Operations on Vectors

Sam Mason

Introduction

Vectors are useful data storage objects in R. We'll come to think of vectors as variables. For example, we could define a *student height* variable and describe it using a vector of values corresponding to the height of all Gordon College students. Let's actually create this vector. Undergraduate enrollment at Gordon is currently 1451, and let's say the mean height of the student body is 170 cm (just under 5' 8"). We'll also assume that height is normally-distributed. To create our vector, we can use the `rnorm()` function, which takes three arguments: the sample size (n), the mean, and the standard deviation.

```r
set.seed(255) # so our random number generator always produces the same vector
height <- rnorm(n = 1451, mean = 170, sd = 6)
```

This code creates a vector containing 1451 values randomly drawn from a normal distribution with a mean of 170 cm and a standard deviation of 6. I name and define the arguments of the function explicitly, but you can also feed information to function implicitly.

```r
height <- rnorm(1451, 170, 6)
```

This produces the same vector because R always expects the order of arguments for the `rnorm()` function to be **n**, then **mean**, then **sd**. Consequently, if you choose not to define the arguments explicitly, you must feed data into the function in the correct order. You can always look at the arguments of a function using the `?` prefix.

```r
?rnorm()
```

Describing Data

R is a statistical programming language, and shows up to the party with all kinds of helpful statistical functions built-in, including `max()`, `min()`, `mean()`, `median()` `sd()`, and `var()`.

Task 1

We can call many of these functions independently, or we can print out a bunch of these summary statistics using the `summary()` function. Call the `summary()` function on the `height` vector without assigning it to a new object. This will print the results directly to the console, but will not save the results to the environment.

Task 2

We can visualize some of the results of `summary()` using the `boxplot()` function. Call boxplot on the `height` vector. A box plot reports the median (thick black line), the first and third quartiles (bottom and top of the box), and a measure of spread by adding "whiskers" (dashed lines with solid, horizontal caps) equal to the interquartile range (the difference between the third and first quartiles; `IQR()`) times 1.5 (this multiplier can be modified by the `range =` argument of the `boxplot()` function). All points shown explicitly have values outside of the whiskers.

Task 3

The coefficient of variation (CV) is a useful metric when comparing the spread (the degree to which data falls far from the mean; the thickness of a distribution) of two vectors. We calculate the CV by dividing the standard deviation of a numeric vector by its mean. Create a new objected called `height_cv` and assign it the value of the CV of the `height` vector.

Task 4

Create a new object called `min_height` and assign it the value of the shortest person. Assign the height of the tallest person to `max_height`. Create a third object called `height_diff` and assign it the value of the difference between the tallest and shortest person.

Indexing Vectors

Task 5

Let's calculate this same value a slightly different way. Call up the help documentation on the `range()` function. What does the `...` (ellipsis) argument mean? What does the `range()` function return? Create a new object called `height_range` and assign it the output of the `range()` function.

I said at the top that vectors are a useful way to store data. This is, in part, because R keeps a perfect record of where each value exists in the (potentially) long sequence of data. We can pull out specific values from a vector using some special syntax in a process called *indexing*. Unlike other languages, R starts counting at 1 (instead of 0). To pull out just the first value of the `height` vector we can use the following code.

```r
height[1]
```

```
## [1] 186.4318
```

In the code below I demonstrate numerous ways to extract data from a vector.

```r
height[1:5] # The ":" symbol clips out all values with indices from 1 to 5
height[37:162] # Printing the 37th through the 162nd values of height

height[c(1, 4, 106, 563, 1011)] # The c() function concatenates (creates a vector out of)
                                # all of its arguments. Here we make a vector of indices
                                # within the square brackets, letting R know that we only
                                # want to print the values of height at those specific
                                # locations

height[-(1001:1451)] # Using the "-" symbols tells R that we want to print everything from
                     # height except the last 451 values
```

```
height(-c(2, 280, 313)) # Print everything except the specific values corresponding to
                        # these specific indices
```

Task 6

Calculate the difference in height between the tallest and shortest person by subtracting the first value of the `height_range` vector from the second value. No need to create a new object for this operation. Just print it to the console to make sure it is the same value assigned to `height_diff`.

Manipulating Vectors

In data science we often need to create new data from existing data. We can do this by applying a function to a vector to transform it. For example, we can calculate the natural log of every value of `height` by calling the `log()` function on it.

```
log_height <- log(height)
```

We can also transform a vector by operating on it using basic arithmetic.

```
half_height <- height*0.5 # Multiplying every value of height by 0.5
```

Task 7

We often also create new data by combining existing vectors. Create a new vector called `taller` that is the sum of `height` and `half_height`.