Week 3 Module 3: Vector Indexing

CSCI E-5a: Introduction to R

Let's clear the global computing environment:

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

Let's load in the objects for this module from the R object file:

```
load( "Module 3 R Objects.Rdata")
```

Module Overview and Learning Objectives

Hello! And welcome to Module 3: Indexing Vectors.

In this module, we'll learn about vector indexing, which allows us to select elements from a vector.

- In section 1, we'll discuss the four types of vector indexing methods.
- In section 2, we'll learn how to index a vector by using positive integers.
- In section 3, we'll learn how to index a vector by using negative integers.

Once you've completed this module, you'll be able to:

- Explain the concept of vector indexing, and list the four types of indexing methods.
- Index a vector by using positive integers.
- Index a vector by using negative integers.

There are no new built-in R functions in this module.

All right! Let's get started by discussing the four types of vector indexing methods.

Section 1: Vector Indexing Principles

Main Idea: *There are four types of vector indexing methods*

In this section, we'll discuss the four types of vector indexing methods.

Vector *indexing*, also called *subsetting*, is one of the most important techniques in R, and it serves as the basis for many operations.

The basic idea of indexing is to take a vector as the input argument and return a subset of that vector.

You can think of this as selecting certain elements from the vector, or equivalently as filtering other elements out.

There are different ways of specifying this procedure, and these methods allow us to extract a variety of kinds of information from the vector.

There are 4 different ways that we can index a vector:

- We can use positive integers.
- We can use negative integers.
- We can use character strings.
- We can use logical values.

In this module, we'll focus on the first two methods, and we'll learn how to index using character values in Module 5 of this week.

We'll see how to index a vector using logical values in Week 5.

So those are the four different types of vector indexing methods.

Now let's learn how to index a vector by using positive integers.

Section 2: Indexing with Positive Integers

Main Idea: *We can index a vector by using positive integers*

In this section, we'll learn how to index a vector by using positive integers.

The first method of vector indexing uses a positive integer to specify the element to be selected.

This is a very intuitive approach – the positive integer represents the location of the value that we want to select, and the values at the other locations are ignored.

It's pretty much the only method used in many standard programming languages.

This approach is useful if the number of elements that you want to be included in the subset is small, and you know the exact locations of the included elements.

Let's examine the `numeric.vector` that we loaded in from the R Object file:

```
numeric.vector
```

```
## [1] 4 7 9 2 3
```

For instance, suppose we want to select the first element of `numeric.vector`.

To do this, we use a special notation:

- First, we write the name of the vector that we want to index.
- Next, we have an opening square bracket '['.
- Then we have the positive integer that represents the location we want to select.
- Finally, we have a closing square bracket ']'.

Let's see this in action:

```
numeric.vector[ 1 ]
```

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

If we use positive integer indexing on the right-hand side of an assignment operation, R will select the specified element from the vector, and will create a new object consisting of those values.

We can store this new object in a variable:

```
x <- numeric.vector[ 1 ]

x
```

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

When we do this, the original vector is unchanged:

```
numeric.vector
```

```
## [1] 4 7 9 2 3
```

In fact, since the values are stored in a new object, we can modify them without altering the original vector:

```
x <- -3

numeric.vector
```

```
## [1] 4 7 9 2 3
```

However, when we use positive integer indexing on the left-hand side of an assignment operator, the expression denotes the actual location in the vector, and if we assign a value to this location then it will modify the original:

```
numeric.vector[ 1 ] <- -3

numeric.vector
```

```
## [1] -3  7  9  2  3
```

You can see that now we really have changed this vector.

In fact, instead of just using one positive integer value, we can pass an entire vector of positive integer values:

```
numeric.vector[ c(2, 4, 5) ]
```

```
## [1] 7 2 3
```

Again, if you use positive integer indexing with multiple positive integers on the right-hand side of an assignment operator, then R will create a new vector consisting of those values:

```
y <- numeric.vector[ c(2, 4, 5) ]

y
```

```
## [1] 7 2 3
```

On the other hand, if we use positive integer indexing with multiple positive integers on the left-hand side of an assignment operator, then R will interpret that expression as referring to the locations in the original vector, so if we assign a value to them then we are actually destructively modifying the original.

Let's make a test vector consisting of the value 0 repeated 10 times:

```
test.vector <- rep( x = 0, times = 10 )
```

Now we'll assign the value 5 to the locations 1, 3, and 7, and then display this vector directly:

```
test.vector[ c(1, 3, 7) ] <- 5

test.vector
```

```
##  [1] 5 0 5 0 0 0 5 0 0 0
```

It's important to realize that if we want to use positive integer indexing with multiple values, then these integer values have to be stored in a vector.

If you forget to put the integer values in a vector and instead just try to list them separated by commas, you'll get an error:

```
numeric.vector[ 1, 2, 4 ]
```

```
## Error in numeric.vector[1, 2, 4]: incorrect number of dimensions
```

We can construct the indexing vector however we like, as long it consists entirely of positive integers.

For instance, to select the second through fourth elements of `numeric.vector`, we can use the colon operator:

```
numeric.vector[ 2:4 ]
```

```
## [1] 7 9 2
```

What happens if you use an index that is too big?

That is, what happens if one of the indexing values is greater than the length of the vector?

R will not issue an error, but will instead simply return the value `NA`:

```
numeric.vector[ 10 ]
```

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

We'll explore this `NA` value in Week 5.

Don't do this – as a general rule, if you're writing code that is trying to index a vector using a value that is too large, that's an indication that something is wrong with your program, and you should fix it so that it *never* does this.

So that's how to index a vector by using positive integers.

Now let's see how to index a vector by using negative integers.

Exercise 3.1: Indexing with positive integers

Let's construct a simple vector:

```
exercise.vector <-
    c( 6, 3, 2, 8, 6, 4, 1, 5, 9, 0)
```

Part (a)

Select the fourth element of this vector.

Solution

Part (b)

Select the elements in the second, fifth, and seventh positions.

Solution

Part (c)

Select the fourth through the eighth elements of the vector.

Solution

Section 3: Indexing with Negative Integers

Main Idea: *We can index a vector by using negative integers*

In this section, we'll learn how to index a vector by using negative integers.

Another approach for indexing or selecting elements from a vector uses negative integers.

Here the negative sign indicates that the elements should be dropped or excluded.

Any element with an index not explicitly indicated by a negative integer is included in the output subset.

This approach is useful if the number of elements that you want to be excluded in the subset is easy to specify, and you know the exact locations of the excluded elements.

For instance, suppose we want to select all the elements of `numeric.vector` except for the second element.

In this case, we know exactly the location of the element to be excluded, so it's easy to drop this element by using a negative integer:

```
numeric.vector[ -2 ]
```

```
## [1] -3  9  2  3
```

We can also use a vector of negative integers.

To drop the second and fifth element from the vector, we have:

```
numeric.vector[ c(-2, -5) ]
```

```
## [1] -3  9  2
```

We can construct the indexing vector however we like, as long as it consists entirely of negative integers.

For instance, to select all the elements of `numeric.vector` except for the second, third, and fourth elements, we can use the colon operator:

```
numeric.vector[ -2:-4 ]
```

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

What happens if you try to use both positive and negative values in a single vector indexing operation?

R will generate an error, and with good reason too, because such an expression is uninterpretable.

```
numeric.vector[ c(1, 3, -5) ]
```

```
## Error in numeric.vector[c(1, 3, -5)]: only 0's may be mixed with negative subscripts
```

The problem here is that R can't determine what do with the second element.

The rule for positive integer indexing is that any element that isn't explicitly included is dropped from the subset vector.

Thus, since we are using positive integer indexing with 1 and 3, that would imply that R should exclude the second element from the vector.

However, the rule for negative integer indexing is that any element that is *not* specified is retained.

By that convention, R should include the second element.

Even if you don't quite understand this reasoning, it's easy to remember this simple rule: don't try to use both positive and negative integer indexing in a single expression, because it's not going to work.

So that's how to index a vector by using negative integers.

Now let's review what we've learned in this module.

Exercise 3.2: Indexing with negative integers

Let's construct a simple vector:

```
exercise.vector <-
    c( 6, 3, 2, 8, 6, 4, 1, 5, 9, 0)
```

Part (a)

Select all the elements of this vector except for the seventh element.

Solution

Part (b)

Select all the elements of this vector except for the elements in the third, sixth, and eighth positions.

Solution

Part (c)

Select all the elements of the vector except for the first through third elements.

Solution

Module Review

In this module, we learned about vector indexing, which allows us to select elements from a vector.

- In section 1, we discussed the four types of vector indexing methods.
- In section 2, we learned how to index a vector by using positive integers.
- In section 3, we learned how to index a vector by using negative integers.

Now that you've completed this module, you should be able to:

- Explain the concept of vector indexing, and list the four types of indexing methods.
- Index a vector by using positive integers.
- Index a vector by using negative integers.

There were no new built-in R functions in this module.

All right! That's it for Module 3: Indexing Vectors.

Now let's move on to Module 4: Vectorized Operations.

Solutions to the Exercises

Exercise 3.1: Indexing with positive integers

Let's construct a simple vector:

```
exercise.vector <-
    c( 6, 3, 2, 8, 6, 4, 1, 5, 9, 0)
```

Part (a)

Select the fourth element of this vector.

Solution

```
exercise.vector[ 4 ]
```

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

Part (b)

Select the elements in the second, fifth, and seventh positions.

Solution

```
exercise.vector[ c(2, 5, 7) ]
```

```
## [1] 3 6 1
```

Part (c)

Select the fourth through the eighth elements of the vector.

Solution

```
exercise.vector[ 4:8 ]
```

```
## [1] 8 6 4 1 5
```

Exercise 3.2: Indexing with negative integers

Let's construct a simple vector:

```
exercise.vector <-
    c( 6, 3, 2, 8, 6, 4, 1, 5, 9, 0)
```

Part (a)

Select all the elements of this vector except for the seventh element.

Solution

```
exercise.vector[ -7 ]
```

```
## [1] 6 3 2 8 6 4 5 9 0
```

Part (b)

Select all the elements of this vector except for the in the third, sixth, and eighth positions.

Solution

```
exercise.vector[ c( -3, -6, -8 ) ]
```

```
## [1] 6 3 8 6 1 9 0
```

Part (c)

Select all the elements of the vector except for the first through third elements.

Solution

```
exercise.vector[ -1:-3 ]
```

```
## [1] 8 6 4 1 5 9 0
```