Video 7: Subsetting Atomic Vectors

Stats 102A

Miles Chen, PhD

Related Reading

Advanced R: Chapter 4 - Subsetting

https://adv-r.hadley.nz/subsetting.html

Subsetting Atomic Vectors

Subsetting Atomic vectors

Let’s explore the different types of subsetting with a simple vector, x.

```
x <- c(2.1, 4.2, 3.3, 5.4)
```

We start with a simple vector x, that has been crafted so that the number after the decimal point gives the original position in the vector.

There are a few ways you to subset a vector:

Subsetting with Positive Integers

Positive integers return elements at the specified positions:

```
# x <- c(2.1, 4.2, 3.3, 5.4)
x[c(3, 1)]
```

```
[1] 3.3 2.1
```

```
order(x)
```

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

```
x[order(x)]
```

```
[1] 2.1 3.3 4.2 5.4
```

Subsetting with Positive Integers

```r
# x <- c(2.1, 4.2, 3.3, 5.4)
# Duplicated indices yield duplicated values
x[c(1, 1)]
```

```
[1] 2.1 2.1
```

```r
# Real numbers are silently truncated to integers
x[c(2.1, 2.9)]
```

```
[1] 4.2 4.2
```

Subsetting with Negative Integers

Negative integers omit elements at the specified positions:

```r
# x <- c(2.1, 4.2, 3.3, 5.4)
x[-c(3, 1)]
```

```
[1] 4.2 5.4
```

You can’t mix positive and negative integers in a single subset:

```r
x[c(-1, 2)]
```

```
Error in x[c(-1, 2)]: only 0's may be mixed with negative subscripts
```

Subsetting with Logical vectors

Logical vectors select elements where the corresponding logical value is TRUE. This is probably the most useful type of subsetting because you write the expression that creates the logical vector:

```
# x <- c(2.1, 4.2, 3.3, 5.4)
x[c(TRUE, TRUE, FALSE, FALSE)]
```

```
[1] 2.1 4.2
```

```
x[x > 3]
```

```
[1] 4.2 3.3 5.4
```

Subsetting with Logical vectors

If the logical vector is shorter than the vector being subsetted, it will be *recycled* to be the same length.

```
# x <- c(2.1, 4.2, 3.3, 5.4)
x[c(TRUE, FALSE)]
```

```
[1] 2.1 3.3
```

```
# The above is equivalent to x[c(TRUE, FALSE, TRUE, FALSE)]
```

A missing value in the index always yields a missing value in the output:

```
x[c(TRUE, TRUE, NA, FALSE)]
```

```
[1] 2.1 4.2  NA
```

```
x[c(TRUE, NA)] # recycling rules still apply
```

```
[1] 2.1  NA 3.3  NA
```

Special Cases

Nothing returns the original vector. This is not useful for vectors but is very useful for matrices, data frames, and arrays. It can also be useful in conjunction with assignment.

```r
x[]
```

```
[1] 2.1 4.2 3.3 5.4
```

Zero returns a zero-length vector. This is not something you usually do on purpose, but it can be helpful for generating test data.

```r
x[0]
```

```
numeric(0)
```

Subsetting with Character vectors

If the vector is named, you can also use **Character vectors** to return elements with matching names.

```r
(y <- setNames(x, letters[1:4]))
```

```
  a   b   c   d 
2.1 4.2 3.3 5.4 
```

```r
y[c("d", "c", "a")]
```

```
  d   c   a 
5.4 3.3 2.1 
```

Subsetting with Character vectors

```r
# Like integer indices, you can repeat indices
y[c("a", "a", "a")]
```

```
  a   a   a 
2.1 2.1 2.1 
```

```r
# When subsetting with [ names must be spelled exactly to find a ma
z <- c(abc = 1, def = 2)
z[c("a", "d")]
```

```
<NA> <NA> 
  NA   NA 
```

Useful application: Lookup tables (character subsetting)

Character matching provides a powerful way to make lookup tables.

```r
x <- c("m", "f", "u", "f", "f", "m", "m")
lookup <- c(m = "Male", f = "Female", u = NA)
lookup[x]  # subset the labeled vector with the vector of abbreviat
```

```
       m        f        u        f        f        m        m 
  "Male" "Female"       NA "Female" "Female"   "Male"   "Male" 
```

```r
# we can clean up the resulting vector by removing names
unname(lookup[x])
```

```
[1] "Male"   "Female" NA       "Female" "Female" "Male"   "Male"  
```