Introduction to R

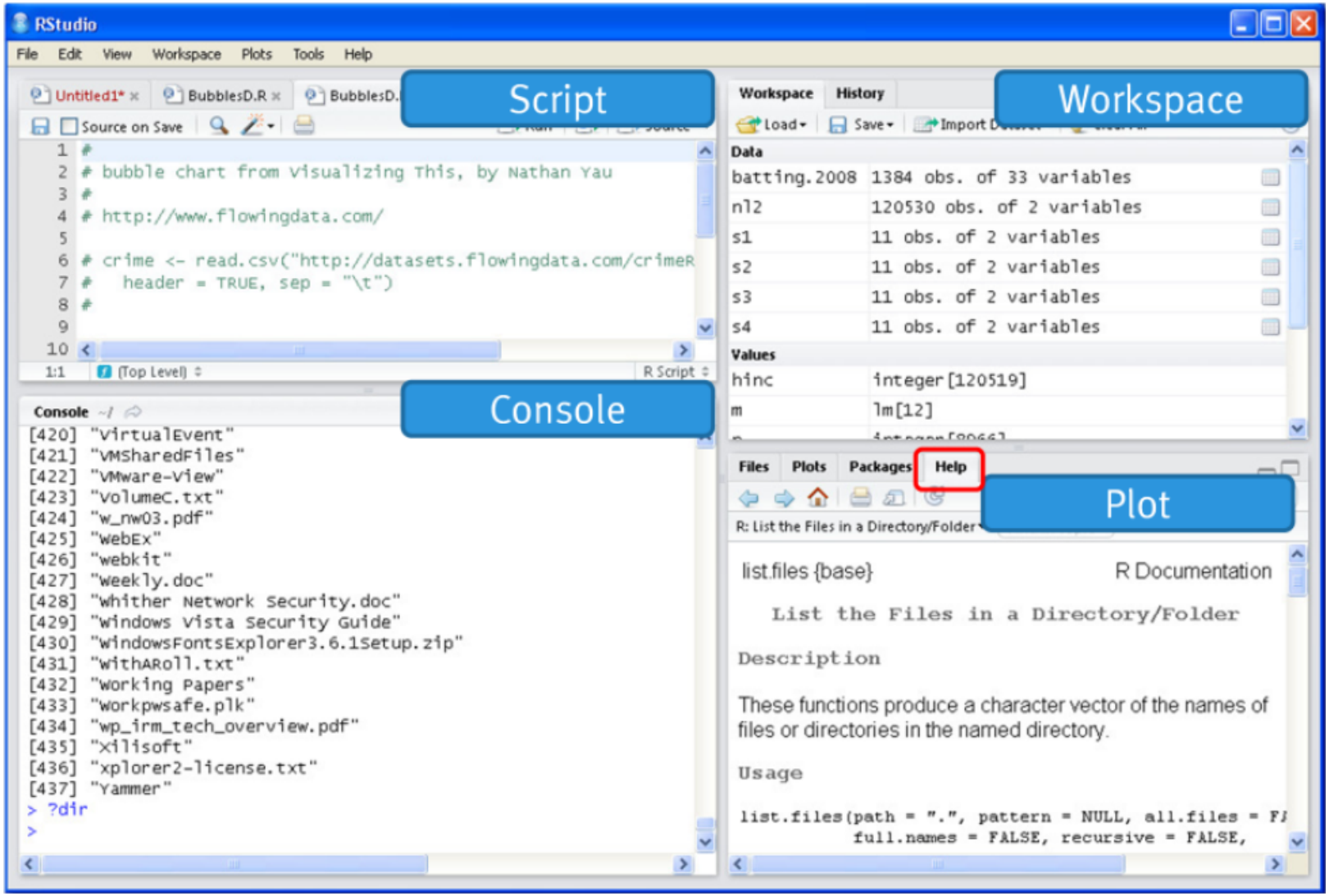
Mikhail Stepanov

October 4, 2012

Five things to remember about R

- (Almost) everything is an object
- (Almost) everything is a vector
  - a$\leftarrow$3 — a is 1x1 vector
  - v$\leftarrow$(1,2,3,4,5) is a 1x5 vector
- All commands are function
  - quit() or q()
- Some commands produce different output ...
- Know your default arguments!

RStudio


RStudio
File Edit View Workspace Plots Tools Help
Untitled1* BubblesD.R BubblesD.
Source on Save
Script
1 #
2 # bubble chart from Visualizing This, by Nathan Yau
3 #
4 # http://www.flowingdata.com/
5
6 # crime <- read.csv("http://datasets.flowingdata.com/crimeR
7 # header = TRUE, sep = "\t")
8 #
9
10
1:1 (Top Level) R Script
Workspace History
Load Save Import
Workspace
Data
batting.2008 1384 obs. of 33 variables
nl2 120530 obs. of 2 variables
s1 11 obs. of 2 variables
s2 11 obs. of 2 variables
s3 11 obs. of 2 variables
s4 11 obs. of 2 variables
Values
hinc integer[120519]
m lm[12]
Console
Console
[420] "VirtualEvent"
[421] "VMSharedFiles"
[422] "VMware-View"
[423] "VolumeC.txt"
[424] "w_nw03.pdf"
[425] "WebEx"
[426] "webkit"
[427] "weekly.doc"
[428] "whither Network Security.doc"
[429] "Windows Vista Security Guide"
[430] "WindowsFontsExplorer3.6.1Setup.zip"
[431] "WithARoll.txt"
[432] "Working Papers"
[433] "Workpwsafe.plk"
[434] "wp_irm_tech_overview.pdf"
[435] "Xilisoft"
[436] "xplorer2-license.txt"
[437] "Yammer"
> ?dir
>
Files Plots Packages Help
Plot
R: List the Files in a Directory/Folder
list.files {base} R Documentation
List the Files in a Directory/Folder
Description
These functions produce a character vector of the names of files or directories in the named directory.
Usage
list.files(path = ".", pattern = NULL, all.files = FA
full.names = FALSE, recursive = FALSE,

Objects

R has five basic or "atomic" classes of objects:

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

The most basic object is a vector

- A vector can only contain objects of the same class
- BUT: The one exception is a list, which is represented as a vector but can contain objects of different classes (indeed, that's usually why we use them)

Empty vectors can be created with the vector() function.

Numbers

- Numbers in R a generally treated as numeric objects (i.e. double precision real numbers)
- If you explicitly want an integer, you need to specify the L suffix Ex: Entering 1 gives you a numeric object; entering 1L explicitly gives you an integer.
- There is also a special number Inf which represents infinity; e.g. 1/0; Inf can be used in ordinary calculations; e.g. 1/ Inf is 0
- The value NaN represents an undefined value ("not a number"); e.g. 0/0; NaN can also be thought of as a missing value

Attributes

R objects can have attributes

- names, dimnames
- dimensions (e.g. matrices, arrays)
- class
- length other user-defined attributes/metadata

Attributes of an object can be accessed using the attributes() function.

Entering Input

At the R prompt we type expressions. The ¡- symbol is the assignment operator.

```
> x <- 1
> print(x)
[1] 1
> x
[1] 1
> msg <- "hello"
```

The grammar of the language determines whether an expression is complete or not.

```
> x <- ## Incomplete expression
```

The # character indicates a comment. Anything to the right of the # (including the # itself) is ignored.

Evalution

When a complete expression is entered at the prompt, it is evaluated and the result of the evaluated expression is returned. The result may be auto-printed.

```
> x <- 5 ## nothing printed
> x ## auto-printing occurs
[1] 5
> print(x) ## explicit printing
[1] 5
```

The [1] indicates that x is a vector and 5 is the first element.

Making Sequence

The : operator is used to create integer sequences.

```
> x <- 1:20
> x
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
[16] 16 17 18 19 20
```

The : operator is used to create integer sequences.

Creating Vectors

The c() function can be used to create vectors of objects.

```
> x <- c(0.5, 0.6) ## numeric
> x <- c(TRUE, FALSE) ## logical
> x <- c(T, F) ## logical
> x <- c("a", "b", "c") ## character
> x <- 9:29 ## integer
> x <- c(1+0i, 2+4i) ## complex
```

Using the vector() function

```
> x <- vector("numeric", length = 10)
> x
[1] 0 0 0 0 0 0 0 0 0 0
```

Operations on Vectors

Function	R Code
Operations on Vectors	`v <- c(1:10); w <- c(15:24) ; nv <- v * pi ; nw <- w * v`
Vector transformations	radius ‹- sqrt(d$population)/ pi) t ‹- as.table(dfm$factor_variable) pct ‹- t/sum(t)* 100
Logical Vectors	v[v ‹ 1000] ndf ‹- subset(dfm, d$population ‹ 10000) nv ‹- v[c(1,2,3,5,8,13)]
Examining data structures	`dim(dfm); attributes(dfm) ;` `class(dfm); typeof(dfm)`

Matrices

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)

```
> m <- matrix(nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```

Matrices

Matrices are constructed column-wise, so entries can be thought of starting in the "upper left" corner and running down the columns.

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
```

Matrices

Matrices can also be created directly from vectors by adding a dimension attribute.

```
> m <- 1:10
> m
 [1]  1  2  3  4  5  6  7  8  9 10
> dim(m) <- c(2, 5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

cbind, rbind

Matrices can be created by column-binding or row-binding with cbind() and rbind().

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
x y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> rbind(x, y)
[,1] [,2] [,3]
x  1    2    3
y 10   11   12
```

Lists

Lists are a special type of vector that can contain elements of different classes. Lists are a very important data type in R.

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1
[[2]]
[1] "a"
[[3]]
[1] TRUE
[[4]]
[1] 1+4i
```

Factors

Factors are used to represent categorical data. Factors can be unordered or ordered. One can think of a factor as an integer vector where each integer has a label.

- Factors are treated specially by modelling functions like lm() and glm()
- Using factors with labels is better than using integers because factors are self-describing; having a variable that has values "Male" and "Female" is better than a variable that has values 1 and 2.

Factors

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
> x
[1] yes yes no yes no
Levels: no yes
> table(x)
x
no yes
2 3
> unclass(x)
[1] 2 2 1 2 1
attr(,"levels")
[1] "no" "yes"
```

Factors

The order of the levels can be set using the levels argument to factor(). This can be important in linear modeling because the first level is used as the baseline level.

```
> x <- factor(c("yes", "yes", "no", "yes", "no"),
levels = c("yes", "no"))
> x
[1] yes yes no yes no
Levels: yes no
```

Data Frames

- Data frames are used to store tabular data
- They are represented as a special type of list where every element of the list has to have the same length
- Each element of the list can be thought of as a column and the length of each element of the list is the number of rows
- Unlike matrices, data frames can store diferent classes of objects in each column (just like lists); matrices must have every element be the same class
- Data frames also have a special attribute called row.names
- Data frames are usually created by calling read.table() or read.csv()
- Can be converted to a matrix by calling data.matrix()

Data Frames

```
> x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
> x
foo bar
1 1 TRUE
2 2 TRUE
3 3 FALSE
4 4 FALSE
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

Names

R objects can also have names, which is very useful for writing readable code and self-describing objects.

```
> x <- 1:3
> names(x)
NULL
> names(x) <- c("foo", "bar", "norf")
> x
foo bar norf
1 2 3
> names(x)
[1] "foo" "bar" "norf"
```

Names

Lists can also have names.

```
> x <- list(a = 1, b = 2, c = 3)
> x
$a
[1] 1
$b
[1] 2
$c
[1] 3
```

Names

And matrices.

```
> m <- matrix(1:4, nrow = 2, ncol = 2)
> dimnames(m) <- list(c("a", "b"), c("c", "d"))
> m
  c d
a 1 3
b 2 4
```

if

This is a valid if/else structure.

```
if(x > 3) {
  y <- 10
} else {
  y <- 0
}
```

So is this one.

```
y <- if(x > 3) {
       10
} else {
       0
}
```

for

These three loops have the same behavior.

```
x <- c("a", "b", "c", "d")
for(i in 1:4) {
   print(x[i])
}
for(i in seq_along(x)) {
   print(x[i])
}
for(letter in x) {
   print(letter)
}
for(i in 1:4) print(x[i])
```

Functions

Functions are created using the function() directive and are stored as R objects just like anything else. In particular, they are R objects of class "function".

```
f <- function(<arguments>) {
     ## Do something interesting
}
```

Functions in R are "first class objects", which means that they can be treated much like any other R object. Importantly,

- Functions can be passed as arguments to other functions
- Functions can be nested, so that you can define a function inside of another function
- The return value of a function is the last expression in the function body to be evaluated.

Defining a Functions

```
f <- function(a, b = 1, c = 2, d = NULL) {
}
```

In addition to not specifying a default value, you can also set an argument value to NULL.

Input and Output

- Getting data into R
  - Type it in (small data)
  - Read from data file
  - Read from Database
- Get data out of R
  - Save in a workspace
  - Write a text file
  - Save an object to a file system (saving plots works as well)

Getting Data In R

- R supports multiple file formats
  - `read.table()` is the main function
- File name can be a URL
  - `read.table("http://ahost/file.csv", sep=",")` is the same as `read.csv(...)`
- Can read directly from a database via ODBC interface
  - `mydb <- odbcConnect("MyPostgresDB", ...)`
- R packages exist to read data from Hadoop or HDFS (more later)

Note! R always uses the forward-slash ("/") character in full file names
"C:/users/janedoe/My Documents/Script.R"

Getting Data Out R

Options	R Code

Save it as part of your workspace (or a different workspace)	`save.image(file="dfm.Rdata")`
	`save.image()  # a .Rdata file`
	`load.image("dfm.Rdata")`

Save it as a data file	`write.csv(dfm, file="dfm.csv")`

Save it as an R object	`save(UCBAdmissions,`
	`  file="UCBadmin.Rdata")`
	`load(file="UCBadmin.Rdata")`

Plots can be saved as images	`saveplot(filename="`*`filename.ext`*`",` `  type="`*`type`*`")`

Descriptive Statistics

Function	R Code
View the data	head(x); tail(x)
View a summary of the data	summary(x)
Compute basic statistics	sd(x); var(x); range(x); IQR(x)
Correlation	cor(x); cor(d$var1, d$var2)

Generic Functions

Specific actions that differ based on the class of the object:

Code	Function
Plot the variable x	plot (x)
Histogram of x	hist (x)
Internal structure of x	str (x)