

Data Science for Business

Lecture #1

Primer on R

Prof. Alan L. Montgomery

The University of Hong Kong & Carnegie Mellon University, Tepper School of Business

email: alanmontgomery@cmu.edu

All Rights Reserved, © 2020 Alan Montgomery

Do not distribute, post, or reproduce without Alan Montgomery's Permission

Basics of R

Installing R Studio

The first step is to install R. We will install a precompiled binary distribution. R is a fairly large package (60mB download and 120mB needed for installation). We will get it from Rstudio:

<https://cran.rstudio.com/>

If you are using Windows:

<https://cran.rstudio.com/bin/windows/base/R-4.0.2-win.exe>

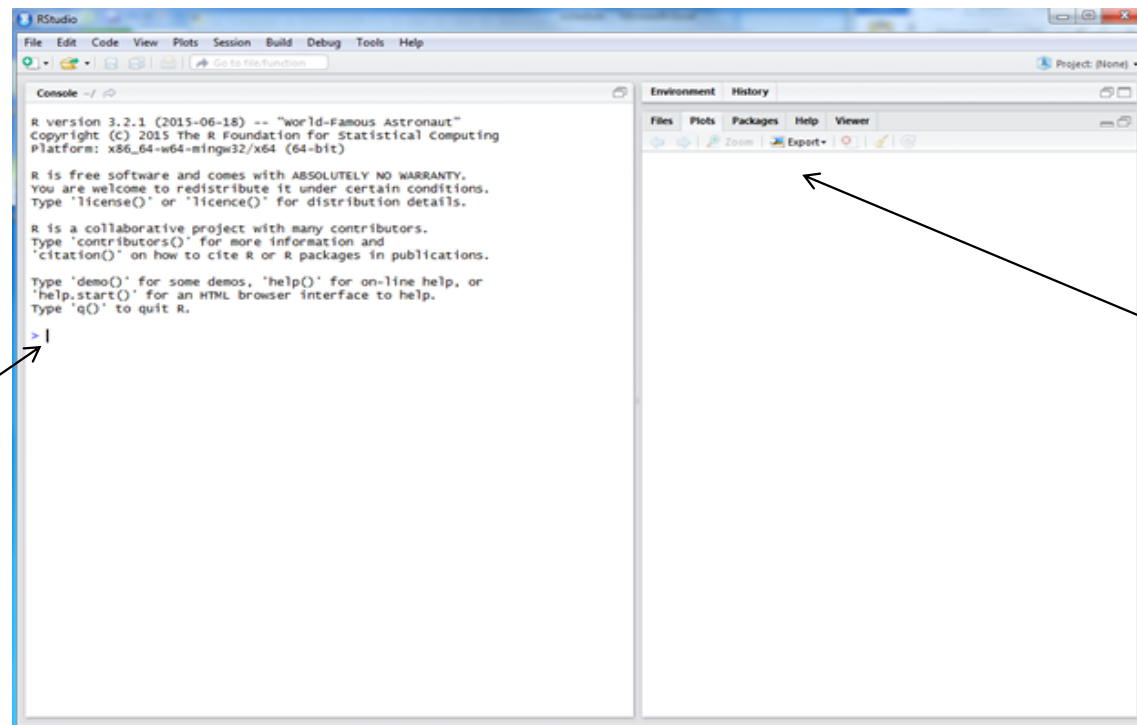
If you are using Mac:

<https://cran.rstudio.com/bin/macosx/R-4.0.2.pkg>

Sometime the previous version is more stable, so you may want to use R-3.6.3

RStudio

Console



Output
Windows

Understanding R

Everything in R is either an object or an expression (function) that uses objects

R stores data as Objects in memory. Objects can be single values, vectors, matrices, code, graphs, models, ...

Expressions (or functions) return objects that may be used for manipulating other objects

Expressions are entered at the “>” prompt, and by default prints the object:

```
> 3+3
[1] 6
> sin(pi)
[1] 1.224606e-016
> sqrt(100)
[1] 10
```

We assign objects using “<-” or “=”

```
> x=3+3
> print(x)
[1] 6
```

The R Language

Expressions are entered at the prompt:
“>”

R evaluates the expression and prints
out the result

```
> 3+3
[1] 6
> sin(pi)
[1] 1.224606e-016
> sqrt(100)
[1] 10
```

Additional prompt

An incomplete expression leads to a second prompt: “+”

Can continue at the second prompt

```
> sqrt(  
+ 100)  
[1] 10
```

Getting stuck at “+” prompt

If the “+” prompt continues after hitting return, then enter many “)” to get the “>” prompt

Then start your expression again

```
> sqrt(  
+  
+ ))))
```

```
Error in parse(text = txt): Syntax  
error: No opening parenthesis,  
before ")" at this point:
```

```
sqrt(  
))
```

```
Dumped
```

```
> sqrt(100)
```


Scalars and Assignments

Read this as “weight” gets 190

This assigns the value 190 to the scalar named weight

The assignment operator is the sequence of characters: “<-” or “=”

```
> weight=190  
> weight  
[1] 190
```

Character Assignments

Character values are inserted in quotes

If the quotes are omitted, R will look for a data object called, Jim , to assign to person

The result is not printed until you enter the object name

```
> person="Jim"  
> person  
[1] "Jim"
```

Vectors

The function: `rnorm()`, returns a vector of random deviates from the normal distribution

The `[n]` on the left shows where the row starts

```
> rnorm(10)
[1]  0.3037020
[2] -0.5248669
[3]  1.4674553
[4]  0.4536315
[5]  0.4077797
[6]  0.5362221
[7]  0.0759569
[8]  0.3239556
[9] -1.3531665
[10] -2.4226150
```

Vectors (cont.)

A single number is a vector of length 1

We can make vectors using the concatenation function: `c()`

We assign the integers 1,2,3 to the vector `x`

```
> mean(rnorm(10))
```

```
[1] 0.5807564
```

```
> x=c(1,2,3)
```

```
> x
```

```
[1] 1 2 3
```

Vectors (cont.)

We can create a vector of names

We can create a vector of sequential integers using the function: `a:b`, where `a` is the starting integer and `b` is the ending integer

```
> people=c("Jim", "Sue", "Dave")
> people
[1] "Jim"  "Sue"  "Dave"

> 5:10
[1] 5 6 7 8 9 10
```

Object Names

Object names may contain:

Letters: abcDEF

Numbers: 0123456789

Dot: .

Examples of valid names:

height

weight

x.var

.yvar

x.y.var

x110

Object names (cont.)

Objects names cannot use an underscore, a hyphen, begin with a number, or use reserved symbols

Examples of invalid object names

`_xvar`

`y_var`

`x-yvar`

`120xvar`

`T`

`F`

`NA`

Handling Objects

We can list out all of the objects

```
> objects()
[1] ".Last.value" ".Random.seed"
[3] "Cars"        "last.dump"
[5] "last.warning" "people"
[7] "person"      "weight"
[9] "weights"     "x"
```

Objects remain until removed, even if one quits R (assuming you save your workspace)

```
> rm(x)
> x
Error: Object "x" not found
Dumped
```

Equivalently, you can use the object browser

Objects as variables

Objects can be
used in
expressions

```
> x=1:10
> mean(x)
[1] 5.5
> y=c(x,10)
> length(y)
[1] 11
> 2*y
[1] 2 4 6 8 10 12 14 16 18 20 20
```

Vector Arithmetic

Scalar Functions
work on
elementwise
basis

Can perform
scalar and vector
arithmetic

```
> x=1:5
> x^2
[1] 1 4 9 16 25

> 2*x
[1] 2 4 6 8 10
> 2*x+sqrt(x)
[1] 3.000000 5.414214
[3] 7.732051 10.000000
[5] 12.236068
```

Logical Vectors

Expressions with
relational operators
return logical vectors

T is True, F is False

```
> x=rnorm(5)
> x
[1] 1.2698616 -1.1080517
[3] 0.5627334 0.2454234
[5] 0.2919052
> x<0
[1] F T F F F
```

Missing values

A missing value is represented by NA

Operations on NA return NA

The function `is.na()` checks for missing values

```
> x=c(1, NA, 3)
> x
[1] 1 NA 3
> x+1
[1] 2 NA 4
> sum(x)
[1] NA
> is.na(x)
[1] F T F
```

Vector indexing

Use brackets, [] to
select elements of
a vector

Negative indices
remove elements

```
> x=c(2,4,6,8,10)
> x
[1] 2 4 6 8 10
> x[1]
[1] 2
> x[3:5]
[1] 6 8 10
> x[c(1,3,5)]
[1] 2 6 10
> x[-(1:3)]
[1] 8 10
```

Logical Indices

A logical index selects elements

Symbols for logical operators:

- < Less than
- > Greater than
- <= Less than or equal to
- >= Greater than or equal to
- == Equal to
- ! Negation operator
- != Not equal to

```
> x=rnorm(5)
> x
[1] -2.4592950  0.9074605
[3]  0.5088648 -1.1184415
[5]  0.5137160
> x[x<=0]
[1] -2.459295 -1.118441
> log.x=log(x)
Warning messages:
  NAs generated in: log(x)
> x[is.na(log.x)]
[1] -2.459295 -1.118441
> log.x[!is.na(log.x)]
[1] -0.09710521 -0.67557299
[3] -0.66608473
```

Replacement

You can use [] on the left hand side of an assignment, =

```
> x=sample(1:8)
> x
[1] 2 6 8 3 1 5 7 4
> x[6]=NA
> x
[1] 2 6 8 3 1 NA 7 4
> x[is.na(x)]=0
> x
[1] 2 6 8 3 1 0 7 4
```

Functions

Functions are called like this:

`function.name(argument,
argument,)`

Functions always return a
value

NULL represents no value

Example:

`seq(from=1, to=end, by=1,
length=inferred,
along=NULL)`

```
> seq(1,5)
[1] 1 2 3 4 5
> seq(10, 20, length=6)
[1] 10 12 14 16 18 20
> seq(to=100, by=15, length=7)
[1] 10 25 40 55 70 85 100
> seq(length=10)
[1] 1 2 3 4 5 6 7 8
[9] 9 10
```


Functions (cont.)

Function arguments have:

- position(first, second,)
- function name
- default values for function options (sometimes)
- Examples:
 - `rnorm(n, mean=0, sd=1)`
 - `rep(x, times=inferred, length.out=inferred)`

```
> rep(1:3,2)
[1] 1 2 3 1 2 3

> rep(1:3, length=8)
[1] 1 2 3 1 2 3 1 2

> rep(1:3, c(3,2,1))
[1] 1 1 1 2 2 3

> rep()
Error in rep: Argument "x" is missing, with
no default: rep() Dumped

> rep(1:3, c(3,2,1), 5)
[1] 1 2 3 1 2
```

Matrices

`matrix(data=NA,
nrow=inferred,
ncol=inferred, byrow=F,
dimnames=NULL)`

The function `matrix()` reads data into a matrix

The number of columns is specified by using the argument `ncol=#`

And/Or the number of rows can be specified by the argument, `nrow=#`

```
> x=matrix(1:10, nrow=2)
> x
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> dim(x)
[1] 2 5
> x.matrix=matrix(c(20,10,3,1,7,4), ncol=2)
> x.matrix
      [,1] [,2]
[1,]   20    1
[2,]   10    7
[3,]    3    4
```

Matrices (cont.)

We can attach names to columns with the `dimnames` option:

```
> x=matrix(c(1:20), ncol=4, byrow=T,  
dimnames=list(NULL, c("col1", "col2",  
"col3", "col4")))  
> x
```

	col1	col2	col3	col4
[1,]	1	2	3	4
[2,]	5	6	7	8
[3,]	9	10	11	12
[4,]	13	14	15	16
[5,]	17	18	19	20

Matrices (cont.)

Specifying
byrow=T forces R
to read the data
in row by row

When the
argument is not
specified, or
specified as
byrow=F , R
assumes the data
is written in
column by column

```
> x.matrix=matrix(c(20,10,3,1,7,4),  
  ncol=2, byrow=T)  
> x.matrix  
      [,1] [,2]  
[1,]   20  10  
[2,]    3    1  
[3,]    7    4
```

Matrices (cont.) - Indexing

To extract a value from a matrix, use two elements in the subscript

The first element applies to rows

The second element applies to columns

If one dimension is not specified, all elements for that dimension are extracted

```
> x=matrix(1:15, nrow=3, byrow=T)
> x
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
> x[2,3]
[1] 8
> x[2:3, 3:5]
      [,1] [,2] [,3]
[1,]    8    9   10
[2,]   13   14   15
> x[,1]
[1] 1 6 11
> x[1,]
[1] 1 2 3 4 5
```

Constructing Matrices from vectors

Matrices can be constructed from row vectors and column vectors using `cbind` and `rbind`

Binding together vectors of different attributes (character and numeric for example), is not allowed - vectors will be coerced to a similar attribute

Numeric and character vectors will be coerced to character

```
> x = c(3,4,5)
> y = c(6,7,8)
> x.y=cbind(x,y)
> x.y
      x y
[1,] 3 6
[2,] 4 7
[3,] 5 8
> x = c(3,4,5)
> x = c(6,7,8)
> x.y=rbind(x,y)
> x.y
  [,1] [,2] [,3]
x     3     4     5
y     6     7     8
```

Constructing Matrices from vectors (cont.)

Example: binding together a character vector and a numeric vector coerces to a character matrix

If we do not want to coerce objects then what we need is a data object called a data frame (similar to SPSS or SAS datasets)

```
> x = c(3,4,5)
> y = c("Three","Four","Five")
> x.y=rbind(x,y)
> x.y
```

	[,1]	[,2]	[,3]
x	"3"	"4"	"5"
y	"Three"	"Four"	"Five"

Converting Matrices to Data Frames

Data Frames are a data object that allows one to bind data vectors of different types together, such that the data can be accessed like a matrix

Most of the dialog boxes in the GUI operate on Data Frames

```
> x = c(3,4,5)
> y = c("Three","Four","Five")
> x.y=data.frame(x, y)
> x.y
```

	x	y
1	3	Three
2	4	Four
3	5	Five

Visualizing Data

R allows the creation of plots and graphics. R's philosophy is that graphics are objects, and they can be layered on top of one another, so you can build up complex graphics from simple primitives.

If you enter the object by itself then it will print it (or plot it) for graphics

```
> x=rnorm(100)
> graphic=hist(x)
> graphic
$breaks
 [1] -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5
$counts
 [1]  4  3  7  8 24 19 22  4  7  2
$density
 [1] 0.08 0.06 0.14 0.16 0.48 0.38 0.44 0.08 0.14 0.04
$mids
 [1] -2.25 -1.75 -1.25 -0.75 -0.25  0.25  0.75  1.25  1.75  2.25
$xname
 [1] "x"
$equidist
 [1] TRUE
attr(,"class")
 [1] "histogram"
```

Lists are ways of collecting different types of data

When you want to collect different types of data together you can use a list.

R often returns lists from functions.

To access the elements of the list use the \$ operator

```
> x=list("alan",2)
> x[[1]] # to get the first element of list
[1] "alan"
> x[[2]] # to get the second element of list
[1] 2
> x=list(name=c("alan","peter"),value=c(2,3,4,5))
> x$name # we can access the first element using the $name
[1] "alan" "peter"
> x$value
[1] 2 3 4 5
> x$name = c("alan","peter","henry")
```

Data frames are special types of lists

Data frames are special types of lists, where all the elements of the list has the same length, but not necessarily the same type

This makes it easy to mix strings, numbers, and logical values

To access a variable in your data frame use \$

```
> x=data.frame(name=c("alan","alan","john","john"),treatment=c(1,2,1,2),value=c(10,20,5,15),stringsAsFactors=FALSE)
> str(x)
'data.frame':      4 obs. of  3 variables:
 $ name      : chr  "alan" "alan" "john" "john"
 $ treatment: num   1  2  1  2
 $ value     : num  10 20  5 15
> x.names = as.factor(x$name) # factors have levels and values
> levels(x.names) # notice this gives a vector of string
[1] "alan" "john"
> as.integer(x.names) # the each values are stored as integers
that reference the levels
[1] 1 1 2 2
> levels(x.names)[1]="Alan" # by separating it makes it easier to
manipulate the levels separately
> x.names # notice that "alan" has become "Alan"
[1] Alan Alan john john
Levels: Alan john
> as.character(x.names) # we can coerce the factor to strings
[1] "Alan" "Alan" "john" "john"
```

Always follow good programming practices when writing R Scripts

Document your code

Write it modularly

Check and test your programs (before you run them)

Backup your data and scripts

Treat R scripts as programs, follow good practices, remember you are likely to have to revisit your program in the future

Learning R

“An Introduction to Data Science” by Jeffrey Stanton

<https://drive.google.com/file/d/0B6iefdnF22XQeVZDSkxjZ0Z5VUE/edit?usp=sharing>

“An Introduction to R” by Venables, Smith and the R Core Team

<http://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>

There are thousands of books on R, so just search for one that suits your interests.

More help on Learning R

References on R

Throughout the course we will illustrate data mining techniques using R-Studio. R is a statistical analysis environment, and R-Studio provides a more user friendly to R. R is currently the most popular program for data mining. Although we will not be able to exploit R to its fullest extent, we will illustrate how to analyze our cases using R-Studio. Technically there are two separate programs, R and R-Studio, however, it will appear to you as a single integrate program. It is available for both Windows and Mac computers. Please follow the installation instructions here:

<https://www.rstudio.com/products/RStudio/>

R is an open source project, which means that it is freely available. There is a lot of good documentation, and here is a suggested list.

- [An Introduction to R](#) [↗]. (Gives an introduction to the language and how to use R for doing statistical analysis and graphics.)
- [R for Beginners](#) [↗] by Emmanuel Paradis. (A short introduction to R).

There are many online resources for R. Here is sampling that students have found useful:

- UCLA has a set of tutorials and videos for learning R (as well as other software). Here is a good place to start on learning R: <http://www.ats.ucla.edu/stat/r/>
- There is also a MOOC on R-Studio at Johns Hopkins (this is a paid course though): <https://www.coursera.org/learn/data-scientists-tools>
- DataCamp (the introduction is free, but the intermediate and advanced ones are paid): <https://www.datacamp.com/courses>
- For more resources try RStudio:

Historically R traces its roots back to the S Language for which a series of excellent books by John Chambers.

- Richard A. Becker and John M. Chambers (1984), *S. An Interactive Environment for Data Analysis and Graphics*, Monterey: Wadsworth and Brooks/Cole. (Brown Book, Larger of historical interest)
- Richard A. Becker, John M. Chambers and Allan R. Wilks (1988), *The New S Language*, London: Chapman & Hall. (Blue Book, Introduced what is now known as version 2.)
- John M. Chambers and Trevor J. Hastie (1992), *Statistical Models in S*, London: Chapman & Hall. (White Book, Introduced version 3, which added structures to facilitate statistical modeling in S.)
- John M. Chambers (1998), *Programming with Data*, New York: Springer. (Green Book, Describes version 4 of S, a major revision of S designed by John Chambers to improve its usefulness at every stage of the programming process.
- John M. Chambers (2010), *Software for Data Analysis: Programming with R (Statistics and Computing)*, Springer. (Latest book that is focused on more advanced programming with R. Best after you have become a good R programmer and want to move to an expert level.)

More help on Learning R

<http://www.ats.ucla.edu/stat/r/>

UCLA Institute for Digital Research & Education

Search this website

HOME SOFTWARE ▼ RESOURCES ▼ SERVICES ▼ ABOUT US

R

- [Classes and Seminars](#)
- [Learning Modules](#)
- [Frequently Asked Questions](#)
- [Code Fragments \(Advanced\)](#)

Statistical Analyses

- [Data Analysis Examples](#)
- [Textbook Examples](#) (see also [Stat Books for Loan on R](#))
- [Downloadable Books on R](#)


Important Links

- [How can I get R? Where can I run R?](#)
- [Installing, Customizing, Updating R](#)
- [Documentation for R packages organized by topical domains](#)

Have you seen?

- [Class Notes: Introduction to R](#)
- [Step-by-step instructions](#) to analyze major public-use survey data sets with R by Anthony Damico

▼ Click here to report an error on this page or leave a comment [How to cite this page](#)

 Optimized for iPhone, Android and iPad.
No app download required!

© 2019 UC REGENTS HOME CONTACT

Data Types in R

Data Types in R

R has a few basic data types. These are the low level building blocks.

Integer (e.g., 3L, `as.integer(3)`)

Numeric (real or decimal, e.g., 2, 2.0, pi)

Logical (e.g., TRUE or FALSE)

Character (or string, e.g., "a", "able is five")

Complex (e.g., 1+0i, 1+4i)

There are a couple of special values: NULL and NA

Data Structures in R

Data structures provide ways of organizing many values

Vector (one dimension, all elements have one type)

Matrix (two dimensions, all elements have one type)

Arrays (multiple dimensions, all elements have one type)

Lists (collection of objects, each element can be of any type)

Data Frames (a list of equal-length vectors)

Factors (a vector that can only take predefined values)

Relationship among Data Structures

	Homogeneous	Heterogeneous
1-dimension	Atomic vector	List
2-dimension	Matrix	Data frame
N-dimension	Array	

Useful operators to understand data type

>typeof() # what is it?

>class() # what is it?

>storage.mode() # what is it?

>length() # how long is it? (for one dimensional objects, use dim for multiple dimensions)

>attributes() # does it have any metadata?

>str() # structure of an object

Understand R object types

R has three popular classes of objects

S3

- This is the default, and the most common. Lists are objects.
- You access an S3 object's elements using \$

S4

- Requires initialization, so more format, strict and verbose.
- Better for programming in teams.
- Access slots using @
- Example: Use setClass to define your class

R6

- Much more like Python or C++. It is built on encapsulated objects rather than generic functions.
- Functions can change slot values. In other words objects are passed by reference not by value.
- Access methods using \$

Formulas in R

Formula Notation in R

R functions like `lm()` or `ggplots()` use a special syntax that is helpful for defining models or selecting variables. The basic format is:

`response ~ predictor`

For example you might see:

`fit = lm(Y~X)`

Technically you are not specifying a model but creating a set of variables to include in the function

Understanding R formulas: response ~ predictor

Symbol	Example	Meaning
+	+X	include this variable
-	-X	delete this variable
:	X:Z	include the interaction between these variables
*	X*Y	include these variables and the interactions between them
	X Z	conditioning: include x given z
^	(X + Z + W) ^ 3	include these variables and all interactions up to three way
I	I (X*Z)	as is: include a new variable consisting of these variables multiplied
1	X - 1	intercept: delete the intercept (regress through the origin)

There is usually more than one way to specify the same model; the notation is not unique.
For example the following three formulae are all equivalent:

$Y \sim X + Z + W + X:Z + X:W + Z:W + X:Z:W$

$Y \sim X * Z * W$

$Y \sim (X + Z + W) ^ 3$

each corresponding to the model

$$Y_i = \beta_0 + X_i\beta_1 + Z_i\beta_2 + W_i\beta_3 + X_iZ_i\beta_4 + X_iW_i\beta_5 + Z_iW_i\beta_6 + X_iZ_iW_i\beta_7 + \epsilon_i.$$