Subsetting Vectors

Raphael Carvalho

20/05/2019

Subsetting Vectors

The way you tell R that you want to select some particular elements (i.e. a ‘subset’) from a vector is by placing an ‘index vector’ in square brackets immediately following the name of the vector. For a simple example, try x[1:10] to view the first ten elements of x.

```
x[1:10]
```

```
##  [1]           NA           NA  1.042252547           NA           NA
##  [6] -0.092668232 -0.500333319 -1.404759367           NA  0.008206288
```

What do you think x[is.na(x)] will give you?

```
print("2: A vector of all NAs")
```

```
## [1] "2: A vector of all NAs"
```

Recall that ! gives us the negation of a logical expression, so !is.na(x) can be read as ‘is not NA’. Therefore, if we want to create a vector called y that contains all of the non-NA values from x, we can use y <- x[!is.na(x)]. Give it a try.

```
y <- x[!is.na(x)]
```

Recall that the expression y > 0 will give us a vector of logical values the same length as y, with TRUEs corresponding to values of y that are greater than zero and FALSEs corresponding to values of y that are less than or equal to zero. What do you think y[y > 0] will give you?

```
print("1: A vector of all the positive elements of y")
```

```
## [1] "1: A vector of all the positive elements of y"
```

You might wonder why we didn’t just start with x[x > 0] to isolate the positive elements of x. Try that now to see why.

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

```
##  [1]          NA          NA 1.042252547          NA          NA
##  [6]          NA 0.008206288          NA          NA          NA
## [11]          NA          NA          NA 0.278735977          NA
## [16]          NA          NA          NA 1.610933847 0.369899514
## [21]          NA 0.043526480          NA          NA          NA
## [26]          NA 0.428470483 0.846031063
```

Since NA is not a value, but rather a placeholder for an unknown quantity, the expression NA > 0 evaluates to NA. Hence we get a bunch of NAs mixed in with our positive numbers when we do this.

Combining our knowledge of logical operators with our new knowledge of subsetting, we could do this – x[!is.na(x) & x > 0]. Try it out.

```
x[!is.na(x) & x > 0]
```

```
## [1] 1.042252547 0.008206288 0.278735977 1.610933847 0.369899514 0.043526480
## [7] 0.428470483 0.846031063
```

Can you figure out how we'd subset the 3rd, 5th, and 7th elements of x? Hint – Use the c() function to specify the element numbers as a numeric vector.

```
x[c(3, 5, 7)]
```

```
## [1]  1.0422525         NA -0.5003333
```

It's important that when using integer vectors to subset our vector x, we stick with the set of indexes {1, 2, . . . , 40} since x only has 40 elements. What happens if we ask for the zeroth element of x (i.e. x[0])? Give it a try.

```
x[0]
```

```
## numeric(0)
```

As you might expect, we get nothing useful. Unfortunately, R doesn't prevent us from doing this. What if we ask for the 3000th element of x? Try it out.

```
x[3000]
```

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

Luckily, R accepts negative integer indexes. Whereas x[c(2, 10)] gives us ONLY the 2nd and 10th elements of x, x[c(-2, -10)] gives us all elements of x EXCEPT for the 2nd and 10 elements. Try x[c(-2, -10)] now to see this.

```
x[c(-2,-10)]
```

```
##  [1]          NA  1.04225255          NA          NA -0.09266823
##  [6] -0.50033332 -1.40475937          NA -0.33069581 -1.24412832
## [11]          NA          NA          NA -0.40215828          NA
## [16]          NA -1.49451239 -0.22967941          NA  0.27873598
## [21] -1.52801554          NA          NA          NA -1.52026283
## [26]          NA  1.61093385  0.36989951          NA  0.04352648
## [31]          NA          NA -0.05411356 -1.53893110          NA
## [36]          NA  0.42847048  0.84603106
```

A shorthand way of specifying multiple negative numbers is to put the negative sign out in front of the vector of positive numbers. Type x[-c(2, 10)] to get the exact same result.

```
x[-c(2,10)]
```

```
##  [1]          NA  1.04225255          NA          NA -0.09266823
##  [6] -0.50033332 -1.40475937          NA -0.33069581 -1.24412832
## [11]          NA          NA          NA -0.40215828          NA
## [16]          NA -1.49451239 -0.22967941          NA  0.27873598
## [21] -1.52801554          NA          NA          NA -1.52026283
## [26]          NA  1.61093385  0.36989951          NA  0.04352648
## [31]          NA          NA -0.05411356 -1.53893110          NA
## [36]          NA  0.42847048  0.84603106
```

So far, we've covered three types of index vectors – logical, positive integer, and negative integer. The only remaining type requires us to introduce the concept of 'named' elements. Create a numeric vector with three named elements using vect <- c(foo = 11, bar = 2, norf = NA).

```
vect <- c(foo = 11, bar = 2, norf = NA)
```

We can also get the names of vect by passing vect as an argument to the names() function. Give that a try.

```
names(vect)
```

```
## [1] "foo"  "bar"  "norf"
```

Alternatively, we can create an unnamed vector vect2 with c(11, 2, NA). Do that now.

```
vect2 <- c(11, 2, NA)
```

Then, we can add the `names` attribute to vect2 after the fact with names(vect2) <- c("foo", "bar", "norf"). Go ahead.

```
names(vect2) <- c("foo", "bar", "norf")
```

Now, let's check that vect and vect2 are the same by passing them as arguments to the identical() function.

```
identical(vect, vect2)
```

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

Now, back to the matter of subsetting a vector by named elements. Which of the following commands do you think would give us the second element of vect?

```
print("3: vect['bar']")
```

```
## [1] "3: vect['bar']"
```

Likewise, we can specify a vector of names with vect[c("foo", "bar")]. Try it out.

```
vect[c("foo", "bar")]
```

```
## foo bar
##  11   2
```