Lecture 5, Module 3: Logical Indexing

CSCI E-5a: Introduction to R

Let's clear the global environment:

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

Module Overview and Learning Objectives

Hello! And welcome to Module 3: Logical Indexing.

In this module, we'll explore another method of indexing a vector, called *logical indexing*.

- In section 1, we'll define the basic concept of logical indexing.
- In section 2, we'll see how to use vectorized comparison operations with logical indexing.
- In section 3, we'll learn how to use logical indexing with two vectors.

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

- Explain how logical indexing works.
- Use vectorized comparison operations to construct logical indexing vectors.
- Use one vector to perform logical indexing on another vector.

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

All right! Let's get started by learning how to index a vector by using logical values.

Section 1: Logical Indexing

> **Main Idea:** *We can index a vector by using logical values*

In this section, we'll define the basic concept of logical indexing.

So far, we've seen 3 different methods for indexing a vector:

- Indexing with positive integers.
- Indexing with negative integers.
- Indexing a named vector with character strings.

When we first encountered vector indexing, I mentioned that there were actually 4 methods for indexing a vector, so we still have one more method to learn.

In this module, we'll learn how to index a vector using logical values.

In this approach, we have a vector of any type from which we want to select elements, and a logical indexing vector of the same length.

The key idea is that R will select the elements of the original vector for which the corresponding element in the logical indexing vector is `TRUE`.

Let's see an example.

Let's start with a simple numeric vector:

```
numeric.vector <-
    c( 4, 7, 2, 1, 8 )
```

We will also construct a logical vector of the same length as `numeric.vector`:

```
logical.indexing.vector <-
    c( FALSE, TRUE, FALSE, FALSE, TRUE )
```

Now look what happens when we index the original numeric vector with the logical indexing vector:

```
numeric.vector[ logical.indexing.vector ]
```

```
[1] 7 8
```

Do you see how this works?

The logical indexing vector has a `TRUE` value in positions 2 and 5.

Therefore, when we use logical indexing with this vector, R will select the elements of the original numeric vector in positions 2 and 5.

Make sure you understand how this process works.

Let's try another example, this time with a character string vector:

```
character.vector <-
    c( "Red", "Cat", "Dog", "Pen", "Phone",
       "White", "Mouse", "Bird", "Piano", "Bob",
       "Anita", "Hedgehog" )
```

Notice that the length of `character.vector` is 12:

```
length( character.vector )
```

```
[1] 12
```

Let's create a logical vector that we can use to index ’character.vector‘:

```
logical.indexing.vector <-
    c(FALSE, FALSE, TRUE, FALSE, FALSE, FALSE,
      TRUE, FALSE, TRUE, FALSE, FALSE, FALSE )
```

Let's check to make sure that `logical.indexing.vector` does indeed have 12 elements:

```
length( logical.indexing.vector )
```

```
[1] 12
```

What will be the output if you run this code?

Try to predict the result before you run the code:

```
character.vector[ logical.indexing.vector ]
```

```
[1] "Dog"   "Mouse" "Piano"
```

So that's the basic concept of logical indexing.

Now let's see how to use vectorized comparison operations with logical indexing.

Exercise 3.1: Logical indexing

Construct a logical indexing vector to select the elements "Cat", "Mouse", and "Hedgehog" from `character.vector`.

Solution

Section 2: Constructing Logical Indexing Vectors

> **Main Idea:** *We can use vectorized comparison operations with logical indexing*

In this section, we'll see how to use vectorized comparison operations with logical indexing.

At first, the method of logical indexing might appear to be a cumbersome approach.

Let's go back to our example with the numeric vector:

```
numeric.vector
```

```
[1] 4 7 2 1 8
```

When we use logical indexing, we have to construct a logical vector of length equal to the vector that we're trying to index.

In this case, that means that we have to construct a vector of length 5.

We also have to explicitly specify a logical value for each element of the original vector, even if we *don't* want to select it.

If we just want to select the elements in positions 2 and 5, then surely it's easier to use positive integer indexing with a vector:

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

```
[1] 7 8
```

And this is indeed the case, if you know in advance that you want to select the elements in positions 2 and 5.

But what happens if you don't know in advance the exact locations of the elements that you want to specify?

In fact, in practice you usually *don't* know in advance the exact locations of the elements that you want to select.

In this case, we want to be able to provide some sort of specification for the elements that we want, and then let R work out which elements to select.

We can provide that specification by using a vectorized comparison operation.

Let's suppose that we want to select all the elements of the vector that are strictly greater than the value 6.

First, we can use a vectorized comparison operation to construct a logical vector of `TRUE` and `FALSE` values, where the value `TRUE` occurs if and only if the corresponding value in the numeric vector is strictly greater than 6.

```
logical.indexing.vector <-
    (numeric.vector > 6)

logical.indexing.vector
```

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

Now we can use this logical indexing vector to index the numeric vector:

```
numeric.vector[ logical.indexing.vector ]
```

```
[1] 7 8
```

Because we are using a vectorized comparison operation, the logical indexing vector will be guaranteed to have the same length as the original numeric vector.

We can put all of this into a single line of code:

```
numeric.vector[ numeric.vector > 6 ]
```

```
[1] 7 8
```

Notice that in this approach we *don't* have to know in advance where the elements we wanted are located.

Instead, we are essentially providing a description of the elements that we want, and then R can do the work of determining their positions and selecting them.

If you're working with a dataset, especially a large dataset, it's usually the case that you don't know the exact locations of the elements that you want to select.

By using logical indexing, you don't have to know these locations.

In the previous module, we saw how to calculate a sample count by using a vectorized logical comparison operation.

For instance, we can count the number of elements in `numeric.vector` that are strictly greater than 6 by using this construct:

```r
sum( numeric.vector > 6 )
```

```
[1] 2
```

Notice how similar these two approaches are since they both use this method of a vectorized logical comparison.

Also, in the previous module, I mentioned how weird this construct is, because it's strange to use a logical comparison operation as the input to a numeric function.

When we use logical indexing to select values from a vector, it's even weirder, because now we're using a logical comparison operation as an index.

If you've had prior programming experience with langauges such as C or BASIC or Java, this can be a difficult concept to navigate, just because it's so far away from the paradigms of these classes.

Paradoxically, this can be a situation where having experience is actually a liability, because you have to revise your conception of how to index a one-dimensional structure.

Let's try this again.

This time, we'll select the elements from `numeric.vector` that are less than or equal to 2.

First, let's construct the logical indexing vector:

```r
## Example 14: Constructing a vector of logical values with vectorized operations

logical.indexing.vector <-
    numeric.vector <= 2

logical.indexing.vector
```

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

Make sure you understand how R determines each of the elements of this vector!

Now we can use the `logical.indexing.vector` to index the original numeric vector:

```r
## Example 15: Indexing with a logical vector

numeric.vector[ logical.indexing.vector ]
```

```
[1] 2 1
```

As before, let's put this into one line of code:

```r
## Example 16: One-line indexing with a logical vector

numeric.vector[ numeric.vector <= 2 ]
```

```
[1] 2 1
```

So that's how we can use vectorized comparison operations with logical indexing.

Now let's see how to use logical indexing with two vectors.

Exercise 3.2: Using a comparison operation

The `rivers` vector is a built-in vector consisting of the lengths of the longest 141 rivers in North America.

Select the lengths of the rivers that are strictly longer than 1000 miles.

Solution

Section 3: Logical Indexing with Two Vectors

> **Main Idea:** *We can use logical indexing with two vectors*

In this section, we'll learn how to use logical indexing with two vectors.

Remember that when we use logical indexing, it doesn't matter how we construct the logical vector.

All that matters is that the logical vector has the same length as the vector that's being indexed.

So far, we've constructed the indexing vector by performing a comparison operation on the vector that's actually being indexed.

Thus, we're using the vector itself to construct the logical indexing vector.

But we don't have to do this – we can use another vector to construct the logical indexing vector.

This is a very powerful technique, and it's going to be important for us throughout the course.

Let's see an example.

Recall our grocery store that sells three brands of breakfast cereal: Sugar Bomz (SBZ), Krispee Yummm!! (KYM), and Healthy Kale and Tofu (HKT).

We have a small dataset of 7 transactions, and for each transaction we have the brand of cereal and the number of boxes that were purchased:

Transaction	Brand Name	Number of Boxes
1	SBZ	4
2	SBZ	6
3	KYM	3
4	SBZ	5
5	HKT	1
6	KYM	2
7	HKT	2

Let's put the brand and number of boxes sold into vectors:

```r
## Example 17: Constructing vectors for the brand name and number of boxes sold

brand.name.vector <-
    c( "SBZ", "SBZ", "KYM", "SBZ",
       "HKT", "KYM", "HKT" )

number.of.boxes.sold.vector <-
    c( 4, 6, 3, 5, 1, 2, 2 )
```

We would like to calculate the total number of boxes of Krispee Yummm! that were sold.

First, let's select the values from the vector of number of boxes sold that correspond to Krispee Yummm!.

To do this, we construct a logical vector using the vector of brands:

```r
## Example 18: Constructing a logical indexing vector for KYM

kym.logical.indexing.vector <-
    brand.name.vector == "KYM"

kym.logical.indexing.vector
```

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

As always, make sure that you're clear on exactly how the values in the `kym.logical.indexing.vector` are determined.

Now we can use the `kym.logical.indexing.vector` to index the vector containing the number of boxes sold:

```r
## Example 19: Selecting the number of boxes sold for KYM

kym.number.of.boxes.sold.vector <-
    number.of.boxes.sold.vector[
        kym.logical.indexing.vector
    ]

kym.number.of.boxes.sold.vector
```

```
[1] 3 2
```

This vector is just a regular numeric vector, and we can calculate the total number of boxes sold using the `sum()` function:

```r
## Example 20: Calculating the total number of boxes sold for KYM

kym.total.boxes.sold <-
    sum( kym.number.of.boxes.sold.vector )

cat(
    "KYM total boxes sold:",
    formatC(
        kym.total.boxes.sold,
        format = "f",
        digits = 0
    )
)
```

```
KYM total boxes sold: 5
```

Let's try this again.

This time, we'll compute the average number of boxes sold, given that the customer is purchasing Healthy Kale and Tofu.

First, we construct the logical indexing vector, using the brand vector:

```
## Example 21: Constructing a logical indexing vector for HKT

hkt.logical.indexing.vector <-
    brand.name.vector == "HKT"

hkt.logical.indexing.vector
```

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

Now we'll select the values from the `number.of.boxes.sold` that correspond to Healthy Kale and Tofu:

```
## Example 22: Selecting the number of boxes sold for HKT

hkt.number.of.boxes.sold.vector <-
    number.of.boxes.sold.vector[
        hkt.logical.indexing.vector
    ]

hkt.number.of.boxes.sold.vector
```

```
[1] 1 2
```

Finally, we can calculate the brand-specific mean number of items purchased per transaction:

```
## Example 23: Calculating the average number of boxes sold for HKT

mean.hkt.boxes.sold <-
    mean( hkt.number.of.boxes.sold.vector )

cat(
    "Mean number of HKT boxes, per sale:",
    formatC(
        mean.hkt.boxes.sold,
        format = "f",
        digits = 2
    )
)
```

```
Mean number of HKT boxes, per sale: 1.50
```

We can do this in a very compressed manner:

```
## Example 24: Compressed calculation of the average number of boxes sold

mean(
    number.of.boxes.sold.vector[
        brand.name.vector == "HKT"
    ]
)
```

```
[1] 1.5
```

OK, that's really compressed.

I think that sometimes people think that R is a difficult language because of expressions like this.

You don't have to write code like that, and in fact I'm not sure that it's particularly readable.

In CSCI 5a, we want you to write correct code above all, and we aren't concerned if you use the most compressed approach imaginable.

Remember – the most important thing is that your code is *correct* – that is, that it really does do the thing that you want it to do.

Also, your code should be clear and readable.

Writing tricky ultra-compressed R expressions is much less important, especially if it means that you compromise either correctness or readability.

So you don't have to use this construct if you're not comfortable with it.

However, I would encourage you to study this so that you understand what's going on, because there are a lot of R concepts of techniques at work in this one small code block.

So that's how to use logical indexing with two vectors.

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

Exercise 3.3: Brand-specific sum and mean

Using logical indexing, determine the total number of boxes of Sugar Bomz sold, and also the average number of boxes sold per transaction.

Solution

Module Review

In this module, we explored another method of indexing a vector, called *logical indexing.*

- In section 1, we defined the basic concept of logical indexing.
- In section 2, we saw how to use vectorized comparison operations with logical indexing.
- In section 3, we learned how to use logical indexing with two vectors.

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

- Explain how logical indexing works.
- Use vectorized comparison operations to construct logical indexing vectors.
- Use one vector to perform logical indexing on another vector.

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

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

Now let's move on to Module 4: The Special Value `NA`.

Solutions to the Exercises

Exercise 3.1: Logical indexing

Construct a logical indexing vector to select the elements “Cat”, “Mouse”, and “Hedgehog” from `character.vector`.

Solution

For your reference, here’s the `character.vector`:

```
character.vector
```

```
 [1] "Red"      "Cat"      "Dog"      "Pen"      "Phone"    "White"
 [7] "Mouse"    "Bird"     "Piano"    "Bob"      "Anita"    "Hedgehog"
```

Now we’ll construct this logical vector:

```
logical.indexing.vector <-
    c( FALSE, TRUE, FALSE, FALSE,
       FALSE, FALSE, TRUE, FALSE,
       FALSE, FALSE, FALSE, TRUE )
```

Now we’ll perform logical indexing:

```
character.vector[ logical.indexing.vector ]
```

```
[1] "Cat"      "Mouse"    "Hedgehog"
```

Exercise 3.2: Using a comparison operation

The `rivers` vector is a built-in vector consisting of the lengths of the longest 141 rivers in North America.

Using logical indexing with a vectorized comparison to construct a vector with the values of the lengths of the rivers that are strictly longer than 1000 miles.

Solution

```
rivers[ rivers > 1000 ]
```

```
 [1] 1459 1450 1243 2348 1171 3710 2315 2533 1306 1054 1270 1885 1100 1205
[15] 1038 1770
```

Exercise 3.3: Brand-specific sum and mean

Using logical indexing, determine the total number of boxes of Sugar Bomz sold, and also the average number of boxes sold per transaction.

Solution

```r
sbz.logical.indexing.vector <-
    brand.name.vector == "SBZ"

sbz.number.of.boxes.sold.vector <-
    number.of.boxes.sold.vector[
        sbz.logical.indexing.vector
    ]

sbz.total.boxes.sold <-
    sum( sbz.number.of.boxes.sold.vector )

cat(
    "Total number of boxes of Sugar Bomz sold:",
    formatC(
        sbz.total.boxes.sold,
        format = "f",
        digits = 2
    )
)
```

```
Total number of boxes of Sugar Bomz sold: 15.00
```

Now for the mean number of boxes of Sugar Bomz sold per transaction:

```r
sbz.mean.boxes.sold <-
    mean( sbz.number.of.boxes.sold.vector )

cat(
    "Mean number of boxes of Sugar Bomz sold:",
    formatC(
        sbz.mean.boxes.sold,
        format = "f",
        digits = 2
    )
)
```

```
Mean number of boxes of Sugar Bomz sold: 5.00
```