Professor Bear - Intro to R

Nik Bear Brown

September 19, 2016

Table of Contents

1.1: Data types in R

R has a wide variety of data types including numerical, character, logical, complex, factors, dates and special objects to handle missing and special values.

It also has functions like is.numeric(), is.character(), etc. to check a data type as well as functions like as.numeric(), as.character() that will attempt to convert an object to a particular data type.

1.1.1: Numeric

A numeric data type is a floating point approximatation of a real number including positive and negative numbers. R has no single precision data type. All real numbers are stored in double precision format. Double precision has 53 bits, and represents to that precision a range of absolute values from about 2e-308 to 2e+308. 2e+308 is a VERY big number so one will use it for most numeric operations without worrying whether R can accurately store the number.

```
x <- c(0.0,1,2,3.3,-5.5) # numeric vector
x

## [1]  0.0  1.0  2.0  3.3 -5.5

class(x)

## [1] "numeric"

y <-5
y

## [1] 5
```

```
class(y)
```

```
## [1] "numeric"
```

```
z<--3.3
z
```

```
## [1] -3.3
```

```
class(z)
```

```
## [1] "numeric"
```

Note: By default all numbers are real numbers. In order to create an integer variable in R, we can invoke the as.integer function or use the suffix L after the number.

```
y <-5
y
```

```
## [1] 5
```

```
class(y)
```

```
## [1] "numeric"
```

```
y <-5.6
y<-as.integer(y)
class(y)
```

```
## [1] "integer"
```

```
y <-5
z<-as.integer(3)
z
```

```
## [1] 3
```

```
class(z)
```

```
## [1] "integer"
```

```
a<-1
a
```

```
## [1] 1
```

```
class(a)
```

```
## [1] "numeric"
```

```
b<-1L
b
```

```
## [1] 1
```

```
class(b)
```

```
## [1] "integer"
```

1.1.2: Logical

The logical (or Boolean) data type is a data type, having two values (usually denoted true and false), intended to represent the truth values of logic and Boolean algebra.

```
x<-3.3; y<-5   # some values
z = x > y      # is x larger than y?
z              # print the logical value
```

```
## [1] FALSE
```

```
class(z)       # print the class name of z
```

```
## [1] "logical"
```

```
u = TRUE; v = FALSE
u & v          # u AND v
```

```
## [1] FALSE
```

```
u | v          # u OR v
```

```
## [1] TRUE
```

```
!u             # negation of u
```

```
## [1] FALSE
```

Notice that u = TRUE; v = FALSE are capitalized. Further details and related logical operations can be found in the R documentation.

```
 help("&")
```

1.1.3: Complex

A complex number is a pair of numbers $a \quad and \quad b$ forming a vector representing a projection in the real and the complex plane. "Re" is the real axis, "Im" is the imaginary axis, and i is the [[imaginary unit]] which satisfies $i^2 = -1$.

```
x <- 1 + 2i     # create a complex number
x
```

```
## [1] 1+2i
```

```
class(x)
```

```
## [1] "complex"
```

The following gives an error as −1 is not a complex value.

```
sqrt(-1)       # square root of -1
```

Instead, we have to use the complex value $-1 + 0i$.

```
sqrt(-1+0i)       # square root of -1
```

Alternatively we can coerce -1 into a complex value.

```
sqrt(as.complex(-1))  # square root of -1
```

1.1.4: Character and strings

Character string are sequences of ASCII characters. By default R uses a UTF-8 encoding. UTF-8 is a character encoding capable of encoding all possible characters, or code points, defined by Unicode and originally designed by Ken Thompson and Rob Pike. Professor Bear will create another tutorial on using other character encodnings in R.

```
x<-'bear'
x

## [1] "bear"

class(x)

## [1] "character"

y<-as.character(3.14)
y

## [1] "3.14"

class(y)

## [1] "character"
```

1.1.5: Factors

It is common to have categorical data in statistical data analysis. A categorical variable is a variable that can take on one of a limited, and usually fixed, number of possible values (e.g. {Male,Female} or {yes,no} or a human blood type {A, B, AB or O}. In R such variables are referred to as factors. Nominal variables such as yes/no has no intrinsic ordering. An ordinal variable has a clear ordering of the variables, such as ranks like *private*, *corporal*,*sergeant*, *lieutenant*, *captain*, etc.. In R, *all categorical data* is treated as a factor with a set of levels. The levels allow for the ordering (or reordering of categorical data) for display even if there is no intrinsic ordering.

A factor is very similar to an integer vector with a set of labels. While factors look like character vectors, they are not. So be careful when converting factors to characters and vice-versa. For example, use `stringsAsFactors = FALSE` when reading dataframes (more on this later).

```
months =
c("March","April","January","November","September","October","June","August",
"February","May","July","December")
months
```

```
##  [1] "March"     "April"     "January"   "November"  "September"
##  [6] "October"   "June"      "August"    "February"  "May"
## [11] "July"      "December"
```

```
class(months)
```

```
## [1] "character"
```

```
months = factor(months,levels=c("January","February","March",
"April","May","June","July","August","September","October","November","Decemb
er"),ordered=TRUE)
class(months)
```

```
## [1] "ordered" "factor"
```

```
levels(months)
```

```
##  [1] "January"   "February"  "March"     "April"     "May"
##  [6] "June"      "July"      "August"    "September" "October"
## [11] "November"  "December"
```

```
is.factor(months)
```

```
## [1] TRUE
```

1.1.6: Dates

Dates are represented as the number of days since 1970-01-01, with negative values for earlier dates. We use as.Date( ) to convert strings to dates.

```
life <- as.Date(c("1972-07-08", "2016-09-05")) # use as.Date( ) to convert
strings to dates
lifespan <- life[2] - life[1]
lifespan
```

```
## Time difference of 16130 days
```

```
Sys.Date( ) # returns today's date.
```

```
## [1] "2018-09-09"
```

```
date() # returns the current date and time.
```

```
## [1] "Sun Sep  9 21:36:59 2018"
```

1.1.7: Missing and special values

One thing you may also want to do is to see if there are any **missing values**. For that we can use the `is.na()` function. Missing values in R are coded as NA (NA stands for "not available."). The code below, for example, asks for NA values for the variable

```
x<-1:33 # create a numeric sequence
x

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33

x[sample(1:length(x), 11)] <- NA # Add 11 NA's randomly
x

##  [1]  1  2  3  4 NA  6  7  8  9 NA 11 NA NA 14 15 NA NA NA 19 20 21 22 23
## [24] 24 25 NA 27 28 NA 30 31 NA NA

is.na(x) # Create a logical vector asking which elements are NA's

##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
## [12]  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE
```

Inf and -Inf

If a computation results in a number that is too big, R will return Inf for a positive number and -Inf for a negative number (meaning positive and negative infinity, respectively).

```
2^3333

## [1] Inf

-2^3333

## [1] -Inf

1/0

## [1] Inf
```

NaN

A computation will Sometimes produce a result that makes no sense. In these cases, R will often return NaN (meaning "not a number").

```
Inf - Inf

## [1] NaN

0 / 0

## [1] NaN
```

NULL

It is common for programming languages to have a NULL value. NULL is used to represent missing or undefined values. NULL represents the null object in R: it is a reserved word. NULL is often returned by expressions and functions whose values are undefined.

While this is similar to NA, rhere is a sublte but important distinction between NA and NULL. NA is a logical constant of length 1 which contains a missing value indicator. The important distinction is that NA is a 'logical' value that when evaluated in an expression, yields NA. Porofessor Bear will create a future tutorial on the difference but that requires an explanation of pointers, so for the moment just remember to use NA for missing data values.

```
NA
```

```
## [1] NA
```

```
class(NA)
```

```
## [1] "logical"
```

```
NA > 1
```

```
## [1] NA
```

```
NULL
```

```
## NULL
```

```
class(NULL)
```

```
## [1] "NULL"
```

```
NULL > 1
```

```
## logical(0)
```

1.17: Further resources

LearnR

[Try R @codeschool](http://tryr.codeschool.com)

Datacamp R Tutorials

rstudio online learning