THE UNIVERSITY OF MEMPHIS

Computer Science

COMP7/8150: Fundamentals of Data Science

Week 4: Tutorial in R

Max H. Garzon

Data Life Cycle

- Problem Definition/goal
  - Identify/specify goals of the data analysis
  - commit to specific deliverables
- Data pre-processing
  - Identify appropriate data
  - Acquire data (gather, lookup, understand)
- Data processing
  - Identify methods (gather, cleanse, store)
  - Carry out the analysis (patterns, trends, predictions?)
- Data post-processing
  - Visualize and present
  - Deploy and evaluate. Iterate, if necessary

Learning Objectives

- To identify the requirements to install and run the package R.
- To identify the basic structure of an R script and how to run it
- To identify the basic data structures in R and learn how to use them for DS
- To identify the basic I/O commands to enter data for processing
- To identify the basic I/O commands to display/visualize to view results

Introduction – Why R ?

- R is a statistical programming language and environment for data manipulation, calculation and graphical display.
  - many useful operators for arrays and matrices.
  - many handy tools for interactive data analysis.
  - great graphical facilities for data analysis.
  - a programming language with conditionals, loops, user defined functions and input and output facilities

Features of R

- R is an interpreted computer language.
  - branching and looping as well as modular programming using functions.
  - user-defined functions in R are usually written in R, calling upon a smaller set of internal primitives.
  - allows user interface to procedures written in C, C++ or FORTRAN languages
    - for efficiency
    - write additional primitives

Strength of R: What R can do ?

- data handling and manipulation: numeric, textual and many matrix operations
- high-level data analytic and statistical functions
- simple to produce great graphics
- programming language: loops, branching, subroutines
- it is free and it has a strong user-support

Weaknesses of R

- R is not a database, but it can be connected to DBMSs
- R is basically a command-line interface but some package like Rcmdr can provide nice graphical user interfaces.
- R is an interpreted language which can be very slow, but you can call own C/C++ code from R.
- R lacks many spreadsheet features, but R can input/output data from/to Excel

THE UNIVERSITY OF MEMPHIS
Computer Science

Data Analysis and Presentation

- The R distribution contains functionality for large number of statistical procedures.
  - linear and generalized linear models
  - nonlinear regression models
  - time series analysis
  - classical parametric and nonparametric tests
  - clustering
  - smoothing
- R also has a large set of functions which provide a flexible graphical environment for creating various kinds of data presentations.

Getting help

Details about a specific command whose name you know (input arguments, options, algorithm, results):

```
>? t.test
or
>help(t.test)
```

R Information - Help for `t.test'

File Edit View

```
t.test                  package:ctest                  R Documentation

Student's t-Test

Description:

     Performs one and two sample t-tests on vectors of data.

Usage:

     t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
            mu = 0, paired = FALSE, var.equal = FALSE,
            conf.level = 0.95, ...)
     t.test(formula, data, subset, na.action, ...)

Arguments:

       x: a numeric vector of data values.

       y: an optional numeric vector data values.

alternative: a character string specifying the alternative hypothesis,
          must be one of `"two.sided"' (default), `"greater"' or
          `"less"'.  You can specify just the initial letter.

      mu: a number indicating the true value of the mean (or difference
          in means if you are performing a two sample test).

  paired: a logical indicating whether you want a paired t-test.

var.equal: a logical variable indicating whether to treat the two
```

Documentation and help file in R

- All the R functions have been documented in the form of help pages in an "output independent" form which can be used to create versions for HTML, LATEX, text etc.
  - The document "An Introduction to R" provides a more user-friendly starting point.
  - An "R Language Definition" manual
  - More specialized manuals on data import/export and extending R.

Standard packages in R

- Classical and modern statistical techniques have been implemented.

- There are several packages supplied with R (called “standard” packages) and many are available through internet sites (such as http://cran.r-project.org)

- `install.packages()`
  lists packages available to install over the internet

Issuing commands in R

- Start R: click the icon of R after you have successfully installed the R.
- When R is started, it will prompt (>) and you can type in any R command.
- After you finished typing in a R command, just hit Enter key.
- After R finished excuting your command, it will display a prompt (>) for your next command.
- q() – quits R, you will be asked whether to save workspace created.

The Workspace

- The workspace contains any user-defined objects that you might have created during an open session of R.
  - Data frames, matrices, vectors, lists
  - Functions
- Workspace is saved as a ".RData" file.
- You will want to know where your workspace is saved.

Working directory in R

- getwd() – displays current working directory
- setwd("PATH") – sets the working directory to PATH. Useful to work on different projects.
- `> getwd()`

```
[1] "C:/Documents and
Settings/LYD/My Documents"
```

- `> setwd("C:/class/7150-2011/hw1")`
- `> getwd()`

```
[1] "C:/class/7150-2011/hw1"
```

Storing data

- Every R object can be stored into and restored from a file with the commands “save” and “load”.
- This uses the XDR (external data representation) standard of Sun Microsystems and others, and is portable between MS-Windows, Unix, Mac.

```
> save(x, file=“x.Rdata”)
> load(“x.Rdata”)
```

Managing objects in Workspace

- `ls()` – lists all objects currently in the workspace
- `rm()` – removes the object specified.
- `> ls()`

  `[1] "WD"`
- `> rm(WD) ##or rm("WD")`
- `> ls()`

  `character(0)`

Command History

- You can save all the commands executed in R by saving your command history
- Click File, then click "Save History…"
- Choose directory where you want to save then click OK.
- Command history is saved in a ".RHistory" file
- `history()` lists last 25 commands
- `history(max.show=Inf)` lists all commands

Built-in dataset in R

- R has many built-in datasets that you do not have to create by yourself.
- For example, R has dataset, called mtcars, from 1974 *Motor Trend* US magazine, for fuel consumption (mpg) and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).
- To see the list and description of the built-in datasets, type `data()`

mtcars data listing

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

Partial listing of a dataset

- you can use `head(d,n), tail(d,n), print(d)` (or simply d) to display the first n, bottom n and all (if not too many) of the dataset d.
- `> head(mtcars, 2)`

```
              mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

- `> tail(mtcars, 2)`

```
               mpg cyl disp  hp drat   wt qsec vs am gear carb
Maserati Bora 15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
Volvo 142E    21.4   4  121 109 4.11 2.78 18.6  1  1    4    2
```

Special characters in R

- `#` #user's comment
- `<-` #assignment statement (also allowed: `=` `->` `<<-` `->>`)
  - we will use only `<-` for assignment.
- `[ ]` # indexing of arrays, matrices, dataframes, lists
- `( )` # encloses function input variables/arguments
- `{ }` # groups statements (e.g. loops, functions, defs
- `;` # separates several statements on a single line
- `$` # extracting elements from lists or data frames

  "`$`" is similar to "." in other languages like C/C++/Java.

Variable names

- Like many modern languages (C, C++, Java), the variable names are case-sensitive.
- While R does not have a concept of "reserved words", several variable/function names are better treated as of "reserved words" manly for the purpose of readability.
  - e.g. one-letter "reserved words": `c`, `q`, `t`, `C`, `D`, `F`, `I`, and `T`.
  - `c` (concatenate), `q`(quit), `t`(transpose of matrix), `F`(false), `T`(true), `D`(derivative), …

Basic data types in R

Primitive (or: atomic) data types in R are:

- numeric (integer, double, complex)
- character
- logical
- function

We can build vectors, arrays, lists from basic data types.

The primary data type in R is *vector*.

Operators in R

- `> x <- 2 ; y <-3`
- `> x + y`
  `[1] 5`
- `> x * y`
  `[1] 6`
- `> x / y  #default is floating point division`
  `[1] 0.6666667`
- `> x %/% y # integer division`
  `[1] 0`
- `> y %/% x # integer division`
  `[1] 1`
- `> x ^ y`
  `[1] 8`

Useful functions on strings

- `paste()`
  - # concatenates and converts to string
- `substr(), strsplit()`
  - # substrings and splitting strings
- `grep(), gsub()`
  - # finds matches, replaces matches in a string
- `tolower(), toupper()`
  - # uppercase, lowercase conversion
- `nchar()`
  - # number of characters in string

Example of string functions in R

- `> substr("abcdef",2,4)`
  `[1] "bcd"`
- `> x <- "This is a"`
- `> y <- "test only"`
- `> z <- paste(x,y); z`
  `[1] "This is a test only"`
- `> toupper(z)`
  `[1] "THIS IS A TEST ONLY"`
- `> nchar(z)`
  `[1] 19`
- `> w <- paste(z, "your score is", 90); w`
  `[1] "This is a test only your score is 90"`

Concatenation and selection

```
> x <- c(2, 3, 4)
> y <- c(6, 9, 2)
> z <- c(x, y); z
[1] 2 3 4 6 9 2
> x[c(1,3)]
[1] 2 4
> x[-2]
[1] 2 4
```

```
> length(z)
[1] 6
> x + y
[1]  8 12  6
> x / y
[1] 0.3333333 0.3333333
2.0000000
> x %/% y
[1] 0 0 2
```

Simple functions in R

- `> x <- c(2,3,4)`
- `> sin(x)`
```
[1]  0.9092974  0.1411200 -0.7568025
```
- `> cos(x)`
```
[1] -0.4161468 -0.9899925 -0.6536436
```
- `> sin(x)^2+cos(x)^2 #why ? all = 1`
```
[1] 1 1 1
```
- `> log(x)`
```
[1] 0.6931472 1.0986123 1.3862944
```
- `> exp(x)`
```
[1]  7.389056 20.085537 54.598150
```
- `> log10(x)`
```
[1] 0.3010300 0.4771213 0.6020600
```

Missing values and NaNs

R has some special values

- `NA` represents a missing value in the dataset
- `NaN` (not a number) because of the mathematical operations such as 0/0.
- `Inf` (positive infinity) e.g. 1/0
- `-Inf` (negative infinity) e.g. log(0)
- `NULL` is an empty vector or array.

We can check them by

- `is.infinite(x)`
- `is.nan(x)`
- `is.na(x)`

Sequence generation in R

- Common ways to generate a sequence:
  - `from:to` # increment ±1.
  - `seq(from, to, by= gap)` increment or length can be specified
  - `rep(d,n)` # replicate d n times.
- `> x <- 9:5; x`
  `[1] 9 8 7 6 5`
- `> y <- seq(0.9,0.5, -0.1); y`
  `[1] 0.9 0.8 0.7 0.6 0.5`
- `> z <- rep(x, 2); z`
  `[1] 9 8 7 6 5 9 8 7 6 5`

Logical comparisons in R

- Comparing x and y (vector or scalar) with logical comparison, it will yield a vector of True/False.
  - `x<y, x<=y`
    - #x is less than, less or equal to, y
  - `x>y, x>=y`
    - #x is greater than, greater or equal to, y
  - `x == y, x!=y`
    - # x equal, not equal to, y

Logical operations in R

- We can use some logical operators for conditional expression:
  - `!, &, | , xor(x,y)`
    - # not, and, or, exclusive or
  - `any()`
    - #   true if any of a vector is true
  - `all()`
    - #   true if all values of a vector are true

.. Logical operations in R

- `> x <- c(1, 5, 7, 6); y <- c(2, 6, 4, 3)`
- `> x > 3 & x < 7`
  `[1] FALSE  TRUE FALSE  TRUE`
- `> x <= y`
  `[1]  TRUE  TRUE FALSE FALSE`
- `> x[x <= y]`
  `[1] 1 5`
- `> (x > 3) | (y < 4)`
  `[1] TRUE TRUE TRUE TRUE`
- `> (x > 3)`
  `[1] FALSE  TRUE  TRUE  TRUE`
- `> (y < 4)`
  `[1]  TRUE FALSE FALSE  TRUE`
- `> (x > 3) & (y < 4)`
  `[1] FALSE FALSE FALSE  TRUE`

Vectors and arrays

- vector is the simplest data structure used in R which is created using c() function.
- array is an ordered collection of data of the same type with an integer as its index.
  - an array can have many dimensions.
  - matrix is simply a 2-dim array.

Using array in R

- ```
  > x <- c(3, 5, 7, 11, 13, 19); x
  [1]  3  5  7 11 13 19
  ```
- ```
  > y <- array(x, dim=c(2,3)); y
        [,1] [,2] [,3]
  [1,]     3    7   13
  [2,]     5   11   19
  ```
- ```
  > dim(x) <- c(3,2); x
       [,1] [,2]
  [1,]    3   11
  [2,]    5   13
  [3,]    7   19
  ```
  ```

List in R

- List in R is an object consisting of a collection of objects (components) of (possibly) different types.
- The entry of the list index is usually by some names as the key.
- It can also referenced by its position with an integer.

Using list in R

- ```
  > customer  <- list(name="Fred", wife="Mary",
  + no.children=3, child.ages=c(4,7,9))
  ```
- ```
  > customer$name
  [1] "Fred"
  ```
- ```
  > customer$child.ages
  [1] 4 7 9
  ```
- ```
  > customer[2]
  $wife
  [1] "Mary"
  ```
- ```
  > customer[[2]]
  [1] "Mary"
  ```
  ```

Creating matrix

- ```
  > M1 <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol=3,
  + byrow=TRUE, dimnames = list(c("row1", "row2"),
  + c("C.1", "C.2", "C.3")))
  ```
- ```
  > M1
       C.1 C.2 C.3
  row1   1   2   3
  row2  11  12  13
  ```
- ```
  > M2 <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol=3,
  + dimnames = list(c("row1", "row2"),
  + c("C.1", +"C.2", "C.3")))
  ```
- ```
  > M2
       C.1 C.2 C.3
  row1   1   3  12
  row2   2  11  13
  ```

Matrix operations

- ```
  > M1 + M2
       C.1 C.2 C.3
  row1   2   5  15
  row2  13  23  26
  ```
- ```
  > M1 * M2 # element-wise multiplication
       C.1 C.2 C.3
  row1   1   6  36
  row2  22 132 169
  ```
- ```
  > t(M2)
      row1 row2
  C.1    1    2
  C.2    3   11
  C.3   12   13
  ```
- ```
  > M1 %*% t(M2) #  multiplication
       row1 row2
  row1   43   63
  row2  203  323
  ```

Other matrix functions/op.

- `dim(A)`
  - #returns dimension of matrix or array A
- `nrow(A),ncol(A), NROW(A),NCOL(A)`
  - #number of rows and columns of matrix A
- `rownames(A),colnames(A)`
  - #names of rows and columns of matrix A
- `%*%`
  - # matrix multiplication

Other matrix functions/op.

- `t(A)` # transpose of matrix A
- `solve(A)` # inverse of matrix A
- `svd(A),qr(A),chol(A)`
  - # singular value, QR, cholesky decomposition of matrix A
- `eigen(A),det(A)`
  - # eigenvalues and eigenvectors, determinant of matrix A

Combining matrices and arrays

- `cbind(x,y)`
- \# binds matrices, dataframes,… columnwise
- `rbind(x, y)`
- \# binds matrices, dataframes,… rowwise

```
> x <- c(1, 2,7,9); y <- 5:8
> cbind(x,y)
     x y
[1,] 1 5
[2,] 2 6
[3,] 7 7
[4,] 9 8
> rbind(x,y)
  [,1] [,2] [,3] [,4]
x    1    2    7    9
y    5    6    7    8
> c(x,y)
[1] 1 2 7 9 5 6 7 8
```

THE UNIVERSITY OF MEMPHIS
Computer Science

Data frames

data frame is a rectangular table with rows and columns; data within each column has the same type (e.g. number, text, logical), but different columns may have different types.

data.frame():

- an R command to create data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists,
- used as the fundamental data structure by most of **R**'s modeling software.

THE UNIVERSITY OF MEMPHIS Computer Science

Creating data frames

- You can recreate a data frame from scratch by

```
my_data <- edit(data.frame())
```

  that you can enter data into the given form.
- You can also import from external file (to be discussed later) or you can save the data created.
- ```
  > my_data <- data.frame(x,y); my_data
    x y
  1 1 5
  2 2 6
  3 7 7
  4 9 8
  ```
  ```

Data subsetting in R

- `x[n], x[-n]` # select nth element, all but nth element from vector x
- `x[1:n], x[-(1:n)]` # select first n elements, all but first n elements from x
- `x[c(1,4,6)]` # select element 1,4 and 6 from vector x
- `x[x>3  & x<5]` # select elements that meet condition
- `which(x==3)` # returns indices to values x that meet the condition

.. Data subsetting in R

```
> x <- c(2, 5, 7, 11, 13, 17)
> x[3]
[1] 7
> x[-3]
[1]  2  5 11 13 17
> x[1:3]
[1] 2 5 7
> x[-(1:3)]
[1] 11 13 17
> x[c(1,4,6)]
[1]  2 11 17
> x[-c(1,4,6)]
[1]  5  7 13
> which(x==13)
[1] 5
```

Subsetting matrix/data frame in R

- Same rule for vector subsetting can be used for matrix or data frame (to be discussed later)
- `A[i,j], A[,j], A[i,]` # selects element i,j, the jth column, i-th row from matrix A
- `A[,cols]` # selects columns cols from matrix A
- `A["name",]` # selects row named "name" from matrix A
- `D$name, D[["name"]]` # selects column named "name" from data frame D

..Subsetting matrix/data frame in R

- ```
  > mtcars[1:4,1:5]
                  mpg cyl disp  hp drat
  Mazda RX4      21.0   6  160 110 3.90
  Mazda RX4 Wag  21.0   6  160 110 3.90
  Datsun 710     22.8   4  108  93 3.85
  Hornet 4 Drive 21.4   6  258 110 3.08
  ```
- ```
  > mtcars[1:4,1]
  [1] 21.0 21.0 22.8 21.4
  ```
- ```
  > mtcars[1:4, "mpg"]
  [1] 21.0 21.0 22.8 21.4
  ```
- ```
  > mtcars$mpg[1:4]
  [1] 21.0 21.0 22.8 21.4
  ```

if statements in R

- `if, else, else if`
  - #conditionally execute statements
  - #useful only when comparing two values, not two vectors (why not?)
  - # often used with `all()` or `any()`
- R example:
- `if(all(x < 0)) cat("all x values are negative\n")`

ifelse statement in R

- `ifelse(cond,yes,no)`
  - # if (component-wise) condition is true/false, executes (component-wise) statement ‘yes’/’no’
- R example:
- `x <- c(6:-4)`
- `sqrt(x)#- gives warning`
- `sqrt(ifelse(x >= 0, x, NA))# no warning`

Repetitive execution

- `for (el in seq) {expr}`
  - #repeat expr for each element in seq
- `while (cond) {expr}`
  - #repeat expression while condition is true
  - #be very careful for vector comparison
- `repeat {expr}`
  - #repeat until break encountered

Breaking repetitive execution

- break
  - it terminates execution of for, while, repeat loops
  - it can be used to terminate any loop, possibly abnormally.
- next
  - it transfers execution to next iteration in loops
  - it can be used to discontinue one particular cycle and skip to the “next”.

Example

```
> s <- 0
> for(i in 1:4) {
      s <- s+ i^0.5
     print(s)
  }
[1] 1
[1] 2.414214
[1] 4.146264
[1] 6.146264
```

```
> i <- 1; s <- 0;
> while(s<=10) {
      s <- s+ i^0.5
      print(s); i <- i+1
  }
[1] 1
[1] 2.414214
[1] 4.146264
[1] 6.146264
[1] 8.382332
[1] 10.83182
```

User-defined functions

Example:

```
f <- function(a, b)
{
return (a+b)
}
```

Note:

- Note that `return` is a function in R; its argument must be contained in parentheses.
- The use of return is optional; otherwise the value of the last line executed in a function is its return value.

apply(arr, margin, fct)

Apply the function fct along some dimensions of the array arr, according to margin, and return a vector or array of the appropriate size.

```
> x
     [,1] [,2] [,3]
[1,]    5    7    0
[2,]    7    9    8
[3,]    4    6    7
[4,]    6    3    5
> apply(x, 1, sum)
[1] 12 24 17 14
> apply(x, 2, sum)
[1] 22 25 20
```

lapply(li, *function*)

- To each element of the list li, the function *function* is applied.

```
> li <- list("This","example","is","great")
> lapply(li, toupper)
[[1]]
[1] "THIS"

[[2]]
[1] "EXAMPLE"

[[3]]
[1] "IS"

[[4]]
[1] "GREAT"
```

.. sapply(li, fct)

`sapply` is a simplified version of `lapply` by default returning a vector or matrix if appropriate

```
> li <- list("This","example","is","great")
> sapply(li, toupper)
  [1] "THIS"    "EXAMPLE" "IS"       "GREAT"
> fct = function(x) { return(c(x, x*x, x*x*x)) }
> sapply(1:5, fct)
      [,1] [,2] [,3] [,4] [,5]
[1,]     1    2    3    4    5
[2,]     1    4    9   16   25
[3,]     1    8   27   64  125
```

Input output in R

- By default, the input is from the keyboard and output is to the screen. However, there are many other methods can be used.
- `write.table(x,file)`
  - # writes object x as a dataframe to a table
- `read.table(file)`
  - # reads table from space-delimited file, aligned in columns
- `read.csv(file),read.delim(file)`
  - # reads table comma- delimited or tab-delimited file

Reading data from files

Price	Floor	Area	Rooms	Age	Cent.heat
52.00	111.0	830	5	6.2	no
54.75	128.0	710	5	7.5	no
57.50	101.0	1000	5	4.2	no

...

- `HousePrice <- read.table("houses.data", header=TRUE)`

Importing and exporting data

There are many ways to get data into R and out of R.
Most programs (e.g. Excel), as well as humans, know how to deal with rectangular tables in the form of tab-delimited text files.

```
> x = read.delim("filename.txt")
also: read.table, read.csv

> write.table(x, file="x.txt", sep="\t")
```

Importing data

- Type conversions: by default, the read functions try to guess and autoconvert the data types of the different columns (e.g. number, factor, character).
  - There are options as.is and colClasses to control this – *read the online help*
  - Understand the conventions your input files use and set the quote options accordingly.

Further Topics

Some of the topics listed will be discussed in the later modules.

- Graphics in R (page 25 in [MB], much more on Chapter 15)
- Lattice graphics (page 30 in [MB], skip)
- Finer graphic parameter settings (page 27, [MB])
- Customized options setting (page 34, [MB])

Questions?