R Programming
for Quantitative Finance

Guy Yollin
Applied Mathematics
University of Washington
Outline

1. R Packages
2. Overview of the R language
3. R Syntax
4. R objects
Lecture references

J. Adler.
*R in a Nutshell: A Desktop Quick Reference.*
- Chapters 4-7

W. N. Venables and D. M. Smith.
*An Introduction to R.*
2013.
- Sections 3-6, 9, 13
Outline

1. R Packages
2. Overview of the R language
3. R Syntax
4. R objects
All R functions are stored in packages

The standard R distribution includes core packages and recommended packages:
- Core R packages
  - base, utils, stats, methods, graphics, grDevices, datasets
- Recommended packages
  - boot, rpart, foreign, MASS, cluster, Matrix, etc.
- Additional packages can be downloaded through the R GUI or via the `install.packages` function

When R is initially loaded, only core R packages are loaded by default
- Additional packages are loaded via the `library` command
- Packages datasets are made accessible via the `data` command
The `args` function displays the argument names and corresponding default values of a function.

```r
args(args)
```

```r
## function (name)
## NULL
```

Main arguments:

- **name** name of the function

Return value:

- the argument list of a function

- Note, passing the appropriate function name can be non-trivial for *generic* functions
The library function

The library function is used to load add-on packages

```r
args(library)
```

### function (package, help, pos = 2, lib.loc = NULL, character.only = FALSE,
###     logical.return = FALSE, warn.conflicts = TRUE, quietly = FALSE,
###     verbose = getOption("verbose"))
### NULL

Main arguments:

**package**  name of the package to load

- Note, you must load a library before you can call functions from the library
Loading an add-on package

```r
# Default packages to load at startup
getOption("defaultPackages")

## [1] "datasets" "utils" "grDevices" "graphics" "stats"

# Current packages loaded
.packages()

## [1] "stringr" "tools" "knitr" "stats" "graphics" "grDevices"
## [7] "utils" "datasets" "base"

# load library
library(nutshell)

# Current packages loaded
.packages()

## [1] "nutshell" "nutshell.audioscrobbler"
## [3] "nutshell.bbdb" "stringr"
## [5] "tools" "knitr"
## [7] "stats" "graphics"
## [9] "grDevices" "utils"
## [11] "datasets" "base"
```
The data function

The data function loads specified data sets, or lists the available data sets.

```r
args(data)
```

```r
## function (... , list = character() , package = NULL , lib.loc = NULL ,
##    verbose =getOption("verbose"), envir = .GlobalEnv)
## NULL
```

```r
data(top.bacon.searching.cities)
# alternative: data(top.bacon.searching.cities, package="nutshell")
top.bacon.searching.cities[1,]
```

```r
##     city rank
## 1 Seattle 100
```

Main arguments:

- `...` name of the data sets to load
- `package` character vector of the name of the data set’s package (required if package is not loaded)
The install.packages function can be used to install add-on packages

```r
args(install.packages)
```

Main arguments:

- **pkgs**: character vector of the packages to be installed
- **repos**: character vector giving the name of the repository to use
- **dependencies**: logical indicating if dependencies should be installed
Installing contributed packages

Examples of calling `install.packages`:

- Default repository or prompt if no default set
- Specifying a CRAN repository
- Specifying the R-Forge repository

```r
# using default repository or will get prompted if no default repo set
install.packages("nutshell")

# specifying a CRAN repository
install.packages("forecast", repos="http://cran.fhcrc.org", depend=TRUE)

# or if repository needs to be specified
install.packages("quantstrat", repos="http://R-Forge.R-project.org")
```
Installing contributed packages with the RGui

> x <- 1:50
> nrow(x)
NULL
> help(str)
> str(constants)
  Named num [1:4] 3.14 2.72 1.41 1.62
  - attr(*, "names")= chr [1:4] "pi" "euler" "sqrt2" "golden"
> str(tab)
  'data.frame': 3 obs. of 2 variables:
  $ store: Factor w/ 3 levels "airport","downtown",...: 2 3 1
  $ sales: num 32 17 24
> summary(constants)
  Min. 1st Qu.  Median    Mean 3rd Qu.   Max.
    1.141   1.567   2.168   2.223  2.824  3.142
> dim(constants)
NULL
> dim(tab)
[1] 3 2
> length(constants)
[1] 4
> length(tab)
[1] 2
> length(m)
[1] 9
> m
  [,1]     [,2]     [,3]
[1,] -0.4017547 -0.3146403 -0.04649634
[2,] -0.4864167  1.3425015  0.80295826
[3,] -0.1029465 -1.5566827 -0.92187629
Installing contributed packages with RStudio

Convert time series data to an OHLC series

Description
Convert an OHLC or univariate object to a specified periodicity lower than the given data object. For example, convert a daily series to a monthly series, or a monthly series to a yearly one, or a one minute series to an hourly series.

The result will contain the open and close for the given period, as well as the maximum and minimum over the new period, reflected in the new high and low, respectively.

```
x <- 2
y <- 3
z <- x*y
# w <- n*m
constants <- c(3.1416, 2.7183, 1.4142, 1.6180)
write(x-constants, file="vector.dat", sep="\t")
print("all done!")
```
The following R add-on packages are recommended for computational finance:

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoo</td>
<td>Time series objects</td>
</tr>
<tr>
<td>xts</td>
<td>Extensible time series</td>
</tr>
<tr>
<td>MASS</td>
<td>Companion to Modern Applied Statistics with S</td>
</tr>
<tr>
<td>tseries</td>
<td>Time series analysis and computational finance</td>
</tr>
<tr>
<td>forecast</td>
<td>Forecasting functions for time series and linear models</td>
</tr>
<tr>
<td>PerformanceAnalytics</td>
<td>Performance and risk analysis</td>
</tr>
<tr>
<td>quantmod</td>
<td>Quantitative financial modeling framework</td>
</tr>
</tbody>
</table>
Outline

1. R Packages
2. Overview of the R language
3. R Syntax
4. R objects
R expressions are processed via R’s Read-eval-print loop †:

The Read-Evaluate-Print Loop (REPL) for R

†http://en.wikipedia.org/wiki/Read-eval-print_loop
Assigning values to variables

Like other programming languages, values can be stored in variables

- Variables are typically assigned in 1 of 3 ways:
  - assignment operator: <-
  - assignment function: assign
  - equal sign: =
    - must be used to assign arguments in a function call

```
y <- 5
y
## [1] 5
assign("e", 2.7183)
e
## [1] 2.7183
s = sqrt(2)
s
## [1] 1.4142136
r <- rnorm(n=2)
r
## [1] -0.10904366 0.55681691
s*e+y
## [1] 8.8442567
```
R makes extensive use of functions†

- Functions can be defined to take zero or more arguments
- Functions typically return a value
  - a return value is not required
- Functions are called by name with any arguments enclosed in parentheses
  - even if the function has no arguments the parentheses are required

†http://en.wikipedia.org/wiki/Functional_programming
Objects and classes

Everything in R is an object and all objects have a class

```r
constants <- c(3.1416, 2.7183, 1.4142, 1.6180)
class(constants)

## [1] "numeric"

cities <- c("Seattle", "Portland", "San Francisco")
class(cities)

## [1] "character"

m <- matrix(rnorm(9), nrow = 3)
class(m)

## [1] "matrix"

tab <- data.frame(store = c("downtown", "eastside", "airport"), sales = c(32, 17, 24))
class(tab)

## [1] "data.frame"

class(sin)

## [1] "function"
```
Data types

All R objects have a *type* or *storage mode*

- Use function `typeof` to display an object’s type

- Common types are:
  - double
  - character
  - list
  - integer

```r
typeof(constants)
## [1] "double"

typeof(m)
## [1] "double"

typeof(cities)
## [1] "character"

typeof(tab)
## [1] "list"

typeof(1:4)
## [1] "integer"
```
Weak versus strong typing

R is a weakly typed language as opposed to a strongly typed language†

Specifically, R allows mixing of different variable types and mostly coerces them to a common type without a problem:*  

- Logical values are converted to numbers: TRUE is converted to 1 and FALSE to 0
- Values are converted to the simplest type required to represent all information  
  - logical < integer < numeric < complex < character < list
- Object attributes are dropped when an object is coerced from one type to another


*see R in a Nutshell
```r
x <- c(2.5, 3.1, -7.2, "5.9", "-9.6")
x

## [1] "2.5" "3.1" "-7.2" "5.9" "-9.6"

y <- x > 2
y

## [1] TRUE TRUE FALSE TRUE FALSE

sum(y)

## [1] 3

class(1:5)

## [1] "integer"

(1:5)/2

## [1] 0.5 1.0 1.5 2.0 2.5
```
In addition to characters, integer values, and numeric values, R also uses a few special values

**NA** represents a missing value, stands for "Not Available"

**NULL** represents an empty object (i.e. an empty set)

**Inf** numbers larger (or smaller) than the largest value R can represent (about \(2^{1023}\))

**NaN** NaN means "Not a Number"
Special values

\[
\log(0)
\]

## [1] -Inf

\[
1/0
\]

## [1] Inf

\[
x <- \text{as.numeric}(c("1","2","A"))
\]

\[
x
\]

## [1] 1 2 NA

\[
\text{names}(x)
\]

## NULL

\[
\text{sqrt}(-1)
\]

## [1] NaN
Copying in assignment and function calls

Object are copied when they are assigned and when they are passed to a function\(^\dagger\). Hence, neither of these operations can affect the original values

```r
a <- 2
b <- a
b <- 3
b
## [1] 3
a
## [1] 2

squareIt <- function(x) { x <- x^2 ; print(x) }
squareIt(a)
## [1] 4
a
## [1] 2

\[^\dagger\]http://en.wikipedia.org/wiki/Call_by_value
```

\(^\dagger\)\(\text{http://en.wikipedia.org/wiki/Call\_by\_value}\)
There is no special *constant* variable type in R

Numbers are interpreted literally and default to double-precision floating point numbers

```r
class(2)
## [1] "numeric"

class(as.integer(2))
## [1] "integer"

class(2L)
## [1] "integer"

class(2:4)
## [1] "integer"
```
Outline

1. R Packages
2. Overview of the R language
3. R Syntax
4. R objects
Symbols

Variable names (i.e. symbols) should conform to the following guidelines:

- Symbols should start with a character
- Symbols can contain characters, numbers, periods ‘.’, underscores ‘_’
- Symbols must not be an R reserved word
- Symbols should not be the name of a common R function
  - c - combine function
  - t - transpose function
  - T - also interpreted as TRUE
  - F - also interpreted as FALSE
## R operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:: :: ::</td>
<td>access variables in a namespace (external/internal)</td>
</tr>
<tr>
<td>$ @</td>
<td>component / slot extraction</td>
</tr>
<tr>
<td>[ [ [</td>
<td>indexing</td>
</tr>
<tr>
<td>^</td>
<td>exponentiation (right to left)</td>
</tr>
<tr>
<td>- +</td>
<td>unary minus and plus</td>
</tr>
<tr>
<td>:</td>
<td>sequence operator</td>
</tr>
<tr>
<td>%any%</td>
<td>special operators (including %% and %/%)</td>
</tr>
<tr>
<td>* /</td>
<td>multiply, divide</td>
</tr>
<tr>
<td>+ -</td>
<td>(binary) add, subtract</td>
</tr>
<tr>
<td>&lt; &gt; &lt;= &gt;= == !=</td>
<td>ordering and comparison</td>
</tr>
<tr>
<td>!</td>
<td>negation</td>
</tr>
<tr>
<td>&amp; &amp;&amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Logical operators

- The `&` and `|` operators perform element-wise AND and OR
- The `&&` and `||` operators perform first-element AND and OR
  - use this form in `if` statements

```r
l1 <- c(TRUE, TRUE, TRUE)
l2 <- c(TRUE, FALSE, TRUE)
l3 <- c(FALSE, FALSE, TRUE)
l1 & l2
## [1] TRUE FALSE TRUE
l1 && l2
## [1] TRUE
l1 & l3
## [1] FALSE FALSE TRUE
l1 && l3
## [1] FALSE
```
Matrix multiplication

- The operation $a \ast b$ performs element-wise multiplication
- The operation $a \%\%\% b$ performs matrix multiplication

```r
(x <- 1:4)
## [1] 1 2 3 4

(y <- 5:8)
## [1] 5 6 7 8

x * y
## [1]  5 12 21 32

x %*% y
## [,1]
## [1,]   70

R will try to automatically coerce objects into conformal matrices when matrix multiplication is requested
Additional points about R language syntax

- R is case-sensitive
- vector/matrix/array indexes begin at 1
- Expressions are separated with new lines or with a semicolon
- Character strings should be double (or single) quoted
  - quoted strings within quoted strings require mixing double and single quotes
- Parentheses around an expression returns the result of the evaluation
- Curly braces are used to evaluate a series of expressions and return the last expression evaluated
  - e.g. function blocks, if blocks
R supports if and if/else condition statements as follows:

if (condition) true_expression

if (condition) true_expression else false_expression

```r
if( 2 > 4 ) print("2 > 4")
if( 2 > 4 ) print("2 > 4") else print("2 < 4")

## [1] "2 < 4"

if( 2 < 4 ) print("2 < 4")

## [1] "2 < 4"

if( 2 < 4 ) print("2 < 4") else print("2 > 4")

## [1] "2 < 4"
```
Conditional statements

```r
if( 2 < 4 ) {
  print("it's true that:")
  print("2 < 4")
}

## [1] "it's true that:"
## [1] "2 < 4"

if( 2 > 4 ) {
  print("is this true:")
  print("2 > 4")
} else {
  # note, the right brace must be in front of the else
  print("its not true:")
  print("2 > 4")
}

## [1] "its not true:"
## [1] "2 > 4"

- Note, else can not start a line
The `ifelse` function performs a vectorized `if/else` operation.

```r
a <- c("a","a","a","a","a")
b <- c("b","b","b","b","b")
ifelse(c(TRUE,FALSE,TRUE,FALSE,TRUE),a,b)
```

```r
# [1] "a" "b" "a" "b" "a"
```

```r
(x <- c(0.5, -0.6, -0.9, 0.4, 0.6, 0.8, 0.5, 0.0, -0.8, -0.2))
```

```r
# [1] 0.5 -0.6 -0.9 0.4 0.6 0.8 0.5 0.0 -0.8 -0.2
```

```r
barplot(x,col=ifelse(x>0,"red","blue"))
```

![Barplot](image)
The for loop

```r
for (var in list) expression

for ( i in 1:3 ) {
  print(i^2)
}
```

```r
## [1] 1
## [1] 4
## [1] 9
```

```r
mat.list <- list(matrix(NA,nrow=2,ncol=3), matrix(NA,nrow=4,ncol=2),
                 matrix(NA,nrow=5,ncol=5))
for ( mat in mat.list ) {
  print(dim(mat))
}
```

```r
## [1] 2 3
## [1] 4 2
## [1] 5 5
```

- First example uses a standard loop counter
- Second example iterates through a list of matrices
The while loop

A while loop uses a conditional test to continue rather than a loop counter as in the case of a for loop:

```
while (condition) expression
```

```r
n <- 1
while ( n < 10 ) {
  print(n)
  n <- n * 2
}

## [1] 1
## [1] 2
## [1] 4
## [1] 8
```

- Use "while" rather then "for" when you don’t know how many iterations are needed
  - e.g. reading incrementally from a database or from a file
The repeat loop creates an infinite loop that requires a break statement to terminate

```
repeat expression
```

counter <- 0
t1 <- Sys.time()
repeat {
  t2 <- Sys.time()
  elapsed <- as.numeric(t2-t1)
  if ( elapsed > 0.01 )
    break;
  counter <- counter + 1
}
print(counter)

## [1] 76
Outline

1. R Packages
2. Overview of the R language
3. R Syntax
4. R objects
Vectors

R is a vector/matrix programming language (also known as an array programming language†)

- vectors can easily be created with `c`, the combine function
- most places where a single value can be supplied, a vector can be supplied and R will perform a vectorized operation

```r
my.vector <- c(2, 4, 3, 7, 10)
my.vector

# [1] 2 4 3 7 10

my.vector^2

# [1] 4 16 9 49 100

sqrt(my.vector)

# [1] 1.4142136 2.0000000 1.7320508 2.6457513 3.1622777
```

The matrix object

A matrix is a 2-dimension object where all elements are of the same class.

```r
args(matrix)
```

```
## function (data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
## NULL
```

Main arguments:

- `data`: data to initialize matrix
- `nrow`: number of rows
- `ncol`: number of columns
- `byrow`: apply data across rows rather then down columns (default)
- `dimnames`: names for the columns and rows

Return value:

- a matrix object
Creating matrices

```r
sales <- matrix(c(58, 64, 53, 55), nrow=2,
                 dimnames=list(c("Q1", "Q2"), c("East Side", "Downtown"))

sales

## East Side Downtown
## Q1 58 53
## Q2 64 55

matrix(1:4, ncol=2)

## [,1] [,2]
## [1,] 1 3
## [2,] 2 4

matrix(1:4, ncol=2, byrow=T)

## [,1] [,2]
## [1,] 1 2
## [2,] 3 4

matrix(ncol=3, nrow=2)

## [,1] [,2] [,3]
## [1,] NA NA NA
## [2,] NA NA NA
```
The array object

An array is a 3-dimensional or greater object where all elements are of the same class

```
args(array)
```

```r
## function (data = NA, dim = length(data), dimnames = NULL)
## NULL
```

Main arguments:

- **data** data to initialize the array
- **dim** a vector specifying the length of each dimension
- **dimnames** names for the columns and rows

Return value:

array object
Creating an array

arr <- array(0, dim=c(3,4,2))
arr

## , , 1
## [1,] 0 0 0 0
## [2,] 0 0 0 0
## [3,] 0 0 0 0

## , , 2
## [1,] 0 0 0 0
## [2,] 0 0 0 0
## [3,] 0 0 0 0
Array/matrix/vector relationship

class(arr)

## [1] "array"

dim(arr)

## [1] 3 4 2

class(arr[,1])

## [1] "matrix"

dim(arr[,1])

## [1] 3 4

class(arr[,1,1])

## [1] "numeric"

length(arr[,1,1])

## [1] 3
The list object

A list object is a container that can hold other objects of different types

```r
myList <- list(pi=3.1416,euler=2.7183,golden=1.6180)
class(myList)

## [1] "list"

length(myList)

## [1] 3

myList

## $pi
## [1] 3.1416
##
## $euler
## [1] 2.7183
##
## $golden
## [1] 1.618

diverseList <- list(magic=myList,random=matrix(rnorm(4),ncol=2),
                   state=c("WA","OR"))
```
The list object

A list object is a container that can hold other objects of different types.

```r
length(diverseList)
# [1] 3

class(diverseList[[1]])
# [1] "list"

class(diverseList[[2]])
# [1] "matrix"

class(diverseList[[3]])
# [1] "character"
```
Items in a list can be accessed using `[]`, `[[ ]]`, or `$` syntax as follows:

- `[[ ]]` returns a single element
  - single integer
  - single name
- `$` returns a single element
  - single name
- `[]` returns a sublist
  - vector of positive integers
  - vector of named items
  - logical vector

```r
myList[[2]]
## [1] 2.7183

myList[["pi"]]
## [1] 3.1416

myList$golden
## [1] 1.618

myList[2]
## $euler
## [1] 2.7183

class(myList[2])
## [1] "list"
```
The `data.frame` object

A `data.frame` is a 2D matrix-like object where the columns can be of different classes.

For example, a single `data.frame` object may have the following composition:

- columns with dates
- columns with characters
- columns with integer values
- columns with numeric values

```r
data(batting.2008)
class(batting.2008)
## [1] "data.frame"
dim(batting.2008)
## [1] 1384 32

batting.2008[1,1:4]
## nameLast nameFirst weight height
## 1 Abreu Bobby 200 72
class(batting.2008[,"nameLast"])
## [1] "character"
class(batting.2008[,"weight"])
## [1] "integer"
class(batting.2008[,"height"])
## [1] "numeric"
```
The head and tail functions

```r
args(getS3method("head","data.frame"))

## function (x, n = 6L, ...)  
## NULL

data(dow30)
head(dow30)

## symbol Date Open High Low Close Volume Adj.Close
## 1  MMM 2009-09-21 73.91 74.68 73.91 74.54 2560400 74.54
## 2  MMM 2009-09-18 75.12 75.25 74.50 74.62 4387900 74.62
## 3  MMM 2009-09-17 75.34 75.45 74.50 74.89 3371500 74.89
## 4  MMM 2009-09-16 74.76 75.49 74.50 75.38 2722500 75.38
## 5  MMM 2009-09-15 74.63 74.88 74.00 74.68 3566900 74.68
## 6  MMM 2009-09-14 73.72 74.64 73.42 74.56 3466400 74.56

tail(dow30,3)

## symbol Date Open High Low Close Volume Adj.Close
## 7480 DIS 2008-09-24 32.59 32.59 31.63 31.77 13600300 31.30
## 7481 DIS 2008-09-23 32.88 33.32 32.15 32.53 13450900 32.05
## 7482 DIS 2008-09-22 33.85 34.05 32.84 32.91 18394300 32.42
```
Size-related and diagnostic helper functions

R has a number of size related and diagnostic helper functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>length</code></td>
<td>length a vector or list</td>
</tr>
<tr>
<td><code>dim</code></td>
<td>return dimensions of a multidimensional object</td>
</tr>
<tr>
<td><code>nrow</code></td>
<td>number of rows of a multidimensional object</td>
</tr>
<tr>
<td><code>ncol</code></td>
<td>number of columns of a multidimensional object</td>
</tr>
<tr>
<td><code>head</code></td>
<td>display first n rows (elements)</td>
</tr>
<tr>
<td><code>tail</code></td>
<td>display last n rows (elements)</td>
</tr>
<tr>
<td><code>str</code></td>
<td>summarize structure of an object</td>
</tr>
</tbody>
</table>
Indexing data.frames and matrices

R has extremely powerful data manipulation capabilities especially in the area of vector and matrix indexing

- data.frames and matrices can be indexed in any of the following ways
  - vector of positive integers
  - vector of negative integers
  - character vector of columns (row) names
  - a logical vector (selected if TRUE, not selected if FALSE)
- Since data.frames are stored internally as lists, their columns can be accessed with the $ operator as well
Indexing data.frames and matrices

dow30[1:4,c("symbol","Date","Close")]

<table>
<thead>
<tr>
<th>symbol</th>
<th>Date</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MMM 2009-09-21</td>
<td>74.54</td>
</tr>
<tr>
<td>2</td>
<td>MMM 2009-09-18</td>
<td>74.62</td>
</tr>
<tr>
<td>3</td>
<td>MMM 2009-09-17</td>
<td>74.89</td>
</tr>
<tr>
<td>4</td>
<td>MMM 2009-09-16</td>
<td>75.38</td>
</tr>
</tbody>
</table>

head(dow30[-1,c(1,2,6)],4)

<table>
<thead>
<tr>
<th>symbol</th>
<th>Date</th>
<th>Open</th>
<th>High</th>
<th>Low</th>
<th>Close</th>
<th>Volume</th>
<th>Adj.Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>MMM 2009-09-18</td>
<td>69.15</td>
<td>69.68</td>
<td>69.00</td>
<td>69.57</td>
<td>17984000</td>
<td>69.57</td>
</tr>
<tr>
<td>3</td>
<td>MMM 2009-09-17</td>
<td>69.95</td>
<td>70.45</td>
<td>69.80</td>
<td>69.84</td>
<td>20986000</td>
<td>69.84</td>
</tr>
<tr>
<td>4</td>
<td>MMM 2009-09-16</td>
<td>69.86</td>
<td>70.38</td>
<td>69.18</td>
<td>70.34</td>
<td>22150400</td>
<td>70.34</td>
</tr>
<tr>
<td>5</td>
<td>MMM 2009-09-15</td>
<td>70.27</td>
<td>70.48</td>
<td>69.68</td>
<td>69.99</td>
<td>43247900</td>
<td>69.99</td>
</tr>
</tbody>
</table>

head(dow30[dow30$symbol=="XOM",],4)

<table>
<thead>
<tr>
<th>symbol</th>
<th>Date</th>
<th>Open</th>
<th>High</th>
<th>Low</th>
<th>Close</th>
<th>Volume</th>
<th>Adj.Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOM</td>
<td>2009-09-21</td>
<td>69.15</td>
<td>69.68</td>
<td>69.00</td>
<td>69.57</td>
<td>17984000</td>
<td>69.57</td>
</tr>
<tr>
<td>XOM</td>
<td>2009-09-18</td>
<td>70.27</td>
<td>70.48</td>
<td>69.68</td>
<td>69.99</td>
<td>43247900</td>
<td>69.99</td>
</tr>
<tr>
<td>XOM</td>
<td>2009-09-17</td>
<td>69.95</td>
<td>70.45</td>
<td>69.80</td>
<td>69.84</td>
<td>20986000</td>
<td>69.84</td>
</tr>
<tr>
<td>XOM</td>
<td>2009-09-16</td>
<td>69.86</td>
<td>70.38</td>
<td>69.18</td>
<td>70.34</td>
<td>22150400</td>
<td>70.34</td>
</tr>
</tbody>
</table>
### Indexing data.frames and matrices

```r
dow30[dow30[, "Volume"] > 1.5e9, ]
```

<table>
<thead>
<tr>
<th></th>
<th>symbol</th>
<th>Date</th>
<th>Open</th>
<th>High</th>
<th>Low</th>
<th>Close</th>
<th>Volume</th>
<th>Adj.Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>2047</td>
<td>C</td>
<td>2009-08-07</td>
<td>3.98</td>
<td>4.24</td>
<td>3.83</td>
<td>3.85</td>
<td>1898814600</td>
<td>3.85</td>
</tr>
<tr>
<td>2049</td>
<td>C</td>
<td>2009-08-05</td>
<td>3.32</td>
<td>3.61</td>
<td>3.29</td>
<td>3.58</td>
<td>2672492000</td>
<td>3.58</td>
</tr>
<tr>
<td>2159</td>
<td>C</td>
<td>2009-02-27</td>
<td>1.56</td>
<td>1.93</td>
<td>1.40</td>
<td>1.50</td>
<td>1868209400</td>
<td>1.50</td>
</tr>
</tbody>
</table>

```r
dow30[dow30$Volume > 1.5e9, ]
```

<table>
<thead>
<tr>
<th></th>
<th>symbol</th>
<th>Date</th>
<th>Open</th>
<th>High</th>
<th>Low</th>
<th>Close</th>
<th>Volume</th>
<th>Adj.Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>2047</td>
<td>C</td>
<td>2009-08-07</td>
<td>3.98</td>
<td>4.24</td>
<td>3.83</td>
<td>3.85</td>
<td>1898814600</td>
<td>3.85</td>
</tr>
<tr>
<td>2049</td>
<td>C</td>
<td>2009-08-05</td>
<td>3.32</td>
<td>3.61</td>
<td>3.29</td>
<td>3.58</td>
<td>2672492000</td>
<td>3.58</td>
</tr>
<tr>
<td>2159</td>
<td>C</td>
<td>2009-02-27</td>
<td>1.56</td>
<td>1.93</td>
<td>1.40</td>
<td>1.50</td>
<td>1868209400</td>
<td>1.50</td>
</tr>
</tbody>
</table>

```r
which(dow30$Volume > 1.5e9)
```

|   | [1] 2047 2049 2159 |

```r
dow30[which(dow30$Volume > 1.5e9), ]
```

<table>
<thead>
<tr>
<th></th>
<th>symbol</th>
<th>Date</th>
<th>Open</th>
<th>High</th>
<th>Low</th>
<th>Close</th>
<th>Volume</th>
<th>Adj.Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>2047</td>
<td>C</td>
<td>2009-08-07</td>
<td>3.98</td>
<td>4.24</td>
<td>3.83</td>
<td>3.85</td>
<td>1898814600</td>
<td>3.85</td>
</tr>
<tr>
<td>2049</td>
<td>C</td>
<td>2009-08-05</td>
<td>3.32</td>
<td>3.61</td>
<td>3.29</td>
<td>3.58</td>
<td>2672492000</td>
<td>3.58</td>
</tr>
<tr>
<td>2159</td>
<td>C</td>
<td>2009-02-27</td>
<td>1.56</td>
<td>1.93</td>
<td>1.40</td>
<td>1.50</td>
<td>1868209400</td>
<td>1.50</td>
</tr>
</tbody>
</table>
A factor is a data type for representing categorical data

```r
pet.str <- c("dog","cat","cat","dog","fish","dog","rabbit")
pets <- as.factor(pet.str)
pets

## [1] dog  cat  cat  dog  fish  dog  rabbit
## Levels: cat  dog  fish  rabbit

as.numeric(pets)

## [1] 2 1 1 2 3 2 4

levels(pets)

## [1] "cat" "dog" "fish" "rabbit"
```

- factors are encoded as integers (starting at 1)
- the levels of a factor variable contain the categorical labels
Attributes

Objects can have attributes that indicate what the object represents and how it should be interpreted by R

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>Class of an object</td>
</tr>
<tr>
<td>comment</td>
<td>Comment of an object</td>
</tr>
<tr>
<td>dim</td>
<td>Dimensions of the object</td>
</tr>
<tr>
<td>dimnames</td>
<td>Names associated with each dimension</td>
</tr>
<tr>
<td>names</td>
<td>Names attribute of an object</td>
</tr>
<tr>
<td>row.names</td>
<td>Name of each row</td>
</tr>
<tr>
<td>tsp</td>
<td>Time series attribute of an object</td>
</tr>
<tr>
<td>levels</td>
<td>Levels of a factor</td>
</tr>
</tbody>
</table>

- An object’s class is implemented as an attribute
The attributes function

The attributes function displays an object’s attributes

```
sales

## East Side Downtown
## Q1 58 53
## Q2 64 55

attributes(sales)

## $dim
## [1] 2 2
##
## $dimnames
## $dimnames[[1]]
## [1] "Q1" "Q2"
##
## $dimnames[[2]]
## [1] "East Side" "Downtown"
```

- For some simple objects, the class is implicit and not listed by the attributes function
The `unclass` function returns an object with its class attribute removed.