1/11 3:19:44 ***

← →

R

Why R?

- R is based on S, which is a statistics programming language - so for R, statistics runs in its blood :) Python, Java etc. can do what R can do, but they are general-purpose languages
- very feature-rich, including statistics operations, graphics output..
- provides a suite of operators for calculations on arrays, lists, vectors and matrices
- provides a large, coherent and integrated collection of tools for data analysis
- rich I/O: supports R importing data from CSV files, SAS, and SPSS, or directly from Microsoft Excel, Microsoft Access, Oracle, MySQL, and SQLite; can produce graphical output in PDF, JPG, PNG, and SVG formats, and table output for LaTeX and HTML
- offers a mix of OO and functional programming constructs
- data sets are autosaved between sessions
- cross-platform
- extensible

- open source, PLENTY of add-ons (libraries) [~5000!!]; great user community

To summarize: with R, you can do pretty much anything data-related: I/O, statistics, arbitrary math calculations, data mining, machine learning, visualization.. This is what distinguishes R (a single-minded, special-purpose language) from Python (a Swiss Army Knife language).

A fun example - R is used in 'sports analytics', here as a visualization tool. This is an example of 'data-driven animation'. Another one - here is Kobe Bryant's entire career :)

Where is R?

R's main page ("homepage") is https://www.r-project.org/

History

Wikipedia: 'S is a statistical programming language developed primarily by John Chambers and (in earlier versions) Rick Becker and Allan Wilks of Bell Laboratories. The aim of the language, as expressed by John Chambers, is "to turn ideas into software, quickly and faithfully".'

Scheme is an elegant language, derived from Lisp (an AI language) - Scheme's has clear and simple semantics, and offers many ways to form expressions.

R owes its origins to both S and Scheme - its interpreter is Scheme-based, and its purpose is similar to that of S, to serve as a language for statistics.

You can read more about R's origins here.

IDE(s)

You can choose between several IDEs (Integrated Development Environments) and GUI front ends:

- RStudio: http://www.rstudio.org/ [and optionally/additionally, Shiny, a web application framework] - IDE
- Tinn-R (for Windows only): https://sourceforge.net/projects/tinn-r/ - IDE
- R Commander: http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/ - GUI
- Rattle: http://rattle.togaware.com/ - GUI, tab-oriented (here is a note on it)
- JGR (Java GUI for R): https://cran.r-project.org/web/packages/JGR/index.html - GUI
- StatET: http://www.walware.de/goto/statet/ - Eclipse plugin
- ESS (Emacs Speaks Statistics): http://ess.r-project.org/ - Emacs R package

If you want to pick one to start with, make it RStudio. Note that you need to first install R (from https://cran.r-project.org/src/base/R-3/) before installing RStudio [or any other IDE or GUI front-end].

Syntax, semantics

So what does R actually look like, what datatypes does it have, what operators? How are functions declared and called?

R particularly shines when it comes to operations on vectors (ordered LIST of elements), arrays, matrices and tables - all of which are built-in ('native') datatypes. Just like in Python (where list is the central data type), in R, it is all about vectors (R does have a list type as well).

R code can be executed interactively in a shell, or be run from a .R source file (script).

R is a functional programming language..

```
# create a vector called x
x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

```
typeof(x) [and class(x) and mode(x)]

objects()

# alternative
assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))

c(10.4, 5.6, 3.1, 6.4, 21.7) -> x # !!!
1/x

y <- c(x, 0, x) # y will have 11 entries

v <- 2*x + y + 1
# generates a new vector v of length 11 constructed by adding
together,
# element by element, 2*x repeated 2.2 times, y repeated just
once,
# and 1 repeated 11 times
```

Here are sample operations (in RStudio):

```
> # create a vector called x
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> typeof(x)
[1] "double"
> objects()
[1] "x"
> c(10.4, 5.6, 3.1, 6.4, 21.7) -> x
> 1/x
[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
>
>
>
> sqrt(-17+0i)
[1] 0+4.123106i
>
> sqrt(3+2i)
[1] 1.817354+0.550251i
>
```

Several functions operate on a vector, eg. range(), min(), max(), sum(), prod(), mean(), sort()..

Sequences are easy to create:

```
# c(1,2,3,4.....30)
> m <- 1:30
> m
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
[20] 20 21 22 23 24 25 26 27 28 29 30

> n <- seq(2,5)
> n
```

```
[1] 2 3 4 5
>
```

Logical vectors (comprised of T,F) are created via conditions, eg.

```
> temp <- x>13
> temp
[1] FALSE FALSE FALSE FALSE  TRUE
>
```

Matrices/arrays are muti-dim versions of vectors.

Lists can have elements of unequal types (vectors cannot).

Data frames are matrix-like structures, in which the columns can be of different types - this is how we can create/read in relations (table data!)! Eg. data.frame(x,y) will create a 2-column (x,y) table out of vectors x and y.

Functions are objects as well (like with Python, JavaScript..). "The function tapply() is used to apply a function, here

'mean()', to each group of components of the first argument, here 'incomes', defined by the levels of the second component, here 'statef'" (sound a lot like Python's map()?):

```
> incmeans <- tapply(incomes, statef, mean)
```

Easy to write vector-oriented functions:

```
> stderr <- function(x) sqrt(var(x)/length(x))
```

Arrays can be created from vectors, via dim():

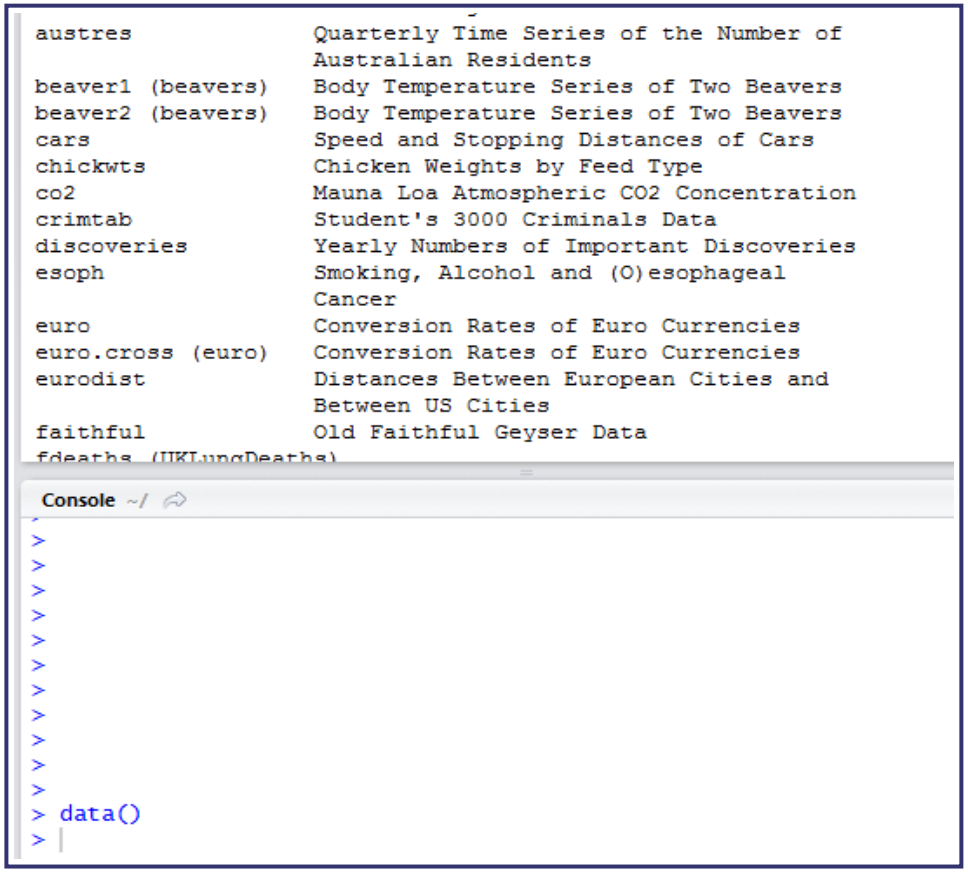
```
# if z has 1500 elements, here is how it can be turned
# into a 3D array:

> z = 1:1500
> dim(z) <- c(3,5,100)
> z
, , 1

     [,1] [,2] [,3] [,4] [,5]
```

```
[1,]    1    4    7   10   13
[2,]    2    5    8   11   14
[3,]    3    6    9   12   15

, , 2

     [,1] [,2] [,3] [,4] [,5]
[1,]   16   19   22   25   28
[2,]   17   20   23   26   29
[3,]   18   21   24   27   30

, , 3

     [,1] [,2] [,3] [,4] [,5]
[1,]   31   34   37   40   43
[2,]   32   35   38   41   44
[3,]   33   36   39   42   45
```

. . . .

. . . .

data() lists sample datasets:



edit() is used to edit data, spreadsheet-fashion:

```
xnew <- edit(xold)

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

R has a lot of probability distribution functions built-in:

Distribution	R name	additional arguments
beta	`beta`	`shape1, shape2, ncp`
binomial	`binom`	`size, prob`
Cauchy	`cauchy`	`location, scale`
chi-squared	`chisq`	`df, ncp`
exponential	`exp`	`rate`
F	`f`	`df1, df2, ncp`
gamma	`gamma`	`shape, scale`
geometric	`geom`	`prob`
hypergeometric	`hyper`	`m, n, k`
log-normal	`lnorm`	`meanlog, sdlog`
logistic	`logis`	`location, scale`
negative binomial	`nbinom`	`size, prob`
normal	`norm`	`mean, sd`
Poisson	`pois`	`lambda`
signed rank	`signrank`	`n`
Student's t	`t`	`df, ncp`
uniform	`unif`	`min, max`
Weibull	`weibull`	`shape, scale`
Wilcoxon	`wilcox`	`m, n`

A stem-and-leaf plot of data is easy to generate:

```
> attach(faithful) # READ in a dataset
> summary(eruptions)
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.600 2.163 4.000 3.488 4.454 5.100
> fivenum(eruptions)
[1] 1.6000 2.1585 4.0000 4.4585 5.1000
> stem(eruptions)
The decimal point is 1 digit(s) to the left of the |
16 | 070355555588
18 | 000022233333335577777778888822335777888
20 | 000022233788000357 78
22 | 0002335578023578
24 | 00228
26 | 23
28 | 080
30 | 7
32 | 2337
34 | 250077
36 | 0000823577
38 | 2333335582225577
40 | 000000335778888800223355557777 8
42 | 033355557788002333335555777 78
44 | 022223355577800000000233333577788 88
46 | 000023335770000002357 8
```

```
48 | 00000022335800333
50 | 0370
```

There is more to learn! If you are interested, go through the books/sites listed at the end of this lecture. Meanwhile, here is a .R script filled with practice commands; this is another (neither of these are mine).

Here is a nice reference card that summarizes all the aspects of R.

Examples

Here is a 'real world' R script.

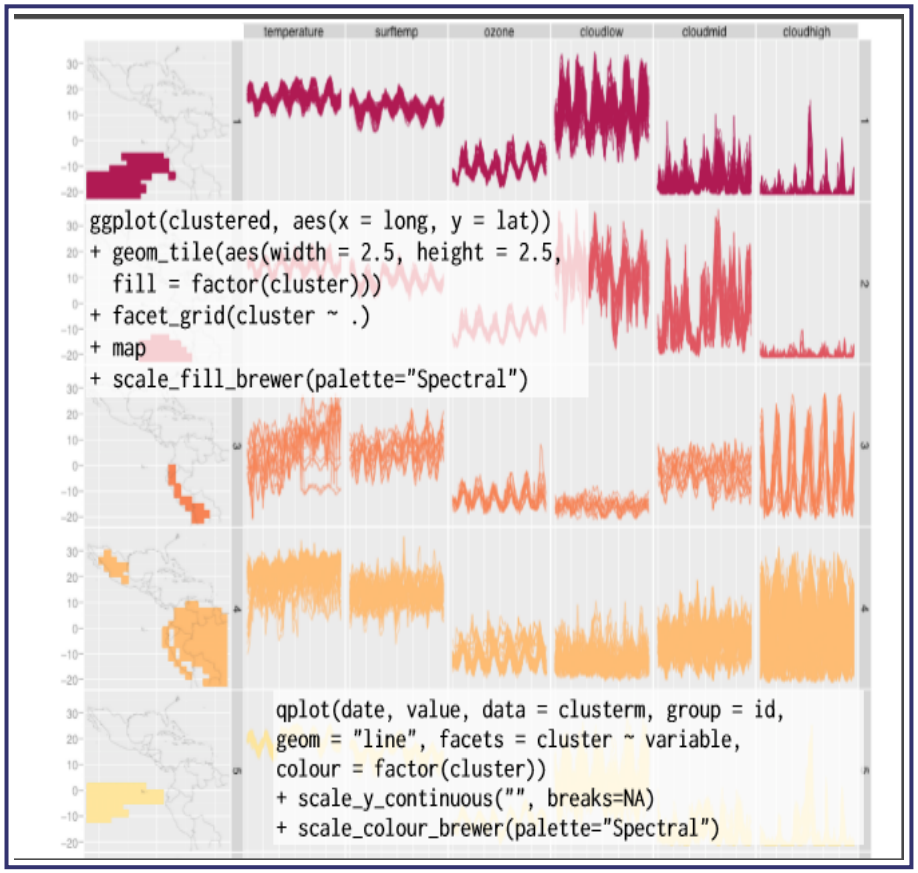
Here, R is used for text mining.

Packages

Just like Python has modules and Java has class libraries, R has packages, which are collections of useful functions - these effectively help extend R (by introducing new input and output types, and new functions).

Here are some popular R packages:

- ggplot2: http://had.co.nz/ggplot2/ - high-level package for creating statistical graphics (compare with matplotlib, LaTeX etc).



- plotly - another plotting package
- plyr: SAC (split-apply-combine) dataflow; dplyr - next-gen plyr
- lubridate - to process dates
- knitr - for report generation
- rCharts - for creating interactive, JavaScript-based visualizations (eg. look at the rCharts gallery samples)

Want to discover and utilize even more packages? Look here :)

http://r-pkgs.had.co.nz/ is the page for a book on (writing) R packages - the site is quite informative (as is the book).

R + Python

PypeR is a Python interface to R.

rPy2 is another module that also provides access to R from inside Python.

Jupyter is a programming workbench that supports mixing and matching multiple languages, including Python and R.

Python or R? Hard to choose :) This is a comparison..

R + Java

Using the RWeka package, it is possible to hook into the Weka runtime, via R commands..

More

There is a *wealth* of material out of there for you to explore further!

Books

There are numerous books on R. Here are ten that I recommend:

- The Art of R Programming, by Norman Matloff : if you want to read just one book on R, read this one!
- Data Mashups in R : contains numerous ideas to use R along with other tools/programs
- R for the Impatient: http://www.burns-stat.com/documents/tutorials/impatient-r/
- A Beginner's Guide to R
- https://cran.r-project.org/doc/manuals/R-intro.pdf
- R Cookbook - with specific 'recipes' (how-tos)
- ....

- Econometrics in R
- Statistics with R ("SwR"): http://zoonek2.free.fr/UNIX/48_R/all.html - this is a MASSIVE BOOK!
- ....
- R Graphics Cookbook and associated site
- Another R Graphics book

Notes [sites devoted to R]

- Quick-R: http://www.statmethods.net/
- The R Journal: https://journal.r-project.org/index.html
- RDataMining: http://www.rdatamining.com/
- RNotes: http://sphaerula.com/legacy/R/index.html
- RTips: http://pj.freefaculty.org/R/Rtips.html
- OnePageR: http://togaware.com/onepager/ - short and sweet notes on R topics
- R Tutorial: http://www.r-tutor.com/
- R Examples: http://www.rexamples.com/

Courses

- https://www.edx.org/course/mitx/mitx-15-071x-analytics-edge-1416
- https://www.edx.org/course/explore-statistics-r-kix-kiexplorx-0#.U7reOf6HaR8
- https://www.coursera.org/learn/r-programming
- R for Data Science: http://r4ds.had.co.nz/
- https://www.codeschool.com/courses/try-r
- http://swirlstats.com/ ['swirl' is a package that lets you learn R, right inside R :)] [from within RStudio, do '> install.packages("swirl")', then '>library("swirl")', then '>swirl()', to bring up swirl]

Misc

- mailing list
- RSeek
- A complete Microsoft tutorial in a single web page
