Intermediate R

Allison Theobold

Relational Operators

Relational operators tell how one object relates to another. There are a variety of relational operators that you have used before in mathematics, but take on a slightly different feel in computer science. We will walk through a few examples of different relational operators and the rest will be left as exercises.

- **Equality & Inequality**: This type of operator tells whether an object is equivalent to another object, or its negation.

```
# write R code to see if
    # TRUE equals FALSE.
    # if -6 * 14 is not equal to 17 - 101
    # if the strings "useR" and "user" are equal
    # are TRUE and 1 equal?
```

- **Greater & Less**: These statements should be familiar, with a bit of a notation twist. To write a greater than or equal to statement you would you `= >` in your statement, and similarly with less than or equal to.

```
# write R code to see if
    # -6 * 5 + 2 is greater than or equal to -10 + 1
    # "raining" is less than or equal to "raining dogs"
    # TRUE is greater than FALSE
```

- **Matricies**: For matricies the above relational statements can be applied across the entire matrix, or to a subset of the matrix (a vector). The relational statement are applied elementwise (a stepwise progression through the entries)

```
# These dating data, of messages received per week, have been created for you

okcupid <- c(16, 9, 13, 5, 2, 17, 14)
match <- c(17, 7, 5, 16, 8, 13, 14)
messages <- matrix(c(okcupid, match), nrow = 2, byrow = TRUE)

# Use the messages matrix to return a logical matrix that answers the following questions:

  # When were the messages equal to 13?
  # For which days were the number of messages less than or equal to 14?
```

Logicals

These statements allow for us to change or combine the results of the relational statements we discussed before, using "and", "or", or "not".

- **"and"** statements evaluate to `TRUE` only if every relational statement evaluates to `TRUE`. For example, `(3 < 5) & (9 > 7)` would evaluate to `TRUE` because both relational statements are `TRUE`. If instead we had `(3 > 5) & (9 > 7)`, this would evaluate to `FALSE` as the first relational statement is `FALSE`.
- **"or"** statements evaluate to `TRUE` if at least one relational statement evaluates to `TRUE`. For example, `(3 > 5) | (9 > 7)` would evaluate to `TRUE` because one of the relational statements is `TRUE` (the

second one). If instead we had `(3 > 5) & (9 < 7)`, this would evaluate to `FALSE` as both relational statements evaluate to `FALSE`.

- **Remark:** The `&&` and `||` logical statements do not evaluate the same as their single counterparts. Instead, these logical operators evaluate to `TRUE` or `FALSE` based only on the first element of the statement, vector, or matrix.

```
# Using the messages data from above, answer the following questions.

    # Is last under 5 or above 10?
    # Is last between 15 and 20, excluding 15 but including 20?
```

Logicals in Your Daily (Statistics) Life

Many of you may be wondering how the topics above relate to your daily lives in statistical practices, but wait! Let's play with some data some functions that are likely very familiar to you. Import the BlackfootFish dataset and carry out the following questions:

- Which fish do no have both length and weight recorded? (hint: use relational operators, logicals, and the `which` or `subset` functions)
- Remove the fish you found from the dataset, renaming the new dataset `BlackfootFish2`.
- Subset these data so that only the Rainbow Trout (RBT) and Brown Trout (Brown) remain.

Conditional Statements

Conditional statements utilize relational statements and logicals to change the results of your `R` code. You may have encountered an `ifelse` statement before (or not), but let's breakdown exactly what `R` is doing when it evaluates them.

If Statements

First, let's start with an `if` statement, the often overlooked building block of the `ifelse` statement. The `if` statement is structured as follows:

```
if(condition){
              statement
}
```

- the condition inside the parenthesis is what the computer executes to verify that it is `TRUE`,
- if the condition evaluates to `TRUE` then the statement inside the `{}` is output, and
- if the condition is `FALSE` nothing is output.

Else Statements

Since whenever an `if` statement evaluates to `FALSE` nothing is output, you might see why an `else` statement might be beneficial! An else statement allows for another statement to be output whenever the `if` condition evaluates to `FALSE`. The `ifelse` statement is structured as follows:

```
if(condition){
               statement1
}
else{
     statement2
}
```

- again, the `if` condition is executed first,
  - if it evaluates to `TRUE` then the first statement (`{statement1}`) is output,
  - if the condition is `FALSE` the computer moves on to the `else` statement, and
- the second statement (`{statement2}`) is output.

Else If Statements

On occation, you may want to add a third (or forth, or fifth, ...) condition to your `ifelse` statement, which is where the `elseif` statement comes it. The `elseif` statement is added to the `ifelse` as follows:

```
if(condition1){
               statement1
}
elseif(condition2){
               statement2
}
else{
               statement3
}
```

- The `if` condition is executed first,
  - if it evaluates to `TRUE` then the first statement (`{statement1}`) is output,
  - if the condition is `FALSE` the computer moves on to the `elseif` condition,
- Now the second condition is executed,
  - if it evaluates to `TRUE` then the second statement (`{statement2}`) is output,
  - if the condition is `FALSE` the computer moves on to the `else` statement, and
- the third statement (`{statement3}`) is output.