Intro to *R*

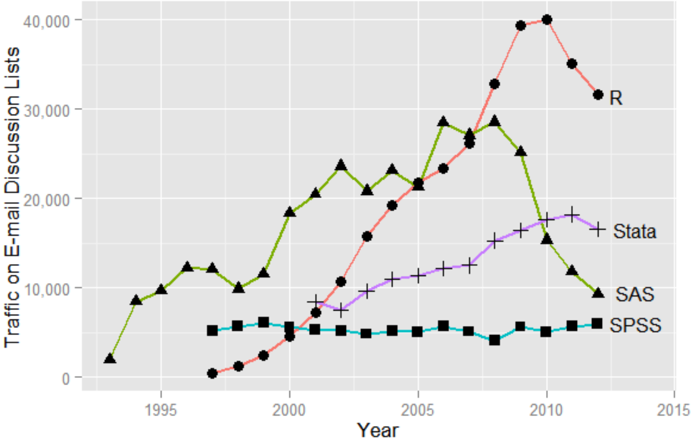
MTH 3220

Contents

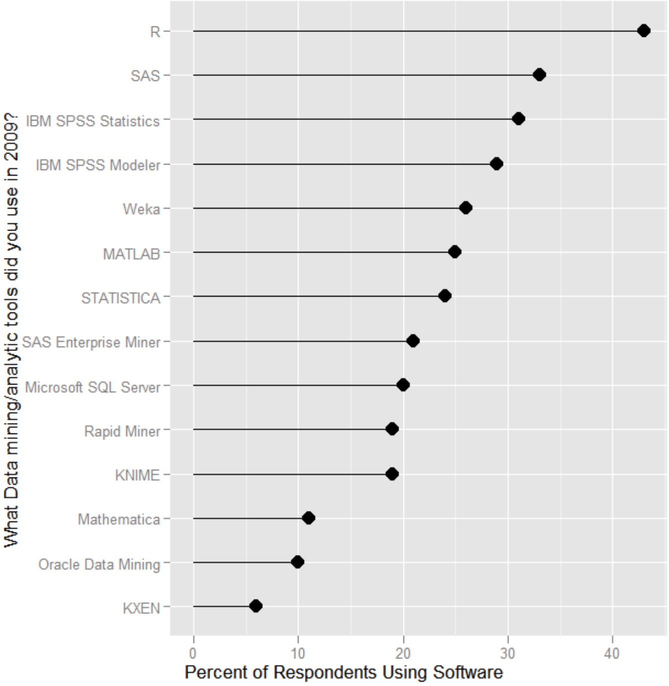
1 Introduction and History

1.1 *R* in use

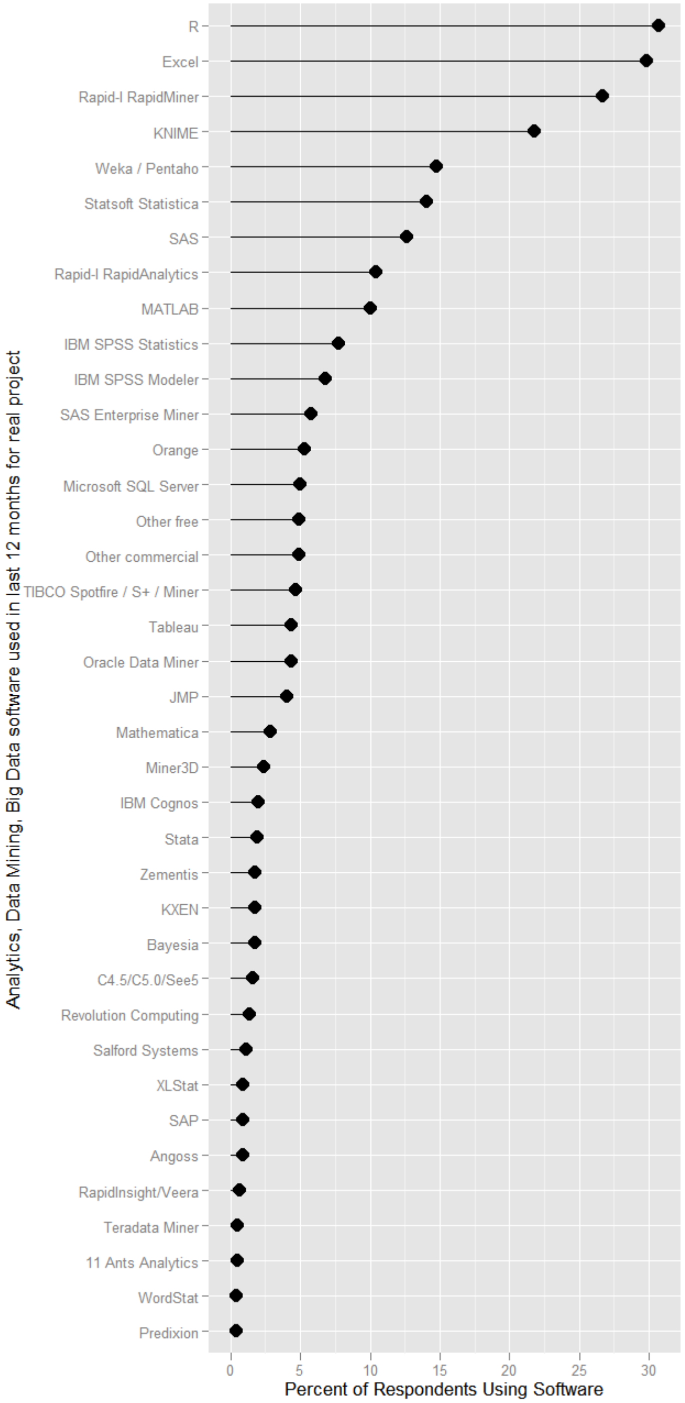
A compelling reason to learn any language: the more people speaking a language, the easier it is to get help, the longer it will stay around, and the more people you can *talk* with.



link



link



link

R's success perhaps attributed to its GPL2 license: created *by* users *for* users to run well on Linux, Mac and Windows. Thus performance is and always will be the primary focus, not marketing or other extraneous efforts that don't benefit the customer.

1.2 History

Before *R* there was the statistical analysis program “S”, developed at Bell Laboratories in 1976 by John Chambers. S was later commercialized as S-PLUS. John Chambers is also a current member of the *R* core group responsible for developing *R*. Ross Ihaka and Robert Gentleman of the Department of Statistics, University of Auckland, New Zealand, began coding (1991) the S clone “*R*” as an open source alternative for academic use. Together with the *R* core group, they released version 1 under GPL2 and GPL3 in 2000. The name *R* probably derives from the first letter of the creators’ names.

1.3 What *is* *R* and how do we use it?

R is a command based, object oriented, functional language; in contrast to Excel which is a cell centric, spreadsheet managing graphical user interface (GUI). CRAN defines it as follows: “*R* is ‘GNU S’,a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modeling, statistical tests, time series analysis, classification, clustering, etc.”

```
##   newObject    <-           function(object)      # a comment
##   ^            ^            ^
##   object       assignment   expression
```

1.4 How *R* works

You type commands, also known as expressions, into a text editor or terminal and send them to *R* for evaluation. *R* evaluates the command, prints the command (by default), and the result of the evaluated command, if any.

1.5 Core of *R*

Define your function:

```
myFunc <- function(object = TRUE) {
  if (object) {
    print("hello world")
  }
}
```

Apply your function to an object:

```
myFunc(TRUE)

## [1] "hello world"

myFunc(FALSE)
```

2 Objects and Functions

2.1 Things you can have

Any *thing* in *R* is of a certain type, defined as a `class` and a *mode*. As an analogy, think of numbers as fruits and text as vegetables. They're not the same, although they can be analyzed, or “cooked” like one another.

2.1.1 Numbers

Fruits.

```
class(1)

## [1] "numeric"

mode(1)

## [1] "numeric"

class(2.3123)

## [1] "numeric"

anInteger <- as.integer(2.3123)
anInteger

## [1] 2

class(anInteger)

## [1] "integer"

mode(anInteger)

## [1] "numeric"

numericAgain <- as.numeric(anInteger) # change class to numeric
class(numericAgain)

## [1] "numeric"
```

2.1.2 Text

The *Veggies*.

```r
class("a")

## [1] "character"

mode("a")

## [1] "character"

class("1")

## [1] "character"

mode("1")

## [1] "character"

a_factor <- factor("a")
a_factor

## [1] a
## Levels: a

class(a_factor)

## [1] "factor"

mode(a_factor)

## [1] "numeric"
```

2.1.3 Logical

It can only be a *yes* or a *no*. More specifically, a `TRUE` or a `FALSE`.

```r
class(TRUE)

## [1] "logical"

class(FALSE)
```

```
## [1] "logical"

mode(TRUE)

## [1] "logical"
```

2.1.4 Functions

An object that does something to the *fruits* and *veggies*: a *frying pan*! Remember the function we defined?

```
myFunc

## function(object = TRUE) {
##   if (object) {
##     print("hello world")
##   }
## }

class(myFunc)

## [1] "function"

mode(myFunc)

## [1] "function"
```

2.2 Containers: more object classes and modes

Besides the basic types of aforementioned things, there are different *containers* available to hold these things, which are also defined by *class* and *mode*. Containers are thus things, or `objects`, that you can put other things, or `objects`, into to conveniently hold them while we work with them.

2.2.1 Vectors: ordered series of elements

Think of them like a *string* or *chain*.

```
aVector <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
class(aVector)

## [1] "numeric"
```

```
mode(aVector)

## [1] "numeric"

length(aVector)

## [1] 12

bVector <- c(1)
length(x = bVector)

## [1] 1
```

Combine vectors to make another vector:

```
combinedVec <- c(aVector, bVector)
combinedVec

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12  1
```

2.2.2 Matrices: vectors with two dimensions

A *tray* upon which to lay the string/chain, first looped top-to-bottom, then left-to-right (unless otherwise specified.

```
aMatrix <- matrix(data = aVector)
aMatrix

##       [,1]
##  [1,]    1
##  [2,]    2
##  [3,]    3
##  [4,]    4
##  [5,]    5
##  [6,]    6
##  [7,]    7
##  [8,]    8
##  [9,]    9
## [10,]   10
## [11,]   11
## [12,]   12

aVector
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12

dim(aMatrix)

## [1] 12  1

bMatrix <- matrix(data = aVector, nrow = 3)
bMatrix

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

dim(bMatrix)

## [1] 3 4

lettersMat <- matrix(data = letters[1:12], nrow = 3)

class(aMatrix)

## [1] "matrix"

mode(aMatrix)

## [1] "numeric"

class(lettersMat)

## [1] "matrix"

mode(lettersMat)

## [1] "character"
```

2.2.3 Arrays are matrices

```
anArray <- array(letters[1:12], c(2, 2, 3))
anArray

## , , 1
```

```
##
##      [,1] [,2]
## [1,] "a"  "c"
## [2,] "b"  "d"
##
## , , 2
##
##      [,1] [,2]
## [1,] "e"  "g"
## [2,] "f"  "h"
##
## , , 3
##
##      [,1] [,2]
## [1,] "i"  "k"
## [2,] "j"  "l"

class(anArray)

## [1] "array"

mode(anArray)

## [1] "character"
```

2.2.4 Lists hold just about anything and everything

Think of them like a *shopping list* of: a string of fruit, a tray of veggies, an array of text, any other sublist of items...

```
aList <- list(aVector, aMatrix, lettersMat)
aList

## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## [[2]]
##       [,1]
##  [1,]    1
##  [2,]    2
##  [3,]    3
##  [4,]    4
##  [5,]    5
##  [6,]    6
##  [7,]    7
##  [8,]    8
##  [9,]    9
## [10,]   10
## [11,]   11
## [12,]   12
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,] "a"  "d"  "g"  "j"
## [2,] "b"  "e"  "h"  "k"
## [3,] "c"  "f"  "i"  "l"
```

```
class(aList)
```

```
## [1] "list"
```

```
mode(aList)
```

```
## [1] "list"
```

```
namedList <- list("vec" = aVector,
                  "mat" = aMatrix,
                  "lets" = lettersMat,
                  "logi" = matrix(rep(c(TRUE, FALSE), 5), nrow = 5),
                  "lis" = list(1:5, letters[1:9]))
namedList
```

```
## $vec
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## $mat
##       [,1]
##  [1,]    1
##  [2,]    2
##  [3,]    3
##  [4,]    4
##  [5,]    5
##  [6,]    6
##  [7,]    7
##  [8,]    8
##  [9,]    9
## [10,]   10
## [11,]   11
## [12,]   12
##
## $lets
##      [,1] [,2] [,3] [,4]
## [1,] "a"  "d"  "g"  "j"
## [2,] "b"  "e"  "h"  "k"
## [3,] "c"  "f"  "i"  "l"
##
## $logi
##       [,1]  [,2]
## [1,]  TRUE FALSE
## [2,] FALSE  TRUE
## [3,]  TRUE FALSE
## [4,] FALSE  TRUE
## [5,]  TRUE FALSE
##
## $lis
## $lis[[1]]
## [1] 1 2 3 4 5
##
## $lis[[2]]
## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i"
```

```
class(namedList)
```

```
## [1] "list"
```

```
mode(namedList)
```

```
## [1] "list"
```

```
namedList[["logi"]]
```

```
##       [,1]  [,2]
## [1,]  TRUE FALSE
## [2,] FALSE  TRUE
## [3,]  TRUE FALSE
## [4,] FALSE  TRUE
## [5,]  TRUE FALSE
```

```
class(namedList[["logi"]])

## [1] "matrix"

mode(namedList[["logi"]])

## [1] "logical"
```

2.2.5 Data frames are special lists

Data frames are even more similar to the familiar excel spreadsheets: a series, or *list*, of equal length columns, or *vectors*. Notably, the columns (*vectors*) can be of different classes unlike a matrix or array.

```
aDF <- data.frame("vec" = aVector, "lets" = letters[1:12])
aDF

##    vec lets
## 1    1    a
## 2    2    b
## 3    3    c
## 4    4    d
## 5    5    e
## 6    6    f
## 7    7    g
## 8    8    h
## 9    9    i
## 10  10    j
## 11  11    k
## 12  12    l

dim(aDF)

## [1] 12  2

class(aDF)

## [1] "data.frame"

mode(aDF)

## [1] "list"
```

```
str(aDF)

## 'data.frame':	12 obs. of  2 variables:
##  $ vec : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ lets: Factor w/ 12 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...

aDF[, 1]

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12

class(aDF[, 1])

## [1] "numeric"

mode(aDF[, 1])

## [1] "numeric"

aDF[, 2]

##  [1] a b c d e f g h i j k l
## Levels: a b c d e f g h i j k l

class(aDF[, 2])

## [1] "factor"

mode(aDF[, 2])

## [1] "numeric"

str(aDF[, 1])

##  num [1:12] 1 2 3 4 5 6 7 8 9 10 ...
```

It's important to understand that the things we work with in *R* have a 'mode' and a 'class'. Pay attention to your modes and classes.

2.3 Functions are things you can do

R comes with predefined functions which do many things from basic file management to complex statistics.

2.3.1 The Arithmetic Operators

That is, *R* as a calculator.

```
x <- 10
y <- 3
x + y

## [1] 13

x - y

## [1] 7

x * y

## [1] 30

x / y

## [1] 3.333333

x ^ y           # exponentiation

## [1] 1000

x %% y          # modular arithmetic, remainder after division

## [1] 1

x %/% y         # integer part of a fraction

## [1] 3
```

How these functions work on vectors and matrices:

```
aVector

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12

aVector + x

##  [1] 11 12 13 14 15 16 17 18 19 20 21 22

bMatrix
```

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

bMatrix + x

##      [,1] [,2] [,3] [,4]
## [1,]   11   14   17   20
## [2,]   12   15   18   21
## [3,]   13   16   19   22
```

2.3.2 Functions you'll use to interact with data

Relational Operators:

```
x < y

## [1] FALSE

x > y

## [1] TRUE

x <= y

## [1] FALSE

x >= y

## [1] TRUE

x == y

## [1] FALSE

x != y

## [1] TRUE
```

2.3.3 Functions which help you work with *R*

`str()` - print the structure of an object
`class()` - print the class of an object
`head()` - the first six elements/rows
`tail()` - the last six elements/rows
`ls()` - list all objects and functions held in your global environment

Generate a sequence:

```
seq(from = 1, to = 10, by = 2)

## [1] 1 3 5 7 9

## or
1:10

##  [1]  1  2  3  4  5  6  7  8  9 10
```

2.4 Define your own functions

```
do_something <- function(x){
  x + 10
}
do_something(5)

## [1] 15
```

2.5 Saving and restoring your session

```
## save.image()                     # saves all objects to the working
                                   # directory in a file called ".RData"
## save.image("filename.RData") # give the file a name

## savehistory()                    # saves all commands entered into the R
                                   # console during your session to the
                                   # working directory in a file called
                                   # ".Rhistory"
```

Save specific objects:

```
## save(object1, object2, object3, file = "r_objects123.RData")
```

Restore or load previous sessions or objects:

```
## load("filename.RData")        # load from the working directory
```

2.6 Getting Help

Offline help ships with *R*:

```
help(help)

## starting httpd help server ...
##  done

?help
```

Broaden your help search:

```
apropos("help")             # quotes are needed

## [1] "help"         "help.request" "help.search"  "help.start"

?help
```

Notes adapted from *Basic Course on R*, Erasmus MC, Rotterdam, the Netherlands (developed by Elizabeth Ribble and Karl Brand).