Data Frames Part 2: Sequences and subsetting

Why, and what?

In this lesson, we address two things that can become cumbersome: precisely typing out long sequences of numbers, and pulling out specific data from a data frame. These things can be made much more efficient through the use of functions in R!

To this end, we will introduce the concept of sequences and the associated function `seq()`, as well as approaches for subsetting data frames. You will then have the opportunity to put the two concepts together in the challenge, which will ideally begin to give you a sense for their general usefulness.

Sequences

There are several ways that we can automatically create sequences of numbers. If we want to make numeric vectors of integers in increasing or decreasing order, we can use a special inline function: `:`. For example, try `1:10` and `10:1`.

For more complex patterns, we can use the function `seq()` (for **seq**uence). You can specify the numbers in the output vector in several different ways:

```r
seq(2, 10, by=2)  # Specify first number, last number, and increment between numbers
```

```
## [1]  2  4  6  8 10
```

```r
seq(5, 10, length.out=3)  # Specify first number, last number, and total vector length
```

```
## [1]  5.0  7.5 10.0
```

```r
seq(50, by=5, length.out=10)  # Specify first number, increment, and total vector length
```

```
##  [1] 50 55 60 65 70 75 80 85 90 95
```

```r
seq(1, 8, by=3) # Sequence stops to stay below upper limit
```

```
## [1] 1 4 7
```

Challenge

1. Create a vector with 6 elements that counts up from 27.
2. Create a vector with 5 equally spaced elements between 100 and 200.
3. Create a vector of every other number between 24 and 42.
4. Create a vector of numbers less than 100 that are divisible by 5.

5. Why does the following code give an error? `seq(2, 10, by=2, length.out=6)`

Subsetting data frames

Let's work with the data frame `trees`.

```r
trees <- read.csv(file="Data/trees.csv")
str(trees)
```

```
## 'data.frame':    300 obs. of  5 variables:
##  $ Province: Factor w/ 3 levels "New Brunswick",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ Site    : Factor w/ 10 levels "Fredericton",..: 5 5 5 5 5 5 5 5 5 5 ...
##  $ Plot    : int  1 1 1 1 1 1 2 2 2 2 ...
##  $ Species : Factor w/ 6 levels "Acer rubrum",..: 4 1 2 5 3 6 4 1 2 5 ...
##  $ Count   : int  11 18 28 15 18 30 30 23 37 20 ...
```

This data frame has 2 dimensions: rows and columns. If we want to extract data from it, we can use square brackekts (`[` and `]`), as we have done with vectors - but we need to specify coordinates for both row and column. When two values are specified, rows always come first and columns always come second. However, the way we specify coordinates will determine the class of the output (vector versus data.frame). Try the examples below.

```r
# If we only use one number, this specifies only the column, and returns a data.frame.
    trees[1]      # First column in the data frame (as a data.frame)

# Here we specify both rows and columns by including a comma.
    trees[1, 1]    # First element (first row) in the first column of the data frame (as a vector)
    trees[1, 5]    # First element (first row) in the 5th column (as a vector)

# We can use vectors and sequences to specify multiple rows or columns.
    trees[c(1,2,3), 4] # First three elements (first 3 rows) in the 4th column (as a vector)
    trees[1:3, 4] # Exactly the same output as above

# We can also leave one place empty to specify *all* rows or *all* columns.
    trees[3, ]     # The 3rd element (row) for all columns (as a data.frame)
    trees[, 3]     # the entire 3rd column (as a vector)

head_trees <- trees[1:6, ] # equivalent to head(trees)
```

We can use this approach to create new data frames from already existing ones.

```r
trees_small <- trees[1:4,]
trees_small
```

```
##   Province   Site Plot         Species Count
## 1  Ontario Ottawa    1  Pinus strobus    11
## 2  Ontario Ottawa    1    Acer rubrum    18
## 3  Ontario Ottawa    1 Cornus florida    28
## 4  Ontario Ottawa    1   Quercus alba    15
```

You can also exclude certain parts of a data frame by using a minus sign.

```r
trees[, -1]    # The whole data frame, except the first column
trees[-c(7:300), ]  # Equivalent to head(trees)
```

Columns in a data frame can also be called by name, rather than using numeric values. There are several ways to do this:

```
trees["Species"]        # Result is a data.frame with only this column.
trees[, "Species"]      # Result is a vector
trees[["Species"]]      # Result is a vector
trees$Species           # Result is a vector
```

The main difference in approach is that with the `$`, you can use partial matching on the name. To demonstrate, try the following:

```
trees$Sp
```

However, this can be a dangerous shortcut! For example, what happens if you only use `trees$S`? Why do you think this happens?

It is better to be explicit, and indicate the full name of the variable. Besides, with RStudio, you can use auto-completion to avoid typing more than a few characters to still get the full name.

Challenge

6. Create a new data frame that contains only the 250th row of `trees`.
7. Create a new data frame that includes the 10th through 20th rows of `trees`.
8. Create a new data frame that only includes data where `Count` is equal to 30. (Hint: How would you identify which rows are the correct ones? Look back at *Conditional Subsetting* in *Introduction to R*!)
9. The function `nrow()` on a data frame returns the number of rows. Use `nrow()` to make a data frame with only the last row of trees.
10. Use `nrow()` again, in conjuction with `seq()`, to create a new data frame that includes every 10th row of `trees` starting at row 10 (10, 20, 30, . . .). (Hint: start by creating the appropriate sequence!)
11. Now that you've seen how `nrow()` can be used to stand in for a row index, let's combine that behavior with the `-` notation above to reproduce the behavior of `head(surveys)`. I.e., exclude the 7th through final row of the `surveys` dataset without explicitly specifying the final row number.