data frame entry in R**.

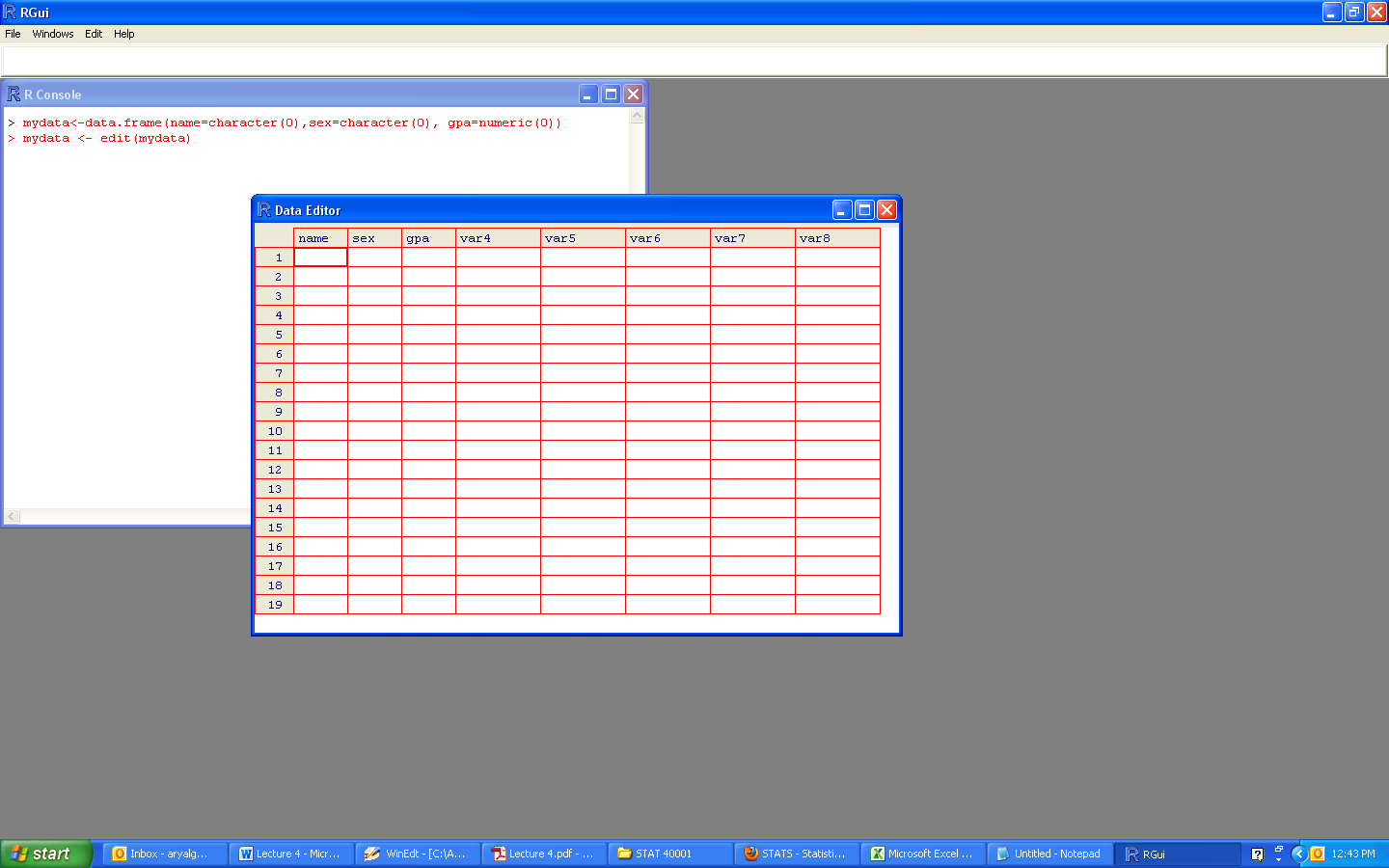
R also has a built-in spreadsheet that you can use to enter data frames interactively, bypassing the need for Excel altogether. To access it, you must use the ***data.frame()*** and ***edit()*** commands in R.

Example:

Suppose we want to create a data frame with three variables: name, sex, and gpa. The name and sex variable are 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)*



**Exporting Results**

Once the analysis has been conducted and the results are obtained, it is often important to send the results to other format readable to other software. For this propose a .text file is most appropriate format.

>write.table(result, "myfile.csv", sep= ",", row.names=FALSE)

The object ***result*** is exported to the file myfile.csv in the current working directory.

The exportation above can be used to control the column separator fixed at “,” and the file name myfile.csv. It is also possible to control other options such as

* Whether or not the results contain the column names (col.names); Note that by default ***col.names=TRUE***
* Whether or not the results contain the row names (row.names); Note that by default ***row.names=TRUE***
* Whether or not character strings are defined by speech marks; by default ***quote=TRUE***
* The decimal point (***dec***) which is “.” by default and the character string for missing values (***na***) which is “NA” by default.

Hence, to export without speech marks, without the row and column names and the tabulation separator, we use

> write.table(result, "myfile.txt",quote=FALSE, row.names=FALSE, col.names=FALSE, sep="\t")

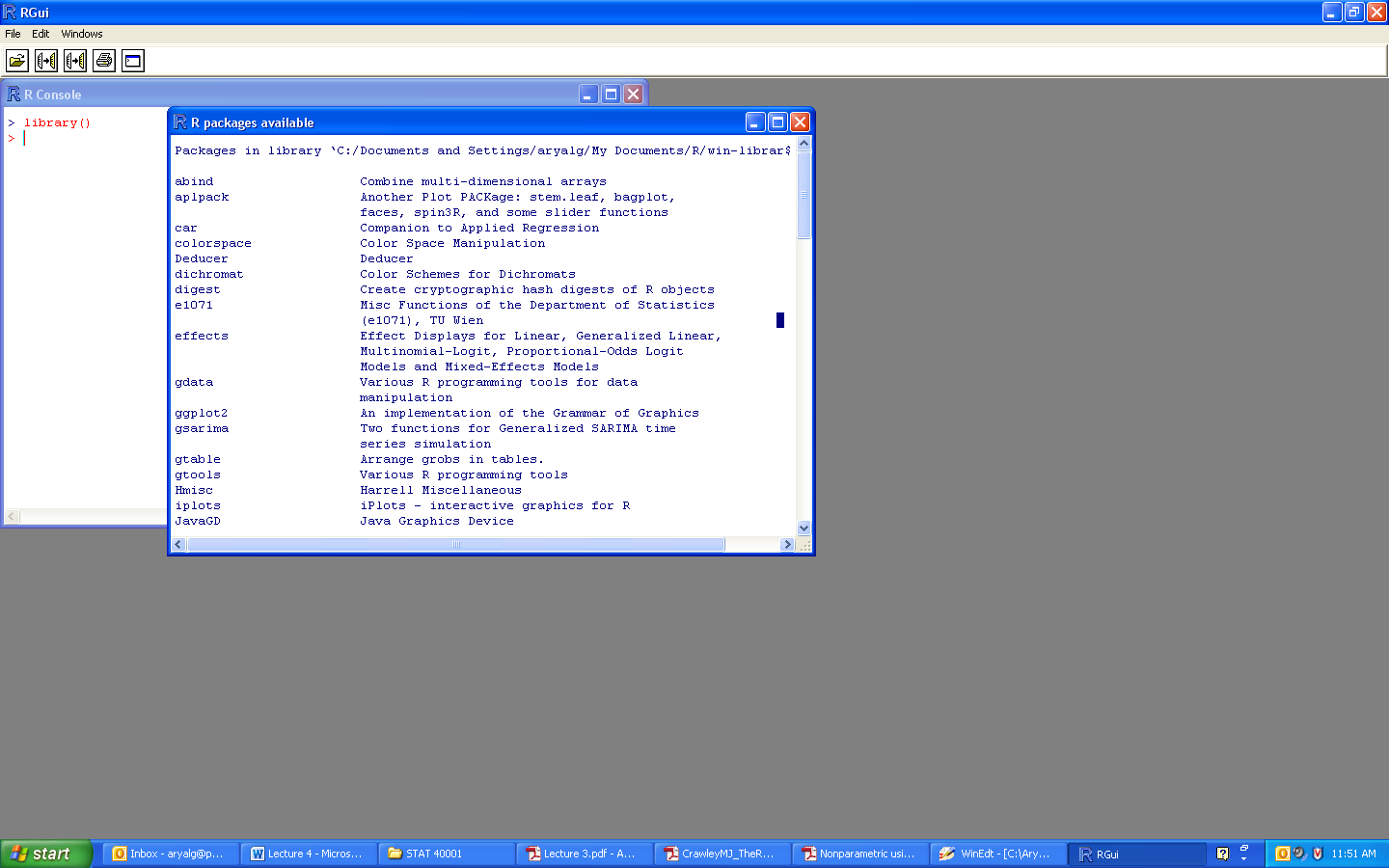
Note that ***write.csv*** is a possible alternative to the ***write.table*** function.

**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.

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).*

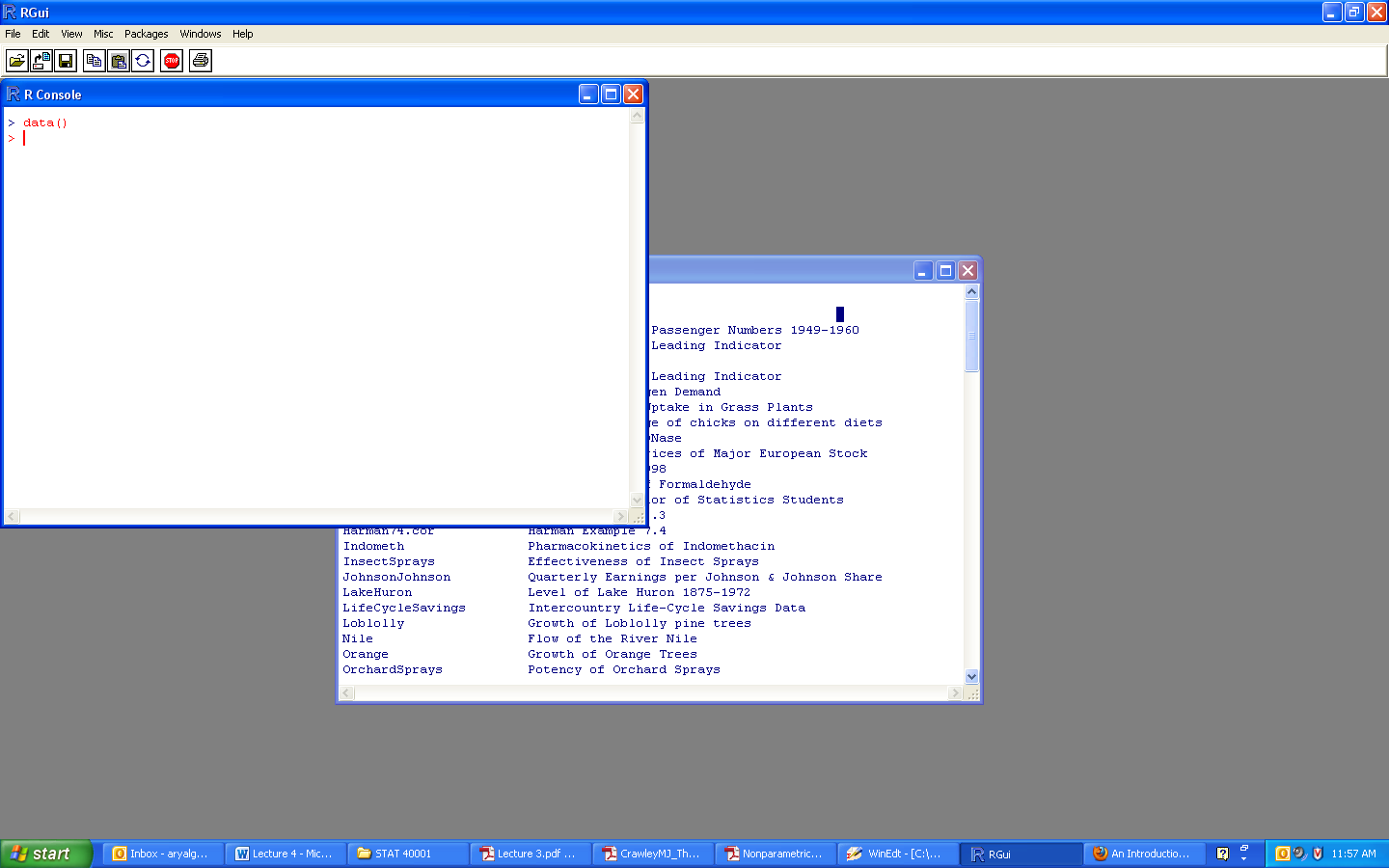
For example: >library(zoo)

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()*



All the datasets supplied with R are available directly by name. However, many packages still use the obsolete convention in which data was also used to load datasets into R, for example

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

In most cases this will load an R object of the same name. However, in a few cases it loads several objects, so see the on-line help for the object to see what to expect.

> data(AirPassengers)

> 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

**Loading data from other R packages**

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

> data(package="rpart")

> data(Puromycin, package="datasets")

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

**Dates and Time 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 datein the longest time scale

> Sys.time()

[1] "2013-08-28 21:26:21 CDT"

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

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

[1] "2013-08-28"

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

[1] "21:26:21"

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

> unclass(Sys.time())

[1] 1130679208

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()

[1] "Wed Aug 28 21:31:38 2013"

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

[1] "2013-08-28 21:33:49 CDT"

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$wday

[1] 3

> date$yday

[1] 239

> date$isdst

[1] 1

It is easier to use ***unclass*** with ***unlist*** to view all of the components of date:

> unlist(unclass(date))

sec min hour mday mon year wday yday isdst

49.01562 33.00000 21.00000 28.00000 7.00000 113.00000 3.00000 239.00000 1.00000

> round(unlist(unclass(date)),0)

sec min hour mday mon year wday yday isdst

49 33 21 28 7 113 3 239 1

**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 >=. We can add or subtract a number of seconds or a time object from a date-time object, but

two date-time objects can not be added. Subtraction of two date-time objects is equivalent to using ***difftime*** . Unless a time zone has been specified, POSIXlt objects are interpreted as being in the current time zone in calculations.

Note that dates and times into must be converted to POSIXlt objects before starting to do any calculations. Once they are POSIXlt objects, it is straightforward to calculate means, differences and so on.

Example: 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**

> difftime("2005-09-20","2003-09-22")

Time difference of 729 days

In order to print only the number of days, we can use

>as.numeric(difftime("2005-09-20","2003-09-22"))

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"

Below is a list of POSIX standard for ***strptime***

%a Abbreviated weekday name

%A Full weekday name

%b Abbreviated month name

%B Full month name

%c Date and time, locale-specific

%d Day of the month as decimal number (01–31)

%H Hours as decimal number (00–23) on the 24-hour clock

%I Hours as decimal number (01–12) on the 12-hour clock

%j Day of year as decimal number (001–366)

%m Month as decimal number (01–12)

%M Minute as decimal number (00–59)

%p AM/PM indicator in the locale

%S Second as decimal number (00–61, allowing for two ‘leap seconds’)

%U Week of the year (00–53) using the first Sunday as day 1 of week 1

%w Weekday as decimal number (0–6, Sunday is 0)

%W Week of the year (00–53) using the first Monday as day 1 of week 1

%x Date, locale-specific

%X Time, locale-specific

%Y Year with century

%Z Time zone as a character string (output only)