Lesson three: Indexing

Brian S. Evans, Ph.D.

Introduction

The process of querying data – in other words subsetting or filtering data by some condition – is one of the most important data operations we use when evaluating data collected in the field. In this lesson we will be exploring how to query data using what is often called “indexing” or “matrix notation”. We will learn a **much easier** way to query data by the end of this course. I start by teaching this method of querying because it is helpful in understanding how R objects work, gives a glimpse of what is happening under the hood when using other querying techniques, and still used for some advanced querying techniques.

Review: logical values

Recall that logical values can be obtained by evaluating objects with **logical operators**. For example, the logical operator `==` tests whether a value is equal to another value.

Use the logical operator `==` to test whether the each set of values is equivalent ($\equiv$).

$21 \equiv 34$

$21 \equiv 21$

$21 \equiv 34 - 13$

$34 \equiv 21 + 13$

$21 \equiv 13 - 34$

$(21 \equiv 34) + (34 \equiv 21 + 13)$

Using logic with objects

We can run a logic test on objects with more than one value. This is executed in the same way as evaluating a single value

Use the logical operator `==` to examine which values in the objects below are equivalent ($\equiv$) to the number 3.

- An atomic vector object:

```
v <-
  c(1, 1, 2, 3, 5, 8)

v
```

- A matrix object:

```
m <-
  matrix(
    c(1, 1, 2, 3, 5, 8),
    ncol = 2
  )

m
```

- A data frame object:

```
df <-
  data.frame(
    a = c(1, 1, 2),
    b = c(3, 5, 8)
  )

df

df == 3
```

The logical operators

Comparing values in objects

The table below provides a set of the logical operators that are most commonly used to compare values. In each of the table items we are evaluating some object (or value) x relative to some object y.

Operator	**Usage**	**Meaning**
==	x == y	x is equal to y
!=	x != y	x is NOT equal to y
!	!(x)	not x
|	x | y	x OR y
&	x & y	x AND y

Operator	Usage	Meaning
%in%	x %in% yz	x is in the vector c(y,z)
<	x < y	x is less than y
<=	x <= y	x is less than or equal to y
>	x > y	x is greater than y
>=	x >= y	x is greater than or equal to y

Each of the logical operators above can be applied in the same way that we have applied `==` thus far.

Using vector `v`, as above, use logical operators to test the following statements:

- `v` is not equal to 3
- `v` is less than 3
- `v` is less than or equal to 3
- `v` is greater than 3
- `v` is greater than or equal to 3
- `v` is equal to 2 **or** `v` is equal to 3
- `v` is equal to 2 **and** `v` is equal to 3

Explore logical operators as above using matrix `m`:

```
m
```

Explore logical operators as above using data frame `df`:

```
df
```

Comparing sets of values

We are often interested in comparing sets of values with one another. Using the code above, we may be interested in testing whether values `1` and `3` appear in vector `v`. We can use the or statement, as above, with the syntax `v == 1 | v == 3`. This can quickly get cumbersome if we are comparing long value sets. Instead, we use the operator `%in%`, which tests the logic statement "is in".

Test which values of `v` are in the vector `c(1,3)`:

The NOT statement, `!(x)` has limited utility for simple statements. For example, the statement `!(x == 1)` is written more simply as `x != 1`. Likewise, `!(x < 1)` may be written as `x >= 1`. In more complex logical operations, such as when it used in combination with `%in%`, it becomes a powerful tool to examine sets.

Test which values of `v` are NOT in the vector `c(1,3)`:

You can also compare sets with *and/or* statements, represented in R by `&` and `|`. As described above, the statement `v == 1 | v ==3` is equivalent to `v %in% c(1,3)`. The "and" statement (`&`) is not very useful for logical problems such as these, because no value can be both 1 and 3 at the same time. The `&` operator becomes very useful when querying across multiple conditions.

Test whether values of `v` are less than 5 and not equal to 2:

Querying vectors

Recall that the position of value `x` in vector `v` would be notated as `v[x]`. This notation, indexing, instructs R to evaluate and return the statement "vector `v` where position v_x is equal to 3". R returns all of the values from vector `v` for which this statement was `TRUE`. In other words, you conducted a query of vector `v` by **position**.

Subset vector `v` to its first and second values by position

You may also query values by **condition**, using logic, in the same manner as above. For example, you may want R to evaluate the statement "vector `v` where value v_x is greater than two".

```
v

v > 2

v[v > 2]
```

Notice in the above that the line `v > 2` returned logical values of `v` associated with that test. The next line of code, `v[v > 2]` returned values from `v` that met this condition.

Why does this work? Let’s take a quick look under the hood. The function `which` provides the index location (i.e.,position) in which a logical statement evaluates as true. Using the `v > 2` query above, we can see that:

```
v

v > 2

which(v > 2)

v[v > 2]
```

All of the various logical statements can be applied in the same manner.

Use indexing to query vector `v` based on the following conditions:

- Values greater than or equal to 3
- Values that do not equal 2
- Values that equal 2
- Values that are greater than or equal to 3 but do not equal 5
- Values that are 3 or 5

Querying matrices

Review: Index matrices by position

Recall that the position of value `x,y` (row, column) in a matrix, `m`, can be notated as `m[x,y]`. Supplying a value or set values for `x` and/or `y` allows us to index matrix `m` by position. For example, the notation `m[1,2]` returns the value of matrix `m` in the first row of the second column.

If the `x` or `y` position is left blank, all values for that dimension are returned. For example, the notation `m[,]` would return all the rows and columns of matrix `m`. The notation `m[,y]` would return all values in column `y` of matrix `m`. Likewise, the notation `m[x,]` returns all values in row `x` of matrix `m`

Use indexing by position to subset matrix `m` as described below:

- The value in the first row and first column
- All values in the first row
- All values in the second column
- The first and third values of the second column

Indexing with logic

In the above, note that the object `m[,1]` is a vector. As such, we can query this vector by position using logic. Note the output of the below code, which tests `m[,1]` to determine whether each value is greater than 1:

```
m

m[,1]

m[,1] > 1
```

We see that this statement only evaluates as TRUE at one position in the vector. Let's use `which`, as above, to determine the index associated with this position:

```
which(m[,1] > 1)
```

Because our `which` statement evaluates to 3, the following queries are equivalent:

```
m[3, ]

m[m[,1] > 1, ]
```

Subset matrix `m` as described below:

- All values in which the values of column 1 are equal to 1
- All values in which the values in column 2 are equal to 3 or 8
- All values in which the values in column 2 are greater than 3
- All values in which the values in column 1 are less than 2

Querying data frames

Review: Index data frames by position

Just like matrices, the position of value `x,y` (row, column) in a data frame, `df`, can be notated as `df[x,y]`.

Use indexing by position to subset matrix `m` as described below:

- The value in the first row and first column
- The value in the second row and second column
- All values in the second row
- All values in the second column
- The first and third values in the second column

An important difference between querying matrices and dataframes is that data frames have a "names" attribute by default:

```
# Attributes of object df:

attributes(df)
```

Recall that the component vectors in a data frame (i.e., columns – in this case `c(1,1,2)`) can be accessed by name using the `$` notation:

```
# Data frame indexing by position ...

df[,1]

# ... is equivalent to:

df$hello
```

The above is very useful for querying data frames, because the `$` notation is much easier to read than "matrix notation".

Convert the following expressions to `$` notation:

```
df[,1] > 1
```

```
## Warning in Ops.factor(df[, 1], 1): '>' not meaningful for factors
```

```
which(df[,1] > 1)
```

```
## Warning in Ops.factor(df[, 1], 1): '>' not meaningful for factors
```

```
df[df[,1] > 1,]
```

```
## Warning in Ops.factor(df[, 1], 1): '>' not meaningful for factors
```

End of lesson!

SMITHSONIAN'S NATIONAL ZOO
& CONSERVATION BIOLOGY INSTITUTE