Introduction to the R Project for Statistical Computing

Adjunct Associate Professor
School of Integrative Plant Sciences, Section of Soil & Crop Sciences
Cornell University.

D G Rossiter

November 27, 2021

Cornell University
College of Agriculture and Life Sciences

Topics – Part 1

1. The R Project for Statistical Computing: what and why?

2. Installing R and RStudio

3. Interacting with R

4. The S language: expressions, assignment, functions

5. The R help system, R manuals, on-line R help

6. Finding, installing and loading contributed packages

7. Finding and loading example datasets

The R Project for Statistical Computing: what and why?

- **R** is an **open-source** environment for **statistical computing, data manipulation and visualisation**;
- Statisticians have implemented over 2 000 specialised statistical procedures as **contributed packages**;
- R and its packages are **freely-available over the internet**;
- R runs on many **operating systems**, including Microsoft Windows, Unix© and derivatives Mac OS X and Linux;
- R is **fully programmable**, with its own modern computer language, **S**;
- Repetitive procedures can be automated by user-written **scripts**, **functions** or **packages**;
- ...

- . . .

- R is supported by comprehensive **technical documentation**, user-contributed tutorials and textbooks; these all have freely-available **R code**

- R is the *lingua franca* ([U+666E] [U+901A] [U+8BDD]) of the computational statistics world.

- R can **import and export** in MS-Excel, text, fixed and delineated formats (e.g. CSV), with databases . . . ;

- R is a major part of the **open source** and **reproducible research** movement for transparent and honest science.

Installing R and RStudio

- **R** is the the computing environment; **RStudio** is an Integrated Development Environment (IDE) which makes using R easier

- **Install R first**; it can run outside RStudio
  - The Comprehensive R Archive Network (CRAN): `http://cran.r-project.org/` to download R, packages and documentation
  - link "Download R for ..." (Linux, Mac OS/X, Windows)
  - Install the "base" version

- **Install RStudio** from its home page `http://www.rstudio.com/`
  - link "Download RStudio" desktop open-source version

- **Start RStudio**; it will automatically start R.

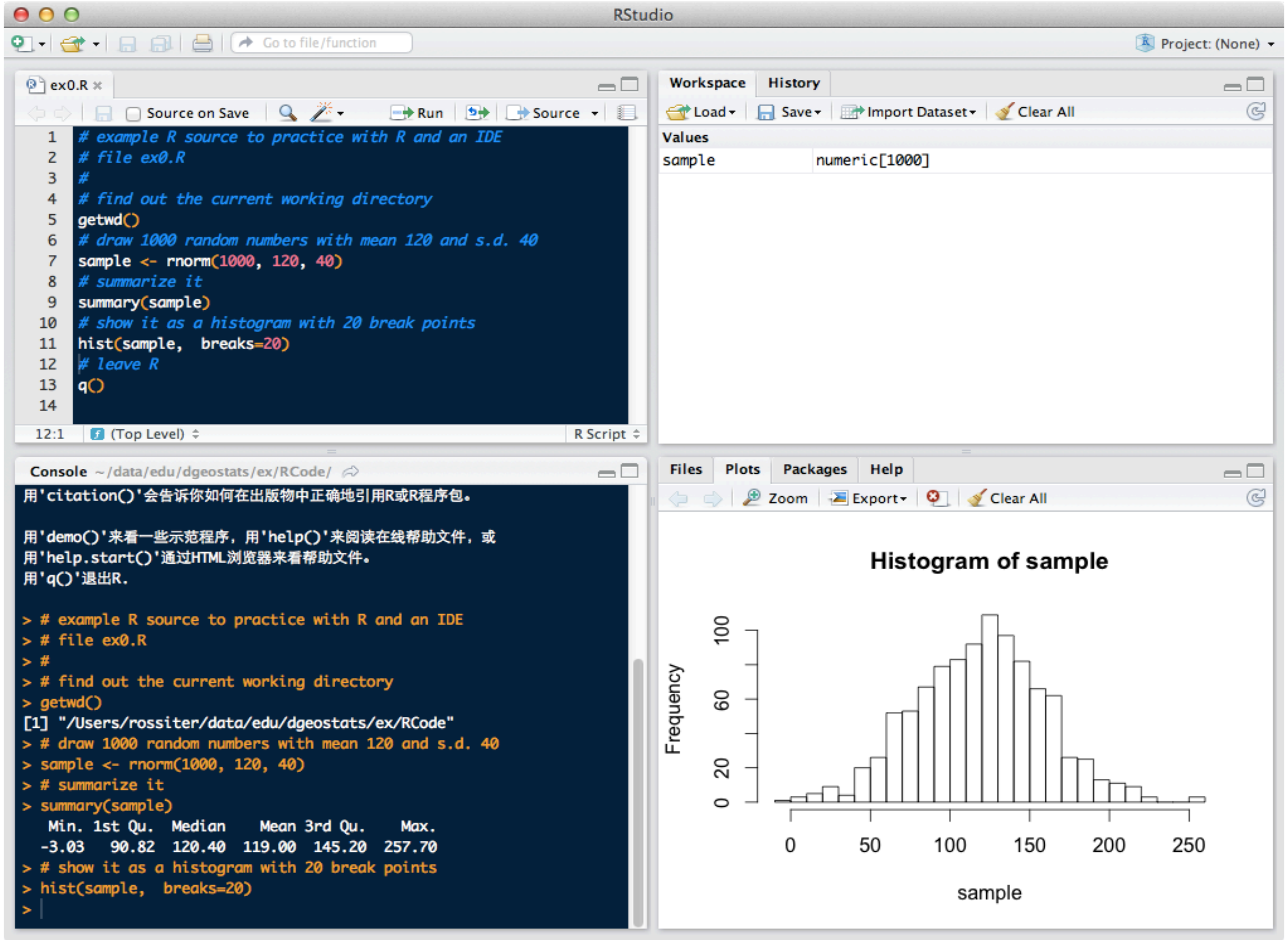
RStudio Features (1/2)

- R console
  - **enter** R commands here, see **text output**
- Code editor
  - write one or more R commands, pass the commands to the console and see the text output there
  - advantage: can **edit and re-run**
  - can save the script to reproduce the analysis
- Graphics viewer (“Plots”)
  - shows output of commands that produce **figures**
  - can save for printing or inclusion in reports

RStudio Features (2/2)

- Workspace viewer
  - shows the **objects** in your workspace
- File manager
- History viewer
- Package manager
  - install (from CRAN) and load (in your workspace) additional **packages**
- Integrated **help** system
- Project manager
  - can switch between data analysis projects, each in its own directory

RStudio Screenshot



Basic interaction with the R console

- > is a **prompt**: R is waiting for input:

```
>
```

- You can type directly after the prompt; press the `Enter` to submit the command to R
- If a command is not syntactically-complete, R will show the **continuation** prompt:

```
+
```

- When the command is complete, R will execute
- Better: type a command in the **code editor** and click the `Run` button or press `Alt+Enter` to pass the command to the console
- Text output (if any) will appear in the console; figures will appear the graphics window

First interaction with the console

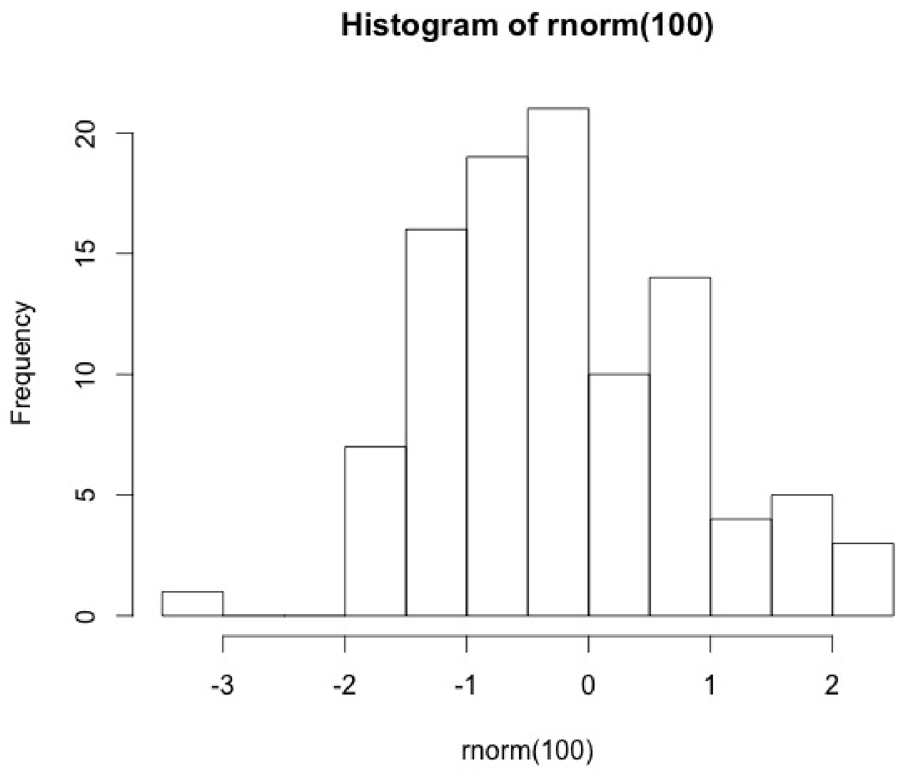
Draw 100 normally-distributed random numbers ($\mu = 0, \sigma^2 = 1$), summarize them:

```
> summary(rnorm(100))

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-2.36000 -0.85350 -0.06113 -0.14620  0.58610  2.47700
```

Draw another set of 100 and graph them as a histogram:

```
> hist(rnorm(100))
```



Your results will be different – why?

The S language

1. Origin; R vs. S
2. Expresssions
3. Assignment and the workspace
4. Functions

Origin of S

- The language implemented in the R environment is **S**
- Developed at Bell Laboratories (USA) in the 1980's (John Chambers etc.)
- Designed for **"programming with data"**, including statistical analysis
- Line between "user" and "programmer" purposely blurred
- Syntax similar to ALGOL-like programming languages (C, Pascal, and Java . . .)
- Operators,functions and methods are generally **vectorized**; vector and matrix operations are expressed naturally
- **Statistical models** specified with a standard notation

Origin of R

- 1990–1994 **R**oss Ihaka, **R**obert Gentleman at Univ. Auckland (NZ), for own teaching and research
- Syntax from **S**, internals from **Scheme** (a LISP-like functional programming language)
- 1997 Kurt Hornik and Fritz Leisch establishe the **CRAN** (**C**omprehensive **R** **A**ction **N**etwork) archive at TU Vienna
- 2000 V1.0 official release
- **R Core Team** of developers (Ripley, Dalgaard, Lumley, Tierney, Plummer . . .)
- S3 and then S4 **object-oriented** systems (V2, V3)
- Independent package developers
- 2015 Microsoft aquires Revolution Analytics `https://mran.revolutionanalytics.com` – still open-source but "industrial-level" support for Big Data projects

Expressions

R can be used as a command-line calculator; these S **expressions** can then be used anywhere in a statemnt.

```
> 2*pi/360

[1] 0.0174533

> 3 / 2^2 + 2 * pi

[1] 7.03319

> ((3 / 2)^2 + 2) * pi

[1] 13.3518
```

Assignment

Results of expressions can be saved as **objects** in the workspace.

There are two (equivalent) **assignment** operators:

```
> rad.deg <- 2*pi/360
> rad.deg = 2*pi/360
```

By default nothing is printed; but all of these:

```
> (rad.deg <- 2*pi/360)
> rad.deg
> print(rad.deg)
```

give the same output:

```
[1] 0.0174533
```

Workspace objects

- Create by assignment
- May be complex **data structures** (see 'methods')
  - In the example below we use the `c` 'catenate; build a chain' function to build a vector
- List with `ls` 'list' or `objects` functions
- Delete with the `rm` (remove) function

```
> (heights <- c(12.2, 13.1, 11.9, 15.5, 10.9))
[1] 12.2 13.1 11.9 15.5 10.9
> ls()
[1] "heights"
> rm(heights); ls()
character(0)
```

Functions and Methods

Most work in S is done with **functions** or **methods**:

1. Method or function **name**; any arguments between parentheses **( )**

2. **Argument list**

   (a) Required
   (b) Optional, with defaults
   (c) **positional** and/or **named**

These usually **return** some values, which can be **complex data structures**

Example of a function call

Function name: `rnorm` (sample from a normal distribution)
Required argument: `n`: number of sampling units
Optional arguments: `mean`, `sd`

```
> rnorm(20)

 [1]  0.388120  0.051022 -1.090701  0.155238  1.725087  2.011053 -2.122989 -0.685271
 [9] -0.112195  0.876962  0.053067 -1.099789  0.299773  0.147167 -0.808183 -0.403877
[17]  1.173150 -1.557166  0.257684 -0.061434

> rnorm(20, mean=180)

 [1] 180.99 180.89 180.64 181.64 179.45 179.90 179.04 179.62 178.94 180.66 179.35
[12] 180.16 179.31 179.66 178.05 180.07 181.58 179.37 179.08 180.21

> rnorm(20, mean=180, sd=10)

 [1] 171.90 179.90 189.82 191.80 182.41 187.19 162.89 202.09 185.78 188.01 174.15
[12] 183.09 158.83 175.42 166.60 188.93 181.84 177.15 167.56 177.75
```

The R help system, R manuals, on-line R help

1. R help

2. R manuals

3. on-line R help

Help on functions or methods

Each function or method is documented with a help page, accessed by the `help` function:

```
> help(rnorm)
```

or, for short:

```
> ?rnorm
```

In R Studio can also search in the `Help` tab.

Output from the help function

- Title and package where found
- Description
- Usage (how to call)
- Arguments (what each one means, defaults)
- Details of the algorithm
- Value returned
- Source of code
- References to the statistical or numerical methods
- See Also (related commands)
- Examples of use and output

Example help page (1/2)

R Help

< > Print Help Search

Normal {stats} R Documentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Arguments

`x,q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.
`mean`	vector of means.
`sd`	vector of standard deviations.
`log, log.p`	logical; if TRUE, probabilities p are given as log(p).
`lower.tail`	logical; if TRUE (default), probabilities are *P[X <= x]*, otherwise, *P[X > x]*.

Details

If `mean` or `sd` are not specified they assume the default values of `0` and `1`, respectively.

The normal distribution has density

f(x) = 1/(sqrt(2 pi) sigma) e^-((x - mu)^2/(2 sigma^2))

where *mu* is the mean of the distribution and *sigma* the standard deviation.

`qnorm` is based on Wichura's algorithm AS 241 which provides precise results up to about 16 digits.

Example help page (2/2)

R Help

< > Print Help Search

Value

`dnorm` gives the density, `pnorm` gives the distribution function, `qnorm` gives the quantile function, and `rnorm` generates random deviates.

Source

For `pnorm`, based on

Cody, W. D. (1993) Algorithm 715: SPECFUN – A portable FORTRAN package of special function routines and test drivers. *ACM Transactions on Mathematical Software* **19**, 22–32.

For `qnorm`, the code is a C translation of

Wichura, M. J. (1988) Algorithm AS 241: The Percentage Points of the Normal Distribution. *Applied Statistics*, **37**, 477–484.

For `rnorm`, see RNG for how to select the algorithm and for references to the supplied methods.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 1, chapter 13. Wiley, New York.

See Also

`runif` and `.Random.seed` about random number generation, and `dlnorm` for the *Log*normal distribution.

Examples

```
require(graphics)

dnorm(0) == 1/ sqrt(2*pi)
dnorm(1) == exp(-1/2)/ sqrt(2*pi)
dnorm(1) == 1/ sqrt(2*pi*exp(1))

## Using "log = TRUE" for an extended range :
par(mfrow=c(2,1))
plot(function(x) dnorm(x, log=TRUE), -60, 50,
     main = "log { Normal density }")
curve(log(dnorm(x)), add=TRUE, col="red",lwd=2)
mtext("dnorm(x, log=TRUE)", adj=0)
```

R manuals

- Included in the R distribution
- Access in R Studio with the `Help` tab or `Help | R help` menu item
- Six manuals; the first two are most relevant to an end-user
  - **An Introduction to R** – somewhat difficult reading but packed with information
  - **R Data Import/Export**
  - R Installation and Administration
  - The R Language Definition
  - Writing R Extensions
  - R Internals
- Reference cards ("cheatsheets")
- FAQ

on-line R help

- R task views
- StackOverflow R tags
- RSeek: `http://www.rseek.org/`
- User-written manuals, reference cards etc.: `http://cran.r-project.org/`, link "Contributed"

StackOverflow

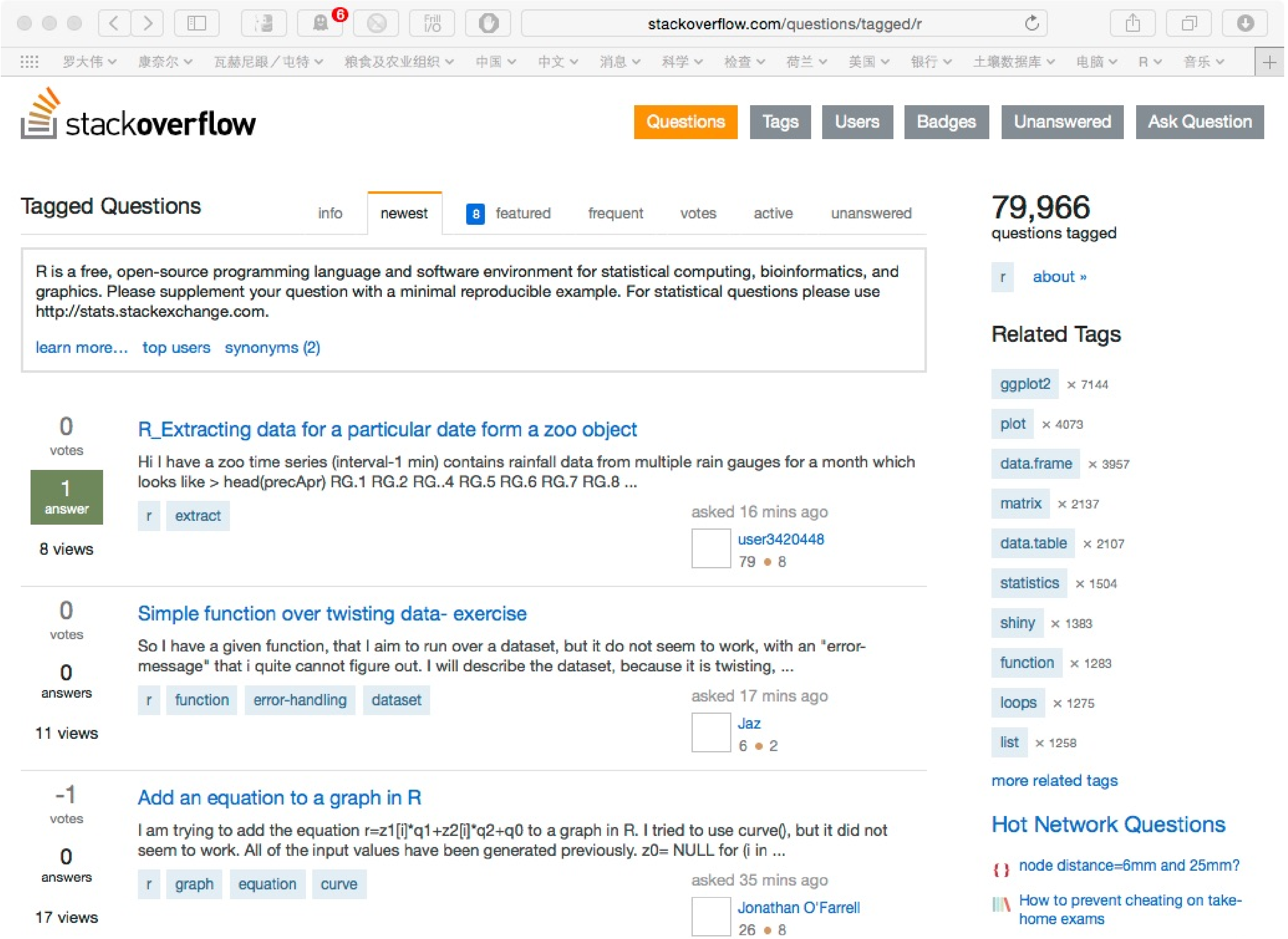
URL: `http://stackoverflow.com/questions/tagged/r`: "Stack Overflow is a question and answer site for professional and enthusiast programmers."

Q&A tagged; the "R" tag is used for R questions.

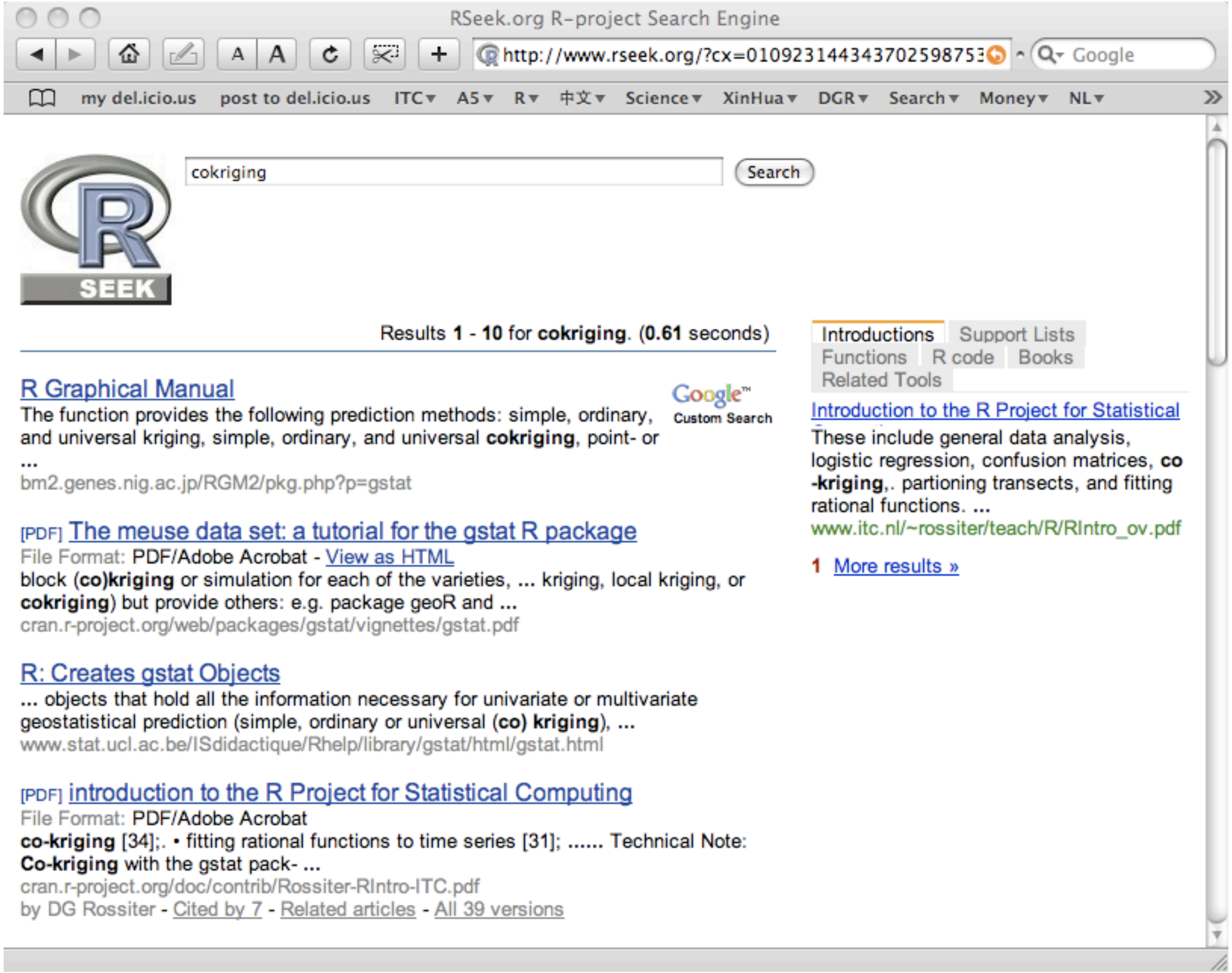
For statistics questions, see `http://stats.stackexchange.com`: "Cross Validated is a question and answer site for people interested in statistics, machine learning, data analysis, data mining, and data visualization."

You can post questions, always with small, reproducible examples – often writing those examples will give you the solution yourself!

StackOverflow R tags



RSeek results



R Task Views

Some applications are covered in **Task Views**, on-line at `http://cran.r-project.org/web/views/index.html`.

These are a summary by a task maintainer of the facilities in R (e.g., which packages and functions to use) to accomplish certain tasks.

Examples:

- **Analysis of Spatial Data**
  `http://cran.r-project.org/web/views/Spatial.html`
- **Handling and Analyzing Spatio-Temporal Data**`https://cran.r-project.org/web/views/SpatioTemporal.html`
- **Multivariate Statistics**
  `http://cran.r-project.org/web/views/Multivariate.html`
- **Analysis of Ecological and Environmental Data**
  `http://cran.r-project.org/web/views/Environmetrics.html`

Contributed packages and example datasets

A major strength of R is the availability of **user-contributed packages**; 18 498 as of 27-Nov-2021!

You don't need all of them!

These are often described in **journal articles**, **books** or **technical reports**, e.g.,

Baddeley, A., & Turner, R. (2004). **spatstat**: An R Package for Analyzing Spatial Point Patterns. Journal of Statistical Software, 12(6). Retrieved from `http://www.jstatsoft.org/v12/i06`

Fox, J. (2002). An R and S-PLUS Companion to Applied Regression. Newbury Park: Sage. (the **car** package)

Diggle, P. J., & Ribeiro Jr., P. J. (2007). Model-based geostatistics. Springer. (the **geoR** package)

Installing packages

1. Find list at `http://cran.r-project.org/`; link "Packages", link "Table of available packages, sorted by name"

2. In RStudio: "Packages" pane, "Install" button; enter the names of the packages to install

3. Also check "Install dependecies" – most packages depend on others to also be on the system

4. The first time you will be prompted to pick a **repository**, also known as **mirror** – R is hosted at 100's of sites around the world; they should all have the same packages

Loading packages

The `library` and `require` functions (almost equivalent) load a package if it's not already in the workspace; they will also load dependencies (assuming these are installed on your system):

```
> library(gstat)    # `require(gstat)` is equivalent

Loading required package: gstat
Loading required package: sp
```

Using RStudio: "Packages" pane, check the packages to load.

How do I find the method to do what I want?

- Mentioned in **journal articles** on subject of interest.
- Look at the **help pages for methods you do know**; they often list related methods.
- **Search for keywords**
  - e.g., `help.search("sequence")` lists methods to generate sequences, vectors of sequences, and sequences of dates for time-series analysis.
- Look at the **Task Views** `http://cran.r-project.org/src/contrib/Views/`
- Search the **contributed documentation** at CRAN
- Find a **textbook** that uses R

Example data

- Base R includes a `datasets` package with many **example datasets**
- Most packages also include example data, which are used to explain the packages' functions and methods
- When a package is loaded, so is its example data
- List datasets with `data()`; this is shown in a file frame
- Once you know the dataset name, see its documentation with `?` or `help`
- To load into the workspace, use the `data` function with the dataset name

Example

```
> data()
> ?CO2
> data(CO2)
> library(sp)
> data(package="sp")
> ?meuse
> data(meuse)
```

CO2 is a dataset in the datasets package

```
Data sets in package 'datasets':

AirPassengers             Monthly Airline Passenger Numbers 1949-1960
BJsales                   Sales Data with Leading Indicator
BJsales.lead (BJsales)    Sales Data with Leading Indicator
BOD                       Biochemical Oxygen Demand
CO2                       Carbon Dioxide Uptake in Grass Plants
ChickWeight               Weight versus age of chicks on different diets
DNase                     Elisa assay of DNase
EuStockMarkets            Daily Closing Prices of Major European Stock Indices, 1991-1998
Formaldehyde              Determination of Formaldehyde
HairEyeColor              Hair and Eye Color of Statistics Students
```

meuse is a dataset in the sp "classes and methods for spatial data" package

```
Data sets in package 'sp':

Rlogo                     Rlogo jpeg image
gt (Rlogo)                Rlogo jpeg image
meuse                     Meuse river data set
meuse.grid                Prediction Grid for Meuse Data Set
meuse.grid_ll             Prediction Grid for Meuse Data Set, geographical coordinates
meuse.riv                 River Meuse outline
```

Topics – Part 2

1. Data types: logical, numeric (integer, double, complex), character, lists
2. Arrays, matrices and dataframes
3. Vectorized operations; applying functions over arrays
4. Matrix and dataframe manipulation
5. Logical expressions
6. Importing and exporting data
7. Summarizing data
8. Basic statistical functions
9. Specifying statistical models; the `lm` (linear models) function

Data types and structures

- Data types: logical, numeric (integer, double, complex), character, lists
- Arrays, matrices and dataframes
- Vectorized operations; applying functions over arrays
- Matrix and dataframe manipulation
- Factors (categorical variables)

Basic data types

- All objects in S have a **data type**

- Operators and functions understand these

- Some basic types: `logical`; `integer`; `double`; `character`; `list`; `expression`; `function`

- `logical`; `integer`; `double`; `character` are all **vectors** with one or more **elements**

- **lists** can combine any objects

Derived data types

These are basic types with some additional **attributes** appropriate to the derived type.

Examples:

- a `array` is a vector with a `dim` "dimensions" attribute
- a `matrix` is a 2-D `array`
- a `dataframe` is a `matrix` with column (field) `names` and `row.names`

Vectorized operations

S works on vectors and matrices as with scalars, with natural extensions of operators, functions and methods.

```
> (sample <- seq(1, 10) + rnorm(10))

 [1] -0.1878978 1.6700122 2.2756831 4.1454326
 [5] 5.8902614 7.1992164 9.1854318 7.5154372
 [9] 8.7372579 8.7256403
```

The ten integers 1 ...10 returned by the call to the `seq` (sequence) method each have a different random noise added to them; here the `rnorm` method also returns ten values.

If one of the vectors is shorter than the other, it is **recycled** as necessary:

```
> (samp <- seq(1, 10) + rnorm(5))

 [1] -1.23919739 0.03765046 2.24047546 4.89287818
 [5] 4.59977712 3.76080261 5.03765046 7.24047546
 [9] 9.89287818 9.59977712
```

Objects and classes

- S is an **object-oriented** computer language
- Everything in S is an **object**; every object has a **class**
- The class determines the way in which it may be manipulated
- Generic methods (e.g., `summary`, `str`) dispatch by the class

```
> class(seq(1:10)); class(seq(1,10, by=.01)); class(letters)

[1] "integer"
[1] "numeric"
[1] "character"

> class(diag(10)); class(iris); class(lm)

[1] "matrix"
[1] "data.frame"
[1] "function"
```

Examples

```
> letters; letters + 3

 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20]  "t" "u" "v" "w" "x" "y" "z"

Error in letters + 3 : non-numeric argument to binary operator

> str(letters); str(diag(10)); str(iris)

 chr [1:26] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" ...

 num [1:10, 1:10] 1 0 0 0 0 0 0 0 0 0 ...

'data.frame':        150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Matrices

```
> (cm <- c(35,14,11,1,4,11,3,0,12,9,38,4,2,5,12,2))

 [1] 35 14 11  1  4 11  3  0 12  9 38  4  2  5 12 2

>   dim(cm)

    NULL
```

Initially, the vector has no dimensions; these are added with the dim function:

```
> dim(cm) <- c(4, 4)
> print(cm)

     [,1] [,2] [,3] [,4]
[1,]   35    4   12    2
[2,]   14   11    9    5
[3,]   11    3   38   12
[4,]    1    0    4    2

> dim(cm)

 [1] 4 4
```

Matrix arithmetic

Many S operators can work directly on matrices; there are also typical matrix functions:

- +, −, *, / etc. work element-wise
- matrix multiplication: `%*%`
- transposition: `t` function
- inversion: `solve` function
- spectral decomposition: `eigen` function
- Singular Value Decomposition: `svd` function
- ...

Data frames

The fundamental structure for statistical analysis; a **matrix** with:

1. **named columns** (roughly, database "fields") and

2. (optionally) **named rows** (roughly, database "cases"):

We illustrate with one of R's **example datasets**, provided in the base `datasets` package:

We first display the help file, then load the data, then view the data structure (field names and types):

```
> ?trees
> data(trees)
> str(trees)

`data.frame': 31 obs. of 3 variables:
$ Girth : num 8.3 8.6 8.8 10.5 10.7 10.8 11 ...
$ Height: num 70 65 63 72 81 83 66 75 80 75 ...
$ Volume: num 10.3 10.3 10.2 16.4 18.8 19.7 ...
```

Accessing fields in a data frame

Using the $ operator:

```
> summary(trees$Volume)

Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
10.2    19.4    24.2    30.2    37.3    77.0
```

This **$** operator **exposes** the field name.

Accessing a dataframe with matrix operators

The dataframe is just a special matrix, so:

```
> trees[1,]    # one case, i.e. the first tree

Girth Height Volume
1 8.3 70 10.3

> trees[,2]   # all cases (trees), second field (heights)

[1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69
[15] 75 74 85 86 71 64 78 80 74 72 77 81 82 80
[ 29] 80 80 87

> trees[1,2]   # one field of one case: height of first tree

[1] 70
```

```
> trees[1:3,]   # first three cases (trees), all fields

  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2

> head(trees[,c(1,3)]) # first and third fields; `head' shows first six

  Girth Volume
1   8.3   10.3
2   8.6   10.3
3   8.8   10.2
4  10.5   16.4
5  10.7   18.8
6  10.8   19.7

> trees[1,"Height"] # named field (i.e., matrix column)

[1] 70
```

Factors

- Variables with a limited number of discrete values (categories) are called S **factors**.

- Internally they are stored as integers but each has a text name.

- They are handled properly by R functions and methods (they are *not* integers!).
  - **Unordered** factors: no intrinsic order
  - **Ordered** factors: intrinsic order relation, $>$ etc. make sense

Example of factors (1/2)

Suppose we have given three tests to each of three students, each with a numeric ID, and we want to compare the students. We might enter the data frame as follows:

```
>  student <- rep(c(700123, 131444, 201113), 3)
> score <- c(9, 6.5, 8, 8, 7.5, 6, 9.5, 8, 7)
>  tests <- data.frame(cbind(student, score))
> str(tests)

'data.frame':        9 obs. of  2 variables:
 $ student: num  700123 131444 201113 700123 131444 ...
 $ score  : num  9 6.5 8 8 7.5 6 9.5 8 7
```

The data type of `student` is `numeric` – this can lead to absurdities:

```
> lm(score ~ student, data=tests)

Coefficients:
(Intercept)      student
  6.682e+00    3.022e-06
```

Meaningless!

Example (2/2)

Convert to a **factor**: the student number is just an ID; use `as.factor`:

```
> tests$student <- as.factor(tests$student)
> levels(tests$student)

[1] "131444" "201113" "700123"

> str(tests)

'data.frame':        9 obs. of  2 variables:
 $ student: Factor w/ 3 levels "131444","201113",..: 3 1 2 3 1 2 3 1 2
 $ score  : num  9 6.5 8 8 7.5 6 9.5 8 7

> lm(score ~ student, data=tests)

  (Intercept)  student201113  student700123
       7.3333        -0.3333         1.5000
```

This is a **meaningful one-way linear model**, showing the difference in mean scores of students 201113 and 700123 from student 131444 (the intercept).

Data manipulation

One of the strengths of R is the ability to manipulate data.

This is especially useful for automatic identification of suspected errors, outlier detection, data transformations, subsetting on a factor . . .

Subsetting on a logical expression

Find the tallest trees using the `subset` function:

```
> sort(trees$Height)
> subset(trees, Height >= 80)
```

```
 [1] 63 64 65 66 69 70 71 72 72 74 74 75 75 75 76 76 77 78 79 80 80 80 80 80 81 81
[27] 82 83 85 86 87
```

```
   Girth Height Volume
5   10.7     81   18.8
6   10.8     83   19.7
9   11.1     80   22.6
17  12.9     85   33.8
18  13.3     86   27.4
22  14.2     80   31.7
26  17.3     81   55.4
27  17.5     82   55.7
28  17.9     80   58.3
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
```

Another way . . .

Can use logical expression as subscripts:

```
> (trees.tall <- trees[trees$Height >= 80 , ])

   Girth Height Volume
5   10.7     81   18.8
6   10.8     83   19.7
9   11.1     80   22.6
17  12.9     85   33.8
18  13.3     86   27.4
22  14.2     80   31.7
26  17.3     81   55.4
27  17.5     82   55.7
28  17.9     80   58.3
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
```

Identifying with a logical expression

Which are the tallest trees? Save indices for later use.

Use the `which` function to find the indices:

```
> (trees.tall.ix <- which(trees$Height >= 80))
> trees[trees.tall.ix, ]

 [1]  5  6  9 17 18 22 26 27 28 29 30 31

   Girth Height Volume
5   10.7     81   18.8
6   10.8     83   19.7
9   11.1     80   22.6
17  12.9     85   33.8
18  13.3     86   27.4
22  14.2     80   31.7
26  17.3     81   55.4
27  17.5     82   55.7
28  17.9     80   58.3
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
```

More complicated logical expression

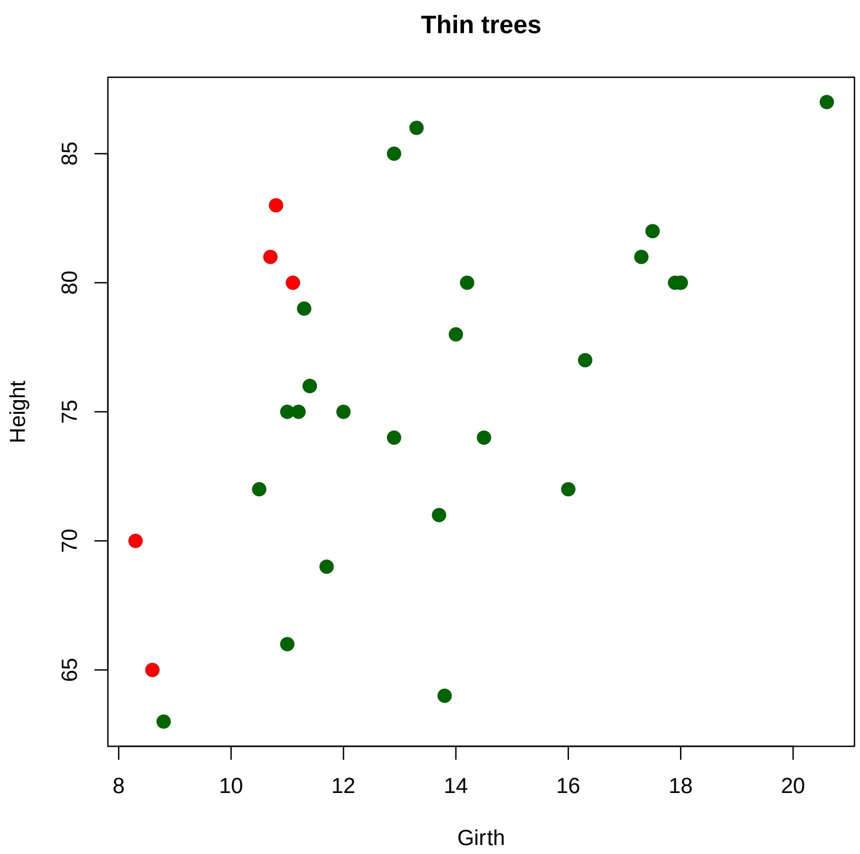
Find very thin trees:

```
> trees$hg <- trees$Height/trees$Girth
> sort(trees$hg)
 [1] 4.223301 4.444444 4.444444 4.469274 4.500000 4.637681 4.682081 4.685714
 [9] 4.723926 5.103448 5.182482 5.571429 5.633803 5.736434 5.897436 6.000000
[17] 6.250000 6.466165 6.589147 6.666667 6.666667 6.696429 6.818182 6.857143
[25] 6.991150 7.159091 7.207207 7.558140 7.570093 7.685185 8.433735
> summary(trees$hg)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.223   4.705   6.000   5.986   6.838   8.434
> sd(trees$hg)
> (trees.thin <- subset(trees, hg > (mean(trees$hg) + sd(trees$hg))))

  Girth Height Volume       hg
1   8.3     70   10.3 8.433735
2   8.6     65   10.3 7.558140
5  10.7     81   18.8 7.570093
6  10.8     83   19.7 7.685185
9  11.1     80   22.6 7.207207
```

Visualizing the thin trees

```
> plot(trees$Height ~ trees$Girth, xlab="Girth", ylab="Height",
  main="Thin trees", pch=20, cex=2,
  col=ifelse(trees$hg >  (mean(trees$hg) + sd(trees$hg)), "red", "darkgreen"))
```



Import/Export

Reference: "R Data Import/Export", R manual installed with R; available under Help menu

Most common interchange format for flat-files: **Comma-Separated Values** (`CSV`)

CSV file import

Look at file in plain-text editor; note header line, and the , field separator

```
x,y,cadmium,elev,dist,om,ffreq,soil,lime,landuse
181072,333611,11.7,7.909,0.00135803,13.6,1,1,1,Ah
181025,333558,8.6,6.983,0.0122243,14,1,1,1,Ah
181165,333537,6.5,7.8,0.103029,13,1,1,1,Ah
181298,333484,2.6,7.655,0.190094,8,1,2,0,Ga
181307,333330,2.8,7.48,0.27709,8.7,1,2,0,Ah
181390,333260,3,7.791,0.364067,7.8,1,2,0,Ga
```

Import with `read.csv`:

```
> ds <- read.csv("test.csv")
> str(ds)
```

(results on next slide)

Result of CSV file input

```
'data.frame':       6 obs. of  10 variables:
 $ x       : int  181072 181025 181165 181298 181307 181390
 $ y       : int  333611 333558 333537 333484 333330 333260
 $ cadmium: num  11.7 8.6 6.5 2.6 2.8 3
 $ elev    : num  7.91 6.98 7.8 7.66 7.48 ...
 $ dist    : num  0.00136 0.01222 0.10303 0.19009 0.27709 ...
 $ om      : num  13.6 14 13 8 8.7 7.8
 $ ffreq   : int  1 1 1 1 1 1
 $ soil    : int  1 1 1 2 2 2
 $ lime    : int  1 1 1 0 0 0
 $ landuse: Factor w/ 3 levels "Ah","Ga","Ga    ": 1 1 1 2 1 3
```

Note that `read.csv` could determine field `landuse` is a **factor**.

But it was not able to do so for the factors `ffreq`, `soil`, `lime`. So, we have to convert:

```
> ds$ffreq <- as.factor(ds$ffreq)
```

General file import

Very flexible `read.table` function:

- field delimeters
- integer / decimal separator
- Header line(s)
- Skip lines
- Specify data types

File export

Very flexible `write.table` function.

```
> write.table(round(as.data.frame(kxy), 4), file="KrigeResult.csv",
    sep=",", quote=T, row.names=F,
    col.names=c("E", "N", "LPb", "LPb.var"))
```

There are also ways to export to spreadsheets, databases, images, GIS coverages . . .

Statistical models in S

- Specified in **symbolic form** with model **formulae**
- These formulae are **arguments** to many statistical methods:
  - `lm` (linear models)
  - `glm` (generalised linear models)
  - `gstat` methods such as `variogram` and `krige`
- Can also be used in other contexts:
  - Base graphics methods such as `plot` and `boxplot`
  - Trellis graphics methods such as `levelplot`

Form of statistical models

- **Left-hand** side: (mathematically) **dependent** variable
- Formula **operator** ~
- **Right-hand** side: (mathematically) **independent** variable(s)

The simplest use is in **simple linear regression**:

```
> model <- lm(Volume ~ Height, data=trees); summary(model)
> # equivalent to: model <- lm(trees$Volume ~ trees$Height)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -87.124     29.273   -2.98  0.00583
Height         1.543      0.384    4.02  0.00038
```

So, the tree volume is modelled as a linear function of the tree height.

Model formula operators

- **Additive** effects: +

```
> model <- lm(Volume ~ Height + Girth, data=trees); summary(model)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -57.988      8.638   -6.71  2.7e-07
Height         0.339      0.130    2.61    0.014
Girth          4.708      0.264   17.82  < 2e-16
```

- **Interactions**: *

```
> model <- lm(Volume ~ Height * Girth, data=trees); summary(model)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   69.3963    23.8358    2.91  0.00713
Height        -1.2971     0.3098   -4.19  0.00027
Girth         -5.8558     1.9213   -3.05  0.00511
Height:Girth   0.1347     0.0244    5.52  7.5e-06
```

- **Crossing factors** to a specified degree

```
> model <- lm(Volume ~ (Height + Girth)^2, data=trees)
```

In this case it's the same as `Height * Girth`, because there are only two factors.

- **Nested models**: /

```
> model <- lm(Volume ~ Height / Girth, data=trees)

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -0.23114    7.74157   -0.03   0.9764
Height        -0.41218    0.12316   -3.35   0.0023
Height:Girth   0.06070    0.00266   22.79   <2e-16
```

- **Remove terms**: −; for example, the intercept:

```
> model <- lm(Volume ~ Height -1, data=trees); summary(model)

Coefficients:
       Estimate Std. Error t value Pr(>|t|)
Height   0.4047     0.0354    11.4  1.9e-12
```

- **Arithmetic in models**: `I()` method if ambiguous

```
> model <- lm(Volume ~ I(Height^2), data=trees); summary(model)
>                 # otherwise would cross height with itself in the model

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -30.19193   15.02843   -2.01  0.05393
I(Height^2)   0.01038    0.00255    4.07  0.00033
```

Updating models

Use the update function, previous LHS and RHS represented by .

```
> model <- lm(Volume ~ Height + Girth, data=trees); summary(model)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -57.988      8.638   -6.71  2.7e-07
Height         0.339      0.130    2.61    0.014
Girth          4.708      0.264   17.82  < 2e-16

> model.2 <- update(model, . ~ . - Girth); summary(model.2)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -87.124     29.273   -2.98  0.00583
Height         1.543      0.384    4.02  0.00038
```

Model objects: structure and access

- Modelling functions like `lm` return an **object** with a **class**
- You can look directly at the structure …
- …but it is preferable to use **access methods** such as `coefficients`, `residuals`, `fitted`.

```
> model <- lm(Volume ~ log(Height), data=trees); class(model); str(model)

[1] "lm"

List of 12
 $ coefficients : Named num [1:2] -461  114
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "log(Height)"
 $ residuals    : Named num [1:31] -10.928  -2.511   0.939  -8.028 -19.005 ...
  ..- attr(*, "names")= chr [1:31] "1" "2" "3" "4" ...
...
```

Using access functions

These extract information from the fitted model.

```
> summary(residuals(model))

     Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-2.09e+01 -9.77e+00 -2.51e+00 -4.73e-16  1.22e+01  3.11e+01
```

Other important access functions: `summary`, `fitted`, `coef`, `anova`, `effects`, `vcov`.

Factors

- For **categorical** variables (can take only a defined set of values)
  - **unordered** (nominal), e.g. land cover class
  - **ordered** (ordinal), e.g. vegetation density class
- S calls these **factors**
- Methods (especially **modelling**) take appropriate action
- These are converted to **contrasts** in the **design matrix** of linear (and other) models

Topics – Part 3

1. R base graphics
2. Scripts
3. User-defined functions
4. Programming in R: control structures
5. The R class structure; object-oriented programming
6. The ggplot2 graphics system
7. Some advanced statistical functions
8. Going further in R: task views, textbooks, tutorials

R Graphics

R has a very rich visualization environment. There are (at least) four graphics systems:

1. Base graphics system: default **graphics** package (always loaded)
2. Trellis graphics: **lattice** package
3. "Grammar of Graphics" **ggplot2** package
4. Grid graphics

R graphics are highly **customizable**; it is usual to write small **scripts** to get the exact output you want.

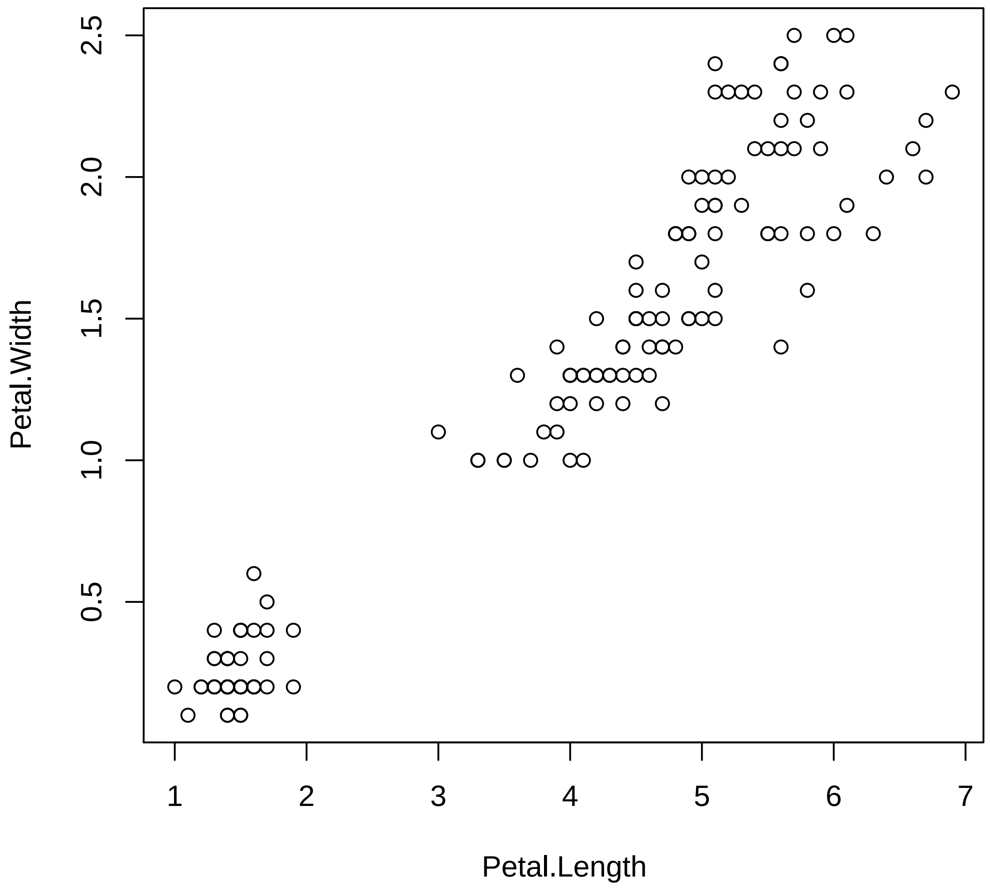
Graphs may be **displayed on screen** or written directly to **files** for inclusion in documents.

Base graphics

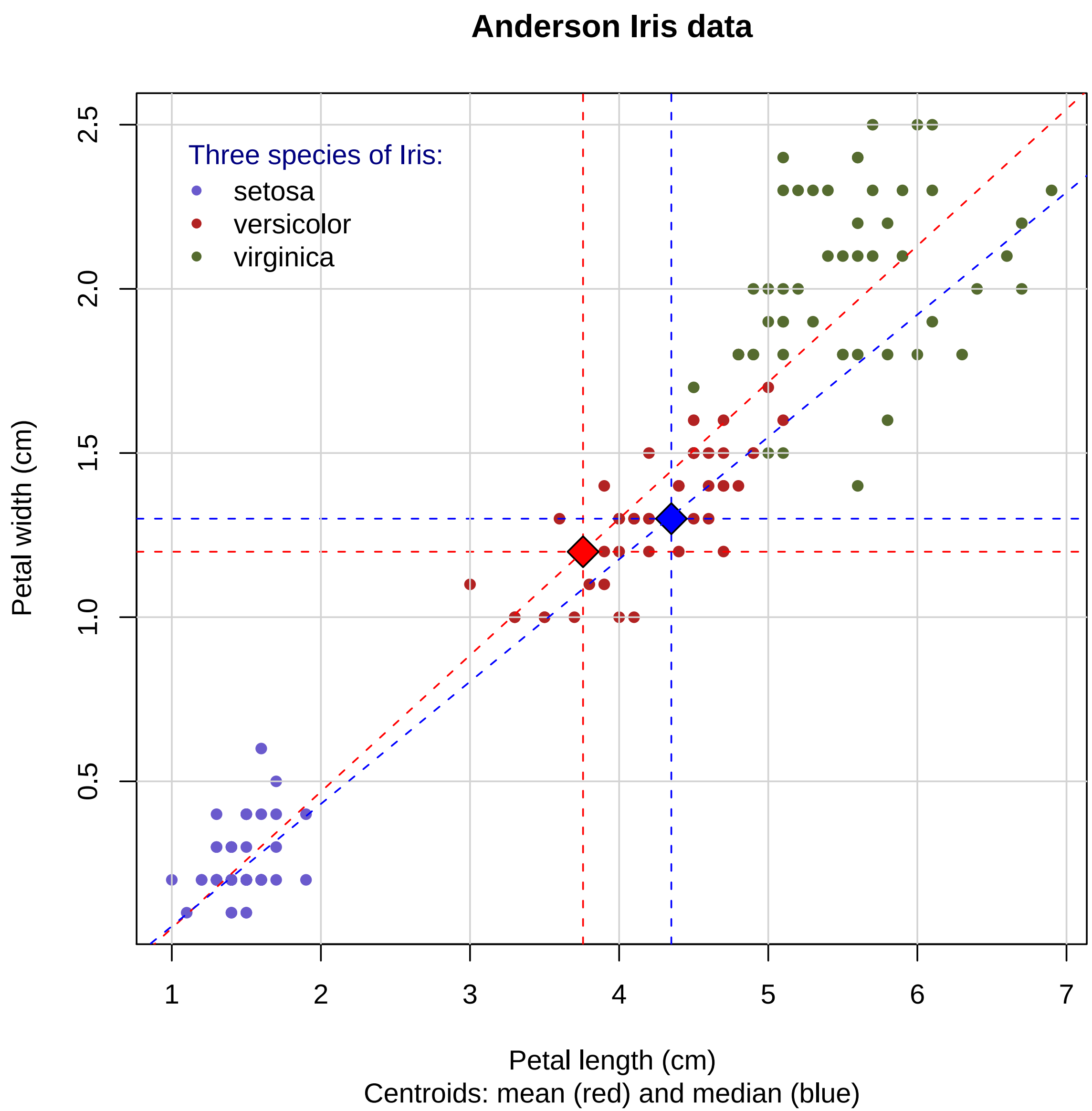
- Simple to learn
- Can make simple plots very easily
- Can also customize at will
- Some methods start a new plot, e.g. `plot`, `hist`, `boxplot`
- Other add to an existing (open) plot, e.g. `points`, `lines`, `rug`

Example of default base graphics

```
> data(iris); with(iris, plot(Petal.Width ~ Petal.Length))
```



Example of customized base graphics



Code for previous graph

```
> attach(iris)
> plot(Petal.Length, Petal.Width, pch=20, cex=1.2,
    xlab="Petal length (cm)", ylab="Petal width (cm)",
    main="Anderson Iris data",
    col=c("slateblue", "firebrick", "darkolivegreen")[as.numeric(Species)]
  )
> abline(v=mean(Petal.Length), lty=2, col="red")
> abline(h=mean(Petal.Width), lty=2, col="red")
> abline(v=median(Petal.Length), lty=2, col="blue")
> abline(h=median(Petal.Width), lty=2, col="blue")
> grid()
> points(mean(Petal.Length), mean(Petal.Width), cex=2, pch=23, col="black", bg="red")
> points(median(Petal.Length), median(Petal.Width), cex=2, pch=23,
            col="black", bg="blue")
> title(sub="Centroids: mean (green) and median (gray)")
> text(1, 2.4, "Three species of Iris", pos=4, col="navyblue")
> legend(1, 2.4, levels(Species), pch=20, bty="n",
             col=c("slateblue","firebrick", "darkolivegreen"))
> detach(iris)
```

Note that `plot` starts a **new** graph; all the others **add** elements to the plot.

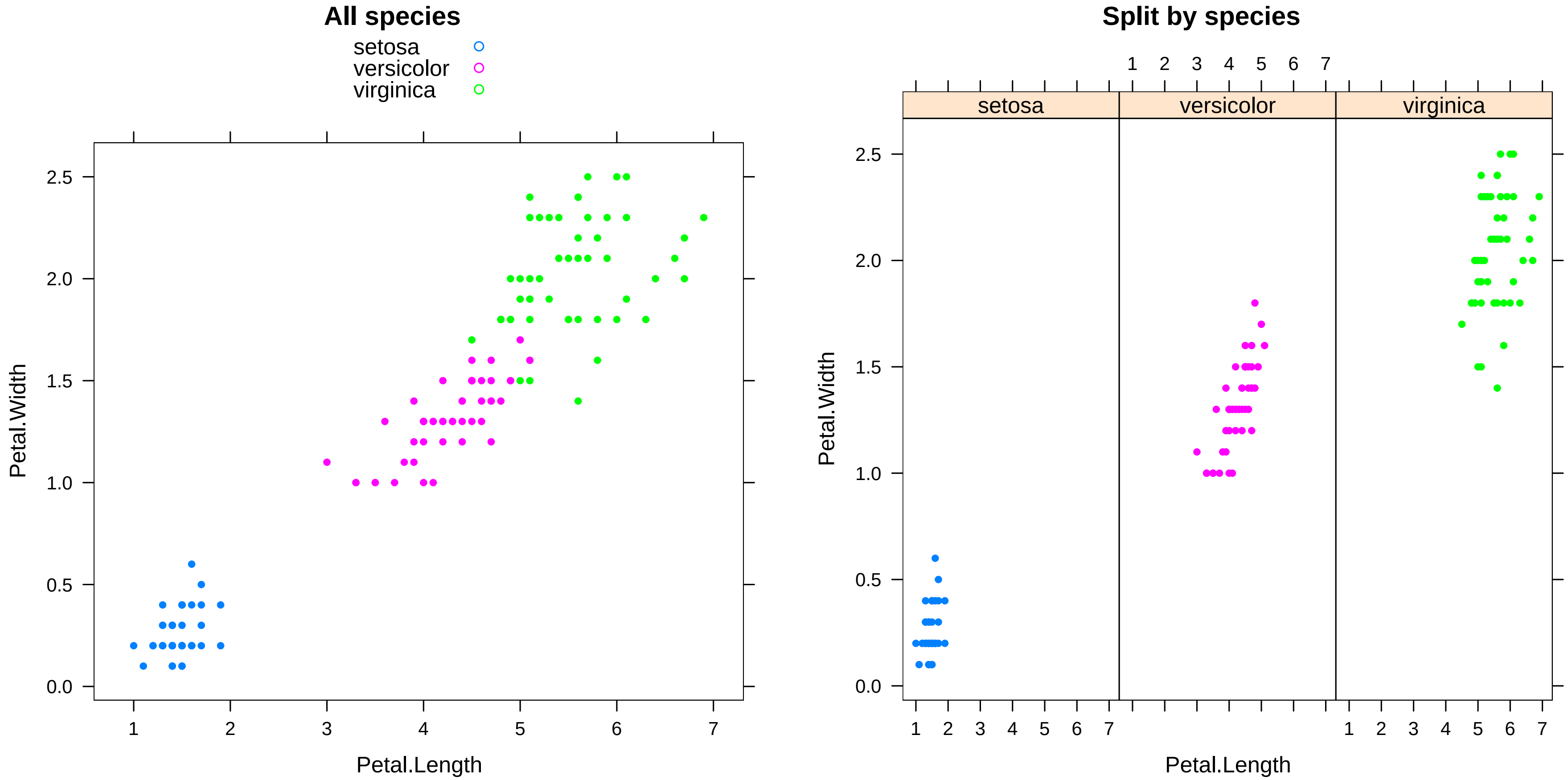
Trellis graphics

An R implementation of the trellis graphics system developed at Bell Labs by Cleveland is provided by packakge `lattice`.

It is especially intended for **multivariate visualization**

- Harder to learn than R base graphics
- Can produce higher-quality graphics, especially for multivariate visualisation when the relationship between variables changes with some grouping factor; this is called **conditioning** the graph on the factor
- It uses **model formulae** similar to the statistical formulae to specify the variables to be plotted and their relation in the plot.
- Multiple items on one plot are specified with user-written **panel functions**

Example of trellis graphics



Note the right plot: it has been **conditioned** on a factor, namely the species.

Cornell University
College of Agriculture and Life Sciences

Code for previous graph

```
> xyplot(Petal.Width ~ Petal.Length, data=iris, groups=Species, auto.key=T)
> xyplot(Petal.Width ~ Petal.Length | Species, data=iris, groups=Species)
```

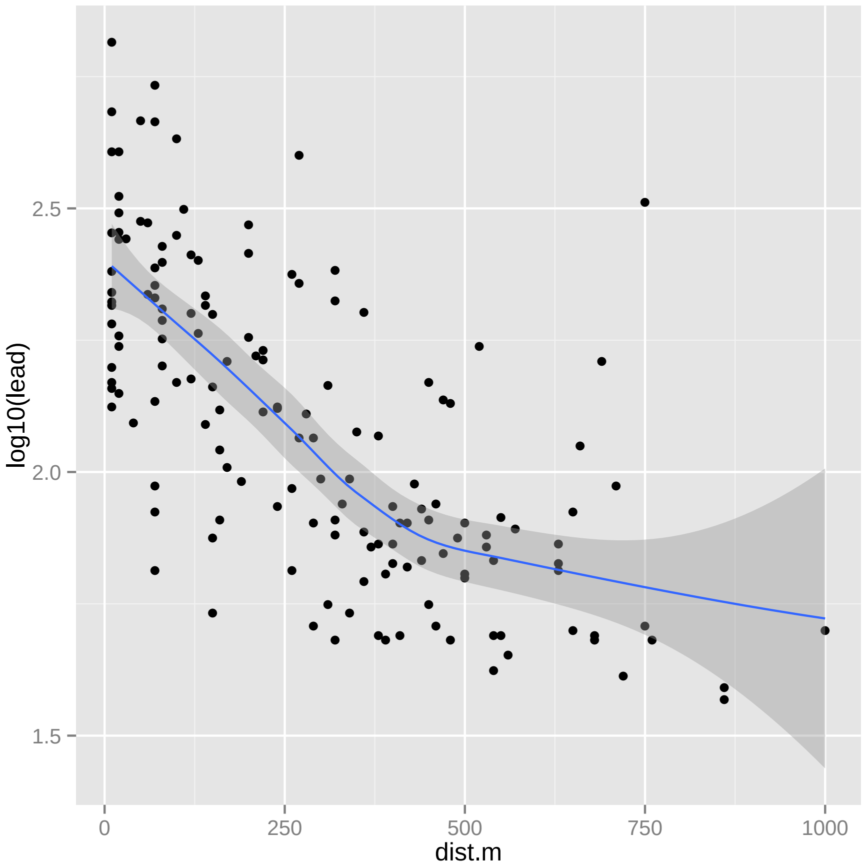
Note the | in the formula; this means "conditioned on".

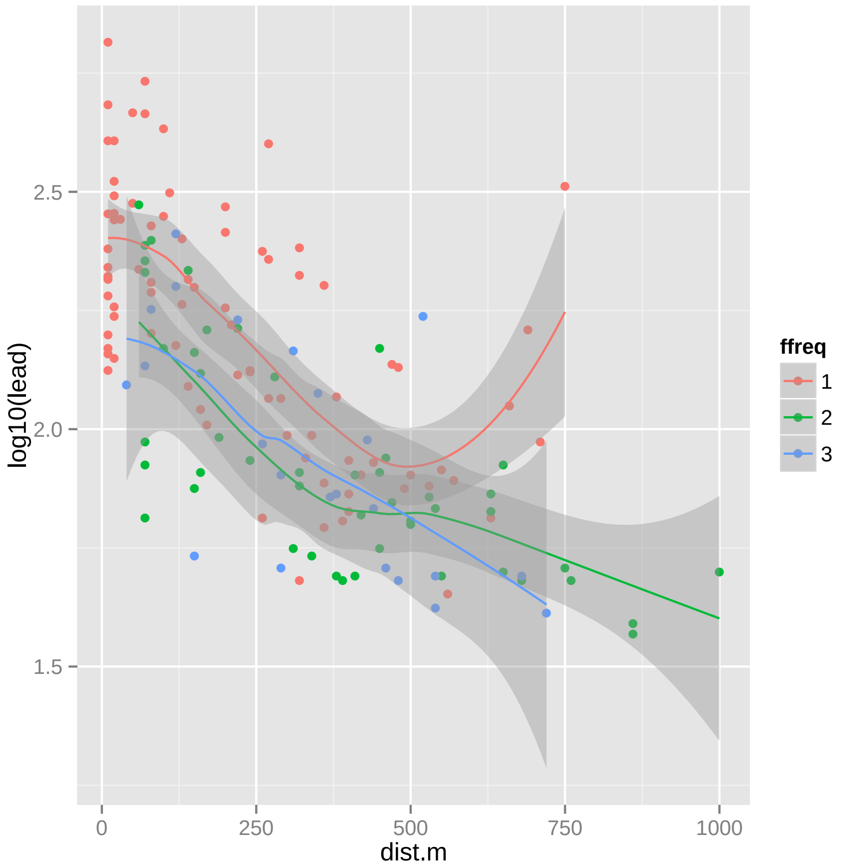
Cornell University
College of Agriculture and Life Sciences

Grammar of graphics

- A completely new way to think about composing **statistical** graphs
- Text: Wickham, H., 2009. ggplot2: Elegant Graphics for Data Analysis, Use R! Springer.
- Web site: `http://ggplot2.org`
- How are statistical graphics a "grammar" ([U+8BED] [U+6CD5])?
  - a **mapping** from **data** to **aesthetic** attributes (colour, shape, size) ...
  - ... of **geometric** objects (points, lines, bars).
  - data may also be statistically-**transformed**
  - the graph must be drawn on a **coordinate system**
  - **subsets** of the data can be shown in sub-windows ("faceting")
- R code to specify can be quite complex
- But the `qplot` "quick plot" method can be used for many simple cases (analogous to `plot` of base graphics).

Example of ggplot2 graphics





Code for previous graph

```
> library(sp); data(meuse)
> qplot(x = dist.m, y = log10(lead), data = meuse,
+     geom = c("point", "smooth", method='loess')
> qplot(x = dist.m, y = log10(lead), data = meuse,
+     colour = ffreq, geom = c("point", "smooth"), method="loess")
```

- Data is the `meuse` dataframe
- two geometries are specified: (1) the points (a scatterplot); (2) a smooth line
- the coordinate system is by default a scatterplot (x-y plot)
- the x and y axes are the two named variables; the Pb content is log-transformed
- in the right-hand graph the points are coloured by a categorical variable (flood frequency class)
- the smooth line and confidence limits are computed by locally-adjusted least squares

Grid graphics

A low-level graphics programming language by Paul Murrel. `lattice` is written in grid. Allows fine control of graphic output.

Complete information on author's R graphics page:

`http://www.stat.auckland.ac.nz/~paul/grid/grid.html`

and in his book:

`http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html`

Programming R

R is a full-featured, modern programming language. This can be accessed four ways, in increasing level of complexity:

S was developed by Chambers for "programming with data"

1. **Commands**: at the > prompt, typed or cut-and-paste
   - These can use **control structures** for looping, conditional execution, and repetition
2. User-written **scripts**
3. User-defined **functions**
4. User-contributed **packages**

Control structures

S has ALGOL-like **control structures**:

- `if ...else`
- `for`; note that **vectorized functions or methods** often are preferable
- `while`, `repeat`
- `break`, `next`

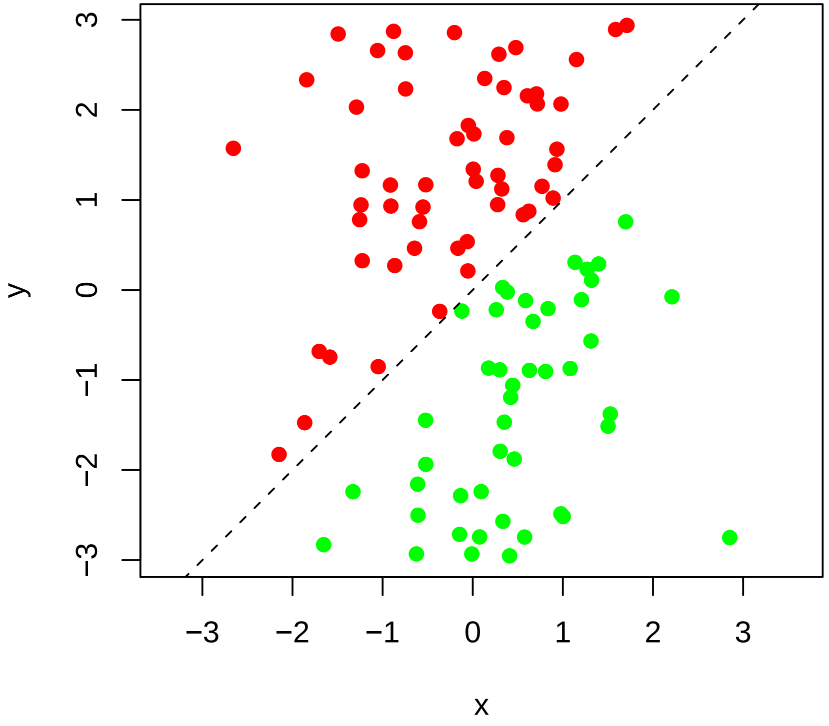
and within an expression:

- the `ifelse` function

Example of the `ifelse` function

Here it is used to select a plotting colour:

```
> x <- rnorm(100); y <- runif(100, -3, 3)
> plot(y ~ x, asp=1, col=ifelse(y > x, "red", "green"), pch=20, cex=1.5)
> abline(0, 1, lty=2)
```



Example of the `while` control structure

For some simulation we want to draw a sample from the normal distribution but make sure there is an extreme value, so we repeat the sampling until we get what we want:

```
> while (max(abs(sample <- rnorm(100))) < 3) print("No extreme")
> range(sample)

[1] "No extreme"
[1] "No extreme"
[1] "No extreme"

[1] -3.2648  2.5457
```

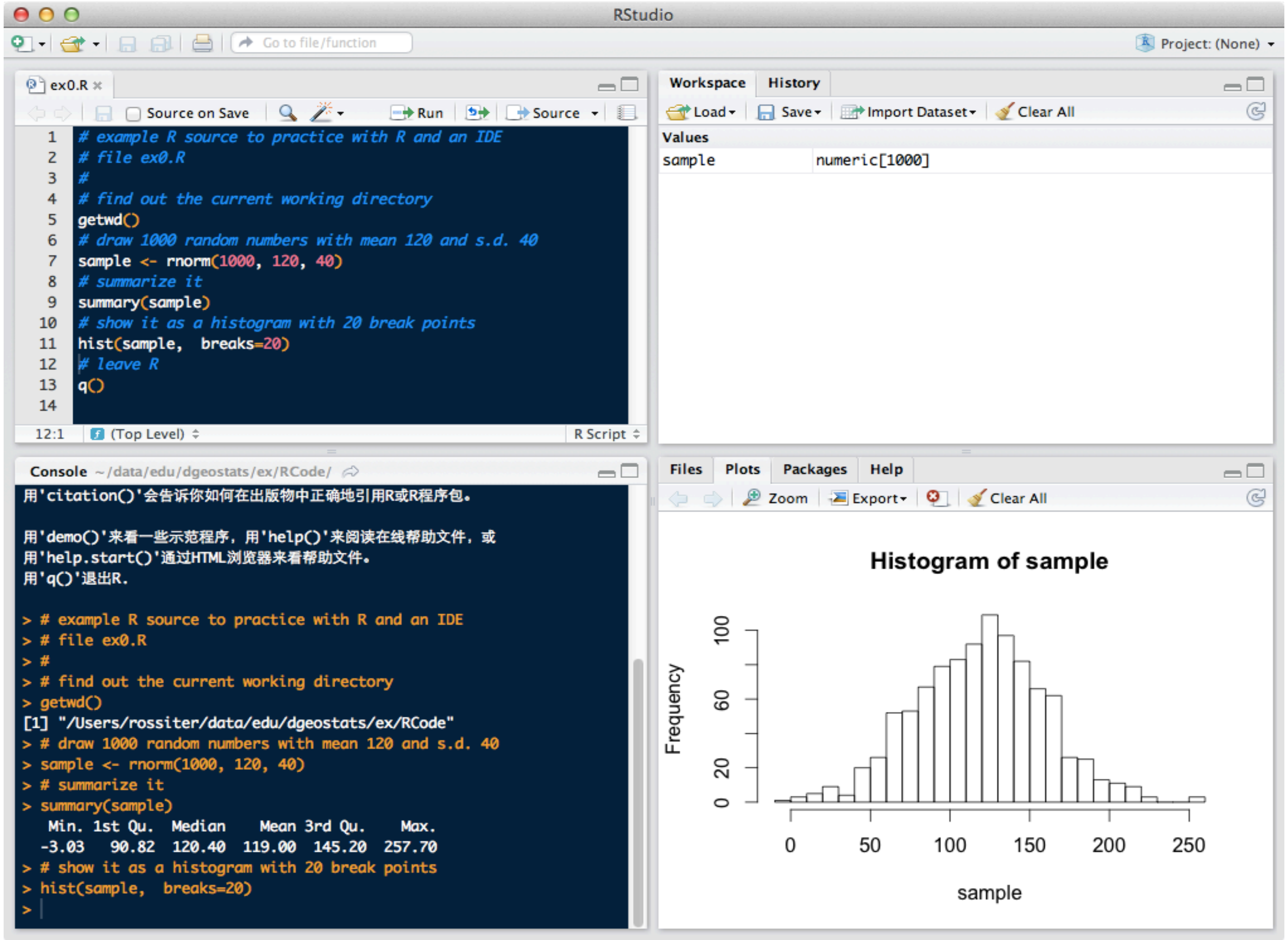
Why use scripts?

- For **reproducible** processing
  - Especially for complicated graphics
  - Also for multi-step analyses
  - For **simulations** where each run is different, due to randomness
- Can **document** the steps internally (as S **comments**)

Writing and running scripts

1. Prepare script in some **editor**
   - Editor component of IDE, e.g., **RStudio**
   - Plain-text editor (no formatting!)
   - Editor built into R: some help with syntax, commands
   - Emacs + ESS ("Emacs speaks statistics") `http://ess.r-project.org/`
2. Run with the `source` function or via editor commands
   - e.g., RStudio "Run Lines" or "Run Region" menu commands

RStudio screenshot with script and console



Example

1. Enter the following in a **plain text** file:

```
# draw two independent normally-distributed samples
x <- rnorm(100, 180, 20); y <- rnorm(100, 180, 20)
# scatterplot
plot(x, y)
# correlation: should be 0
cor.test(x, y, conf=0.9)
```

2. **Save** with name e.g. `test.R` (convention: `.R` extension)

3. In R, **source** the file (or send from the editor):

```
> source("test.R")

t = -0.1925, df = 98, p-value = 0.8477
alternative hypothesis: true correlation is not equal to 0
90 percent confidence interval:
 -0.18433  0.14650
sample estimates:
      cor
-0.019446
```

A more complicated example

Enter this in a script file; save as `test.R`.

```
# see how correlation coefficients are distributed in uncorrelated random samples
m <- 1000 # number of runs
n <- 100 # size of random samples
results <- rep(0, m)
for (i in 1:m) {
  x <- rnorm(100); y <- rnorm(100) # default mu=0, sigma=1
  results[i] <- cor(x, y)
}
summary(results)
tmp <- qplot(results, binwidth=0.02)
print(tmp + geom_bar(colour="white", fill="darkgreen", binwidth=0.02) + geom_rug())
```

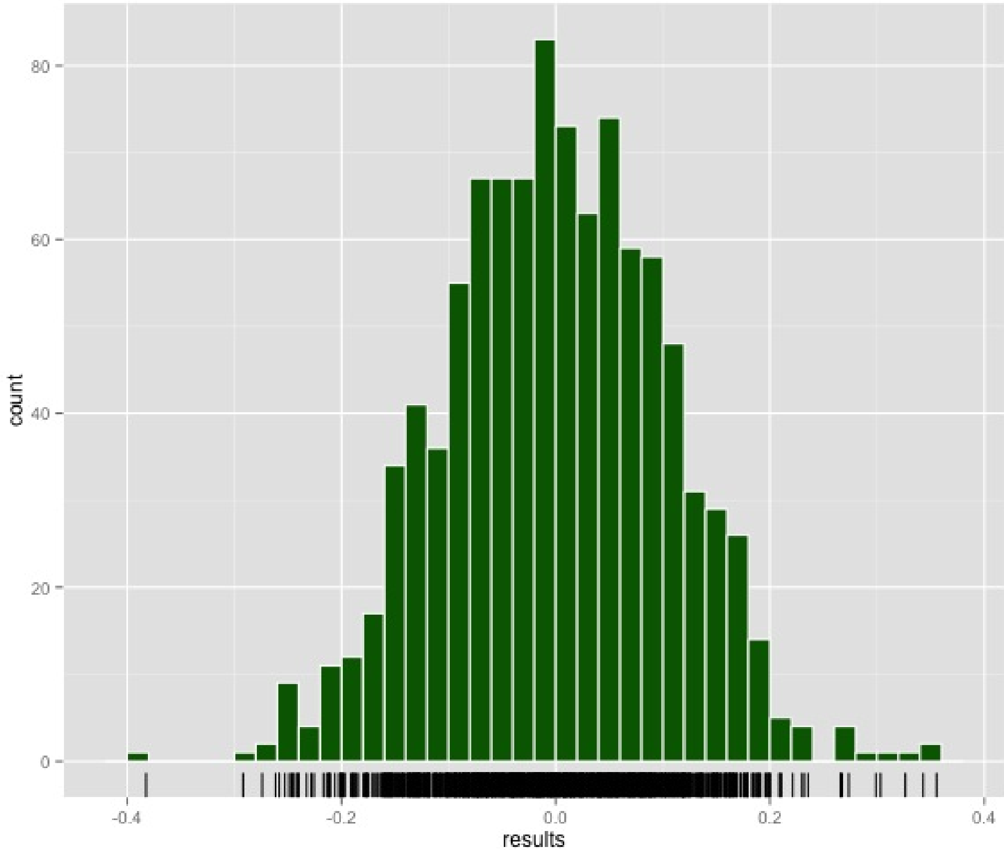
Run the script:

```
> source('test.R')
```

The script can be run several times, also with different numbers of runs and sample sizes, to compare the results.

Results

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.382500 -0.071820 -0.001889 -0.001115  0.072200  0.355600
```



User-defined functions

- These are like R **built-in functions** but simpler
- Defined as objects in the **workspace** (not in the system)
- Why?
  - R may not have a function or method to compute what you want
  - You want to expand a script with **arguments** to apply the script to any suitable object

Simple example of user-defined function

There is no R function to compute the harmonic (geometric) mean of a vector, but we can define it easily enough. For a vector v with n elements:

$$\bar{v}_h = \left[\prod_{i=1 \ldots n} v_i \right]^{1/n}$$

This is computed by taking logarithms, dividing by the length, and exponentiating.

The `function` function is used to define a function (!); it can then be **assigned** to an object in the workspace. The function has **one argument**, here named `v`:

```
> hm <- function(v) exp(sum(log(v))/length(v))
> class(hm)
> hm(1:99); mean(1:99)
```

```
[1] "function"
[1] 37.6231
[1] 50
```

A better version

A function should check for valid inputs. This shows the use of the `if`, `else if`, `else` **control structure**:

```
> hm <- function(v) {
    if (!is.numeric(v)) {
      print("Argument must be numeric"); return(NULL)
      }
    else if (any(v <= 0)) {
      print("All elements must be positive"); return(NULL)
      }
    else return(exp(sum(log(v))/length(v)))
    }
> hm(letters)
> hm(c(-1, -2, 1, 2))
> hm(1:99)

[1] "Argument must be numeric"
NULL
[1] "All elements must be positive"
NULL
[1] 37.6231
```

Another example

The "correlation of two random normal vectors" script can be converted to a function; the **arguments** are the number of runs and sample size:

```
> corr.two.random.normal <- function(m =1000, n=100) {
+  results <- rep(0, m)
+  for (i in 1:m) {
+    x <- rnorm(100); y <- rnorm(100) # default mu=0, sigma=1
+     results[i] <- cor(x, y)
+  }
+  summary(results)
+  tmp <- qplot(results, binwidth=0.02)
+  print(tmp + geom_bar(colour="white", fill="darkgreen", binwidth=0.02) + geom_rug())
+ }
```

The function is now defined in the workspace; to call it:

```
> corr.two.random.normal()  # with defaults
> corr.two.random.normal(256, 20)  # specify m and n
```

Try it! The second histogram will be much more erratic than the first.

Some advanced statistical functions

This is a very small sample of what is available.

Modelling

- Non-linear model fitting: `nls`
- Non-linear mixed-effects models: `nlme` package
- Generalized linear models (GLM): `glm`
- Robust fitting of linear models: `lqs`, `lm.ridge` etc.
- Local (smooth) fitting: `loess`
- Stepwise regression: `step`
- Regression trees: `trees`, `rpart` packages
- Principal component, partial least squares: `prcomp`, `pls` packages
- Random forests: `randomForest` package

Simulation

- Bootstrapping: `boot` package

Time and space

- Time-series analysis: `ts`, `arima` etc.
- Spatially-explicit objects: `sp` package
- Geostatistics: `gstat`, `geoR`, `spatial` packages
- Space-time geostatistics: `spacetime` package
- Point-pattern analysis: `spatstat`, `spatial` packages
- Areal spatial data analysis (like GEODA): `spdep` package
- Interface to GIS: `rgdal`, `RSAGA` packages
- Image processing: `raster` package

Resources for learning R

R is very popular and widely-used; in the spirit of the open-source movement many working statisticians and application scientists have written documentation.

- Introductions and tutorials
- On-line help (within R and on the Internet)
- Contributed documentation
- Textbooks
- Task views
- R Journal, Mailing lists, user's conference

General introductions

- **Venables, W. N. ; Smith, D. M.** ; R Development Core Team, 2014. *An Introduction to R* (Notes on R: A Programming Environment for Data Analysis and Graphics), updated at each version of R

  `http://www.cran.r-project.org`; also included with R distribution

  The standard introduction. This links to:

- **Hornik, K.** 2007. *R FAQ*: Frequently Asked Questions on R. Also updated with each version.

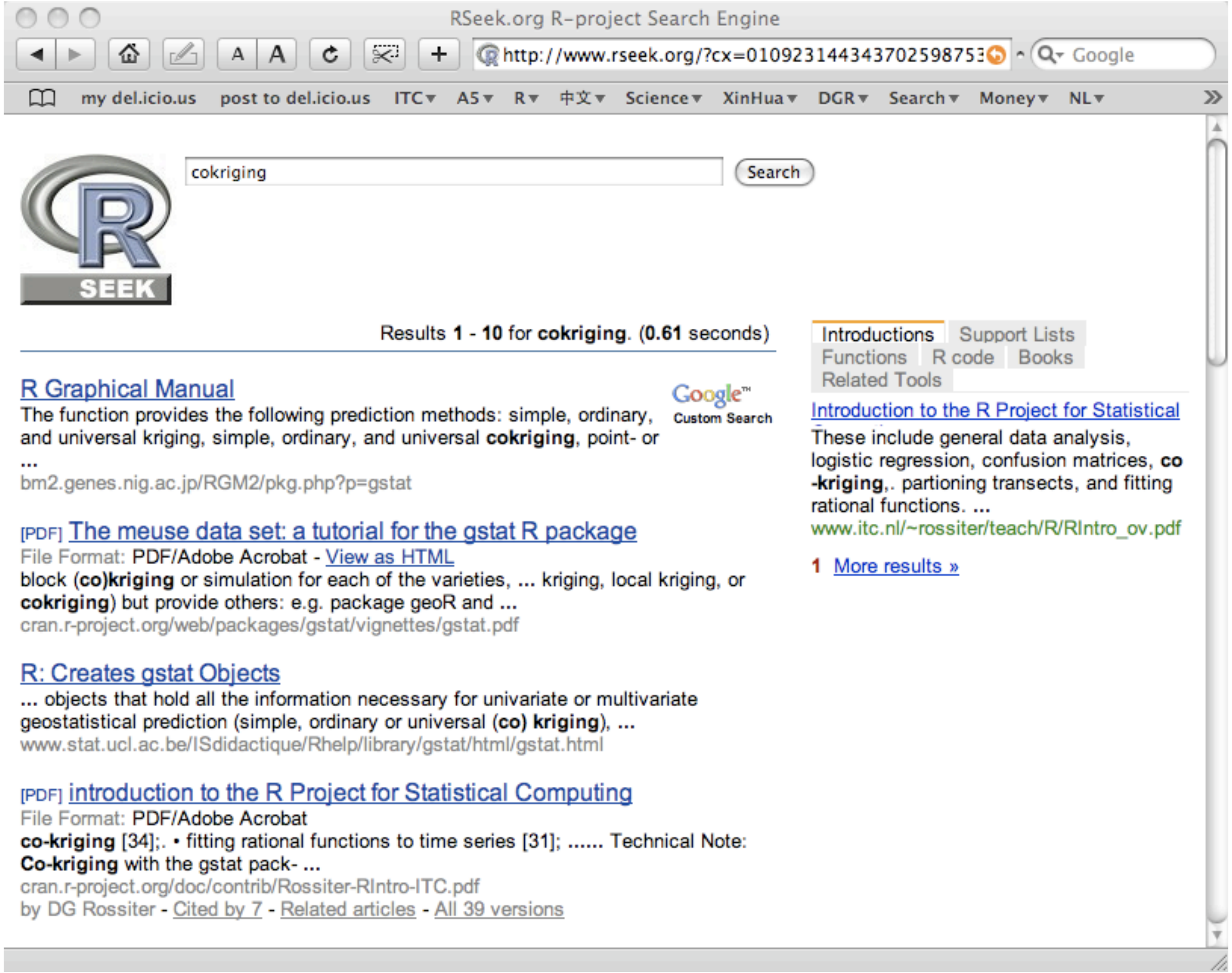
  What is R? Why 'R'? Availability, machines, legality, documentation, mailing lists . . .

  These are updated with each R release.

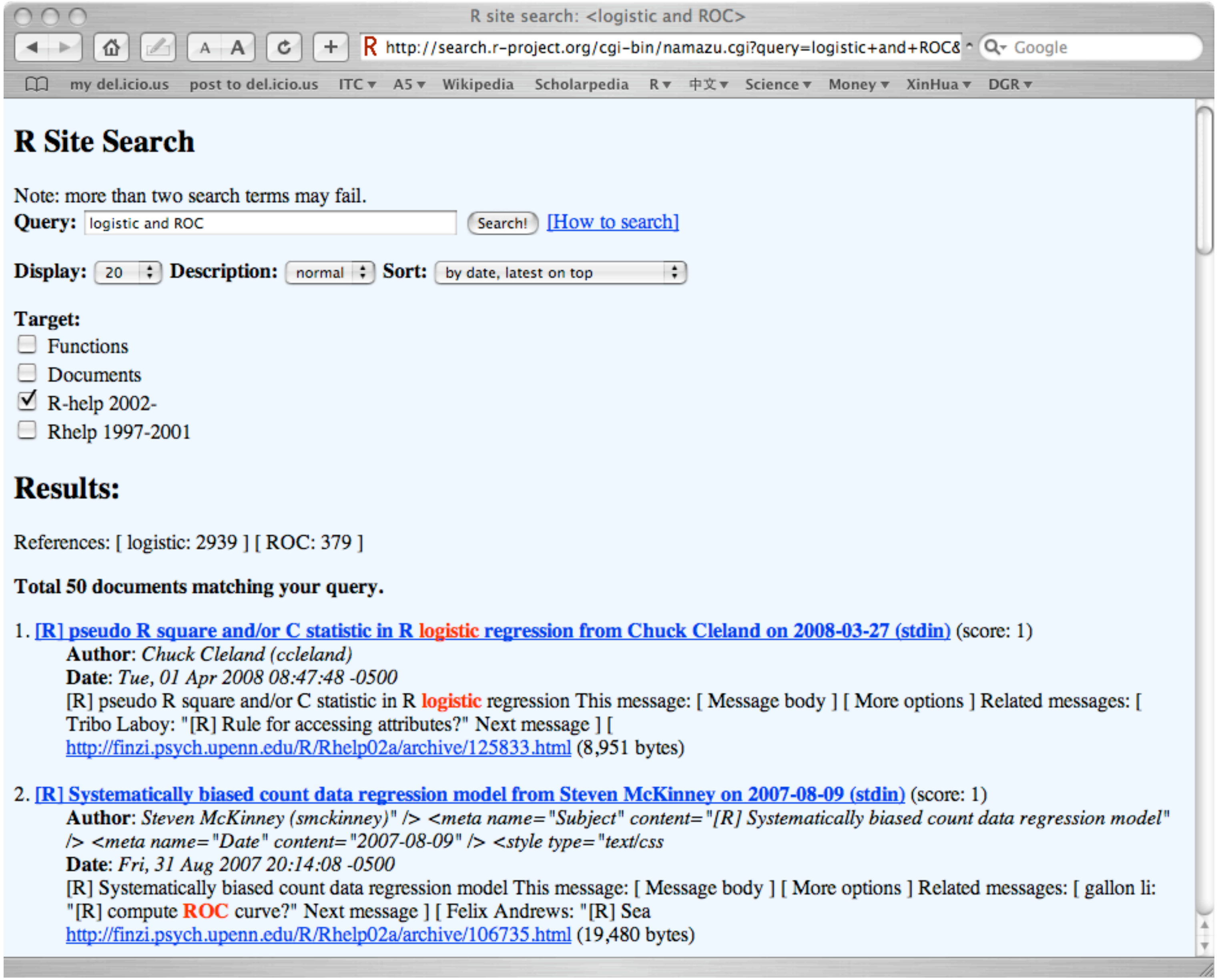
On-line help

- **Within the R environment**: `help` method, abbreviated `?`; `help.search` method
- **On the internet**
  - RSeek: `http://www.rseek.org/`
  - `RSiteSearch` method

RSeek results



`RSiteSearch` method results



Textbooks using R

More and more texts are using R code to illustrate their statistical analyses.

- **Dalgaard, P.** 2002. *Introductory Statistics with R*. Springer Verlag.

  This is a clearly-written introduction to statistics, using R in all examples.

- **Venables, W. N. & Ripley, B. D.** 2002. *Modern applied statistics with S*. New York: Springer-Verlag, 4th edition; `http://www.stats.ox.ac.uk/pub/MASS4/`

  Presents a wide variety of up-to-date statistical methods (including spatial statistics) with algorithms coded in S; includes an introduction to R, R programming, and R graphics.

- **Fox, J. 2002.** *An R and S-PLUS Companion to Applied Regression*. Newbury Park: Sage.

  A social scientist explains how to use R for regression analysis, including advanced techniques; this is a companion to his text: Fox, J. 1997. *Applied regression, linear models, and related methods*. Newbury Park: Sage

The UseR! series

Springer is publishing a series of **practical introductions with R code** to topics such as:

- data manipulation
- Bayesian analysis
- spatial data anlysis
  - **Bivand, R. S., Pebesma, E. J., & Gómez-Rubio, V** 2008. *Applied Spatial Data Analysis with R*: Springer; UseR! series. `http://www.asdar-book.org/`
- time-series
- interactive graphics

List at `http://www.springer.com/series/6991`

Technical Notes using R

I have written a number of technical notes showing how to accomplish some statistical tasks with R; the full list is at

`http://www.css.cornell.edu/faculty/dgr2/tutorials/index.html`

These include general data analysis, logistic regression, confusion matrices, co-kriging, partioning transects, and fitting rational functions.

R Task Views

Some applications are covered in so-called **Task Views**, on-line at `http://cran.r-project.org/web/views/index.html`.

These are a summary by a task maintainer of the facilities in R (e.g., which packages and functions to use) to accomplish certain tasks. Examples:

- **Analysis of Spatial Data**
  `http://cran.r-project.org/web/views/Spatial.html`
- **Multivariate Statistics**
  `http://cran.r-project.org/web/views/Multivariate.html`

Keeping up with developments in R

R is a **dynamic environment**, with a large number of dedicated scientists working to make it both a rich statistical computing environment and a modern programming language.

Daily **new and modified packages** added to CRAN; **new versions of the R base** appear 2–4x yr^{-1}

- **R Journal**: about 4x yr^{-1}; `http://journal.r-project.org/`

  News, announcements, tutorials, programmer's tips, bibliographies

- **Journal of Statistical Software**; `http://www.jstatsoft.org/`

  (continued . . .)

...

- **Mailing lists**: "Mailing Lists" link at CRAN:
  - *R-announce*: major announcements, e.g. new versions
  - *R-packages*: announcements of new or updated packages
  - *R-help*: discussion about problems using R, and their solutions. The R gurus monitor this list and reply as necessary. A search through the archives is a good way to see if your problem was already discussed.

- **useR!** user's conference; proceedings on-line; tutorials, workshops, user presentations, thematic sessions

Topics – Part 4

1. Reproducible research and literate programming

2. The Tidyverse

Reproducible research and literate programming

Reproducible research: "research papers with accompanying software tools that allow the reader to directly reproduce the results and employ the computational methods that are presented in the research paper."

Literate programming:

- both code and comments in the same document; code is executed and produces the results seen in the document; no cut-and-paste
- if data changes, document changes (code is the same, results are different!)

See: Rossiter, DG 2012. Technical Note: Literate Data Analysis using the R environment for statistical computing and the knitr package 26-December-2012, 35 pp; `http://www.css.cornell.edu/faculty/dgr2/_static/files/R_PDF/LDA.pdf`

The Tidyverse

- "[A]n **opinionated** collection of R packages designed for data science[1]
  - The "opinion" of Hadley Wickham
  - Main packages `dplyr`, `tidyr`, `readr`, `stringr`, `tibble`, `ggplot2`

- Well-explained in the (free) on-line text R for Data Science[2]

- Defines a syntax for **pipes** (`magrittr` package), for sequences of operations without having to define intermediate workspace objects

- Defines the **tibble**: "a modern re-imagining of the data frame, keeping what time has proven to be effective, and throwing out what it has not."

[1] `https://www.tidyverse.org`

[2] `https://r4ds.had.co.nz`

Pipes

Example:

```
the_data <-
  read.csv('/path/to/data/file.csv') %>%
  subset(variable_a > x) %>%
  transform(variable_c = variable_a/variable_b) %>%
  head(100)
```

- Only one workspace object (`the_data`) is created
- the results of each expression are passed to the next with the pipe operator %>%.

Exposing variables in a dataframe with the %$% operator :

```
data(iris) # Edgar Anderson's Iris Data, in datasets package
iris %>%
  subset(Sepal.Length > mean(Sepal.Length)) %$%
  cor(Sepal.Length, Sepal.Width)
```

See `https://magrittr.tidyverse.org` for more examples and complete syntax.