R Basics

36-600

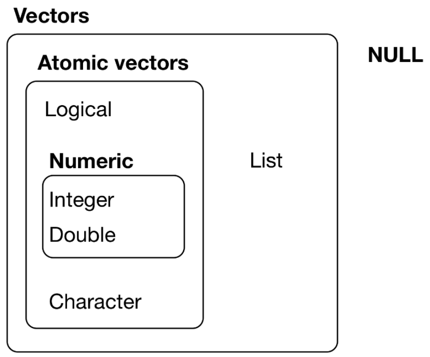
Week 1 Thursday – Fall 2021

Motivation

A *vector* in `R` is a homogeneous collection of numbers, strings, or `TRUE`/`FALSE` values (i.e., logicals).

A collection of (column) vectors, all of the same length, can be bound together into a data table (called a *data frame*) that might provide the input to, e.g., a regression analysis. So in our discussion of statistical learning, it makes sense to start with a discussion of `R` vectors and how you might use (and manipulate!) them.

Vectors in R



Atomic vectors are *homogeneous*, i.e., all elements of the vectors are of the same type. The important types of atomic vectors (or just vectors) for our purposes are:

- `double`: double-precision floating-point numbers (8 bytes per element);
- `integer`: integer numbers (4 bytes per element);
- `logical`: `TRUE` and `FALSE`; and
- `character`: individual strings (at 1 byte per individual character within each string).

Note: the `integer` and `double` types are collectively (and at times confusingly) dubbed `numeric`. (Confusingly because one can cast to a numeric type, which is equivalent to casting to double.)

My advice: unless memory storage is an issue, just forget that `R` supports integer vectors, and assume that all the numerical variables you will deal with will be contained in double-precision `numerical` vectors.

Initializing Vectors

Let's show the various ways in which one can initialize a vector of five integers:

Function Call	
`x <- c(0,0,0,0,0)`	c = "collection" or "container"
`x <- rep(0,5)`	rep = "repeat"
`x <- vector("numeric",5)`	
`x <- numeric(5)`	
`x <- seq(1,5,by=1)`	seq = "sequence"
`x <- 1:5`	steps by 1

We can use all six of these functions to initialize `numeric` vectors, and the first four to initialize those of mode `logical` or `character`, as with, e.g.,

```
vector("logical",5)
```

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

```
character(5)
```

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

Note that in my own life, I almost always use `rep()` (e.g., `rep("",100)` or `rep(TRUE,200)`, etc.).

Initializing Vectors

A few more points to make here:

- You can combine initialization functions, which can be helpful:

```
x <- c(rep(0,5),11:14,numeric(3))
x
```

```
##  [1]  0  0  0  0  0 11 12 13 14  0  0  0
```

- You can concatenate vectors too:

```
x <- 1:3
y <- 78:83
(z <- append(x,y))
```

```
## [1]  1  2  3 78 79 80 81 82 83
```

(Why the parentheses? It's an `R` trick: you can assign to a new variable *and* print its contents in one line of code.)

- Note that the assignment operator `<-`. An equals sign, `=`, works too, but purists use `<-` and thus we'll start purely. (You will see me slip up throughout the semester and type `=` because I learned `R` on my own long ago and wasn't indoctrinated in the use of `<-`.)

Handy Vector Functions

To determine the type of a vector:

```
x <- c(1,0,3,2)
typeof(x)
```

```
## [1] "double"
```

To determine the number of elements in a vector:

```
length(x)
```

```
## [1] 4
```

To display the n^{th} element of a vector, where $n \in [1, length(x)]$:

```
x[1]
```

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

To explicitly cast from one type to another:

```
as.character(x)
```

```
## [1] "1" "0" "3" "2"
```

Handy Vector Functions

To display multiple vector elements, pass in a defined vector:

```
x[c(1,4)]
```

```
## [1] 1 2
```

```
x[1:2]
```

```
## [1] 1 0
```

To *remove* a vector element (from output…not from the vector itself!), utilize the minus sign:

```
x[-2]
```

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

To remove multiple vector elements, pass in a minus sign and a defined vector:

```
x[-c(1,4)]
```

```
## [1] 0 3
```

```
x[-(1:2)]  # without the parentheses, R thinks you mean -1 to 2
```

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

Handy Vector Functions

To sort a vector in ascending order, and to retrieve the sorted vector indices:

```
x
```

```
## [1] 1 0 3 2
```

```
sort(x)
```

```
## [1] 0 1 2 3
```

```
order(x)
```

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

To display the unique values of a vector:

```
unique(x)
```

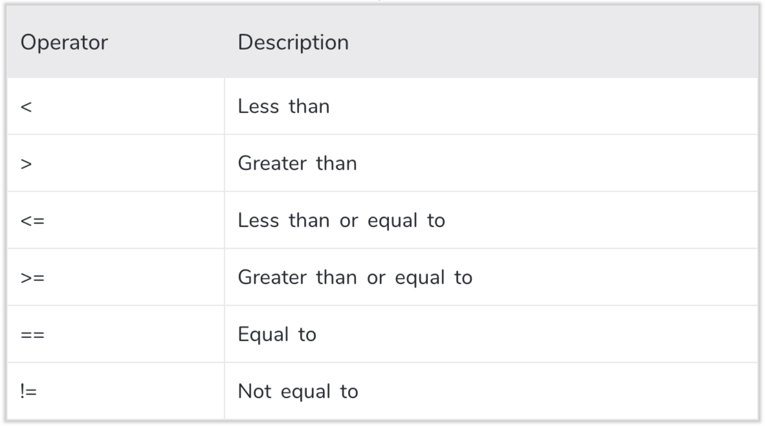
```
## [1] 1 0 3 2
```

```
table(x)
```

```
## x
## 0 1 2 3
## 1 1 1 1
```

Logical Subsetting

Relational Operators in R



Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

If you apply a relational operator to a vector, the output will be a logical vector:

```
set.seed(101)
x <- rnorm(10)
x>0
```

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

Logical Subsetting

¡MUY IMPORTANTE! If you apply a logical vector of length n to a vector of length n, then *only the elements of the second vector associated with the value TRUE will be displayed!* For instance:

```
x
```

```
##  [1] -0.3260365  0.5524619 -0.6749438  0.2143595  0.3107692  1.1739663  0.6187899 -0.
```

```
x[x>0]
```

```
## [1] 0.5524619 0.2143595 0.3107692 1.1739663 0.6187899 0.9170283
```

Logical Subsetting

The output from relational operators can be combined using the logical `and` operator (&) or the logical `or` operator (|):

```
y <- x>0 & x<0.5
x[y]
```

```
## [1] 0.2143595 0.3107692
```

```
y <- x<0 | x>0.5
x[y]
```

```
## [1] -0.3260365  0.5524619 -0.6749438  1.1739663  0.6187899 -0.1127343  0.9170283 -0.2
```

Logical Subsetting: sum()

To determine how many values in your vector satisfy a condition, combine one or more relational operators with the `sum()` function:

```
sum(x>-0.5 & x<0)
```

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

Logical Subsetting: which()

To determine which elements of the original vector satisfy a condition, combine one or more relational operators with the `which()` function:

```
which(x>-0.5 & x<0)
```

```
## [1]  1  8 10
```

Another means by which to subset a vector is to apply the output of the `which()` function. Note how adding a minus sign changes the output!

```
w <- which(x<0)
x[w]
```

```
## [1] -0.3260365 -0.6749438 -0.1127343 -0.2232594
```

```
x[-w]
```

```
## [1] 0.5524619 0.2143595 0.3107692 1.1739663 0.6187899 0.9170283
```

Missing Data: NA

`NA` means "Not Available" and is the preferred way in `R` to denote missing data.

To determine whether vector elements are `NA`, we can use the `is.na()` function, which returns a logical vector.

```
x <- c(1,NA,3)
is.na(x)
```

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