6. Subsetting Vectors

Solutions to Swirl's R Programming Exercises

06-25-2022

```
library(swirl)
```

```
##
## | Hi! Type swirl() when you are ready to begin.
```

Acknowledgements: R Language Concepts and code questions (with minor modifications) are used here from the swirl package.https://www.r-project.org/nosvn/pandoc/swirl.html

Type ls() to see a list of the variables in your workspace. Then, type rm(list=ls()) to clear your workspace. Type swirl() when you are ready to begin.

```
     rm(list=ls())
```

In this lesson, we'll see how to extract elements from a vector based on some conditions that we specify.

For example, we may only be interested in the first 20 elements of a vector, or only the elements that are not NA, or only those that are positive or correspond to a specific variable of interest. By the end of this lesson, you'll know how to handle each of these scenarios.

I've created for you a vector called x that contains a random ordering of 20 numbers (from a standard normal distribution) and 20 NAs. Type x now to see what it looks like.

```
x <- sample(c(1:20))
```

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] 15  9  7  8  4  6  5 14 20 18
```

Index vectors come in four different flavors – logical vectors, vectors of positive integers, vectors of negative integers, and vectors of character strings – each of which we'll cover in this lesson.

Let's start by indexing with logical vectors. One common scenario when working with real-world data is that we want to extract all elements of a vector that are not NA (i.e. missing data). Recall that is.na(x) yields a vector of logical values the same length as x, with TRUEs corresponding to NA values in x and FALSEs corresponding to non-NA values in x.

Remember that is.na(x) tells us where the NAs are in a vector. So if we subset x based on that, what do you expect toh happen?

```
1: A vector of all NAs

2: A vector of TRUEs and FALSEs

3: A vector of length 0

4: A vector with no NAs

Selection: 1
```

```
     x[is.na(x)]
```

```
## integer(0)
```

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)]
```

Print y to the console.

```
y
```

```
##  [1] 15  9  7  8  4  6  5 14 20 18 13 11 10 17 19 12  3  2 16  1
```

Now that we've isolated the non-missing values of x and put them in y, we can subset y as we please. 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?

```
1: A vector of all the negative elements of y

2: A vector of all the positive elements of y

3: A vector of length 0

4: A vector of TRUEs and FALSEs

5: A vector of all NAs

Selection: 2
```

Type $y[y > 0]$ to see that we get all of the positive elements of y, which are also the positive elements of our original vector x.

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] 15  9  7  8  4  6  5 14 20 18 13 11 10 17 19 12  3  2 16  1
```

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] 15  9  7  8  4  6  5 14 20 18 13 11 10 17 19 12  3  2 16  1
```

In this case, we request only values of x that are both non-missing AND greater than zero.

I've already shown you how to subset just the first ten values of x using x[1:10]. In this case, we're providing a vector of positive integers inside of the square brackets, which tells R to return only the elements of x numbered 1 through 10.

Many programming languages use what's called 'zero-based indexing', which means that the first element of a vector is considered element 0. R uses 'one-based indexing', which (you guessed it!) means the first element of a vector is considered element 1.

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] 7 4 5
```

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]
```

```
## integer(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
```

Again, nothing useful, but R doesn't prevent us from asking for it. This should be a cautionary tale. You should lways make sure that what you are asking for is within the bounds of the vector you're working with.

What if we're interested in all elements of x EXCEPT the 2nd and 10th? It would be pretty tedious to construct a vector containing all numbers 1 through 40 EXCEPT 2 and 10.

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] 15  7  8  4  6  5 14 20 13 11 10 17 19 12  3  2 16  1
```

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)
```

When we print vect to the console, you'll see that each element has a name. Try it out.

```
vect
```

```
##  foo  bar norf
##   11    2   NA
```

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

```
vect2 <- setNames(vect2, names(vect))
vect2
```

```
##  foo  bar norf
##   11    2   NA
```

One more time. You can do it! Or, type info() for more options.

Add names to vect2 with names(vect2) <- c("foo", "bar", "norf").

```
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
```

Indeed, vect and vect2 are identical named vectors.

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?

If we want the element named "bar" (i.e. the second element of vect), which command would get us that?

```
1: vect["2"]

2: vect[bar]

3: vect["bar"]

Selection: 3
```