Lab #3: Vectors and Data Types

Jacob Jameson

We expect you to watch the Module 3 material, here prior to lab.

General Guidelines:

You will encounter a few functions we did not cover in the lecture video. This will give you some practice on how to use a new function for the first time. You can try following steps:

1. Start by typing `?new_function` in your Console to open up the help page
2. Read the help page of this new_function. The description might be too technical for now. That's OK. Pay attention to the Usage and Arguments, especially the argument `x` or `x`,`y` (when two arguments are required)
3. At the bottom of the help page, there are a few examples. Run the first few lines to see how it works
4. Apply it in your lab questions

It is highly likely that you will encounter error messages while doing this lab Here are a few steps that might help get you through it.

1. Locate which line is causing this error first
2. Check if you may have a typo in the code. Sometimes another person can spot a typo faster than you.
3. If you enter the code without any typo, try googling the error message
4. Scroll through the top few links see if any of them helps
5. Try working on the next few questions while waiting for answers by TAs

Warm-up

1. In the lecture, we covered `c()`, `:`, `rep()`, `seq()` among other ways to create vectors.

```
dolly = c(9, 10, 11, 12, 13, 14, 15, 16, 17)
bees = c("b", "b", "b", "b", "b")
```

a. Recreate dolly using `:`.
b. Create the same vector using `seq()`.
c. Recreate bees using `rep()`.

2. We are now going to use the functions `rnorm()` and `runif()` to initialize vectors.

```
random_norm = rnorm(100)
random_unif = runif(1000)
```

a. How long are the vectors `random_norm` and `random_unif`? Use `length()` to verify.
b. What are the largest and smallest values in `random_norm` and `random_unif`? Use `min()` and `max()`.
c. Use `mean()` and `sd()` to calculate the mean and standard deviation of the two distributions.
d. Create a new vector with 10000 draws from the standard normal distribution.
e. `rnorm()` by default sets mean = 0 (see `?rnorm`). Create a vector of 10000 draws from the normal distribution with mean = 1. Use `mean()` to verify.

Notice the functions `min()`, `max()`, `mean()` and `sd()` all take a vector with many values and summarize them as one value. These are good to use with `summarize()` when doing data analysis on simple dataframes.

Data Types

a. Use `typeof()` to verify the data types of `dolly`, `bees`, `random_unif`
b. Coerce dolly to a character vector. Recall we have functions `as.<type>()` for this kind of coercion.
c. Try to coerce bees to type numeric. What does R do when you ask it to turn "b" into a number?

Vectorized Math

3. a and b are vectors of length 10. Look at them in the console.

```
a = 1:10
b = rep(c(2, 4), 5)
```

a. Add a and b element by element.
b. Subtract a and b element by element.
c. Divide a by b element by element.
d. Multiply a and b element by element.
e. Raise the element of a to the power of b element by element.
f. Multiply each element of a by 3 then subtract b
g. Raise each element of b to the third power.
h. Take the square root of each element of a.

Calculating Mean and Standard Deviation

Calculating the Mean

In this exercise, we will calculate the mean of a vector of random numbers. We will practice assigning new variables and using functions in R.

We can run the following code to create a vector of 1000 random numbers. The function `set.seed()` ensures that the process used to generate random numbers is the same across computers.

Note: `rf()` is a R command we use to generate 1000 random numbers according to the F distribution, and `10` and `100` are parameters that specify how "peaked" the distribution is.

```
set.seed(1)
random_numbers = rf(1000, 10, 100)
```

Write code that gives you the sum of `random_numbers` and saves it to a new variable called `numbers_sum`:

Hint: To sum the numbers in a vector, use the `sum()` function.

Note: You don't automatically see the output of `numbers_sum` when you assign it to a variable. Type `numbers_sum` into the console and run it to see the value that you assigned it.

Write code that gives you the number of items in the `random_numbers` vector and saves it to a new variable called `numbers_count`:

Hint: To count the number of items in a vector, use the `length()` function.

Now write code that uses the above two variables to calculate the average of `random_numbers` and assign it to a new variable called `this_mean`.

What number did you get? It should have been 1.018. If it isn't, double check your code!

R actually has a built in function to calculate the mean for you, so you don't have to remember how to build it from scratch each time! Check your above answer by using the `mean()` function on the `random_numbers` vector.

Calculating the Standard Deviation

Now that you've got that under your fingers, let's move on to standard deviation.

We will be converting the following formula for calculating the sample standard deviation into code:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

For this, we'll review the concept of *vectorization*. This means that an operation like subtraction will act on all numbers in a vector at the same time.

Subtract `this_mean` from the `random_numbers` vector. Did each number in `random_numbers` change?

Try to write the formula for standard deviation in R code using the `sqrt()`, `sum()`, and `length()` functions, along with other operators (`^`, `/`, `-`). Assign it to a new variable called `this_sd`. Watch out for your parentheses!

```
___ <- sqrt(sum((___ - this_mean) ^ 2) / (length(___) - 1))
```

What number did you get for `this_sd`, or the standard deviation of `random_numbers`? If you didn't get 0.489704, recheck your code!

R also has a built in function for standard deviation. Check if you calculated the standard deviation correctly by using the `sd()` function on the `random_numbers` vector.

Making a Histogram of Our Numbers

What do these random numbers look like, anyway? We can use base plotting in R to visualize the distribution of our random numbers.

Run the following code to visualize the original distribution of `random_numbers` as a histogram.

```
hist(random_numbers)
```

Notice how most of the values are concentrated on the left-hand side of the graph, while there is a longer "tail" to the right? Counterintuitively, this is known as a right-skewed distribution. When we see a distribution like this, one common thing to do is to normalize it.

This is also known as *calculating a z-score*, which we will cover next.

Calculating a Z-Score

The formula for calculating a z-score for a single value, or *normalizing* that value, is as follows:

$$z = \frac{x - \bar{x}}{s}$$

This can be calculated for each value in `random_numbers` in context of the larger set of values.

Can you translate this formula into code?

Using `random_numbers`, `this_mean`, and `this_sd` that are already in your environment, write a formula to transform all the values in `random_numbers` into z-scores, and assign it to the new variable `normalized_data`.

Hint: R is vectorized, so you can subtract the mean from each random number in `random_numbers` in a straightforward way.

Take the mean of `normalized_data` and assign it to a variable called `normalized_mean`.

Note: If you see something that ends in “e-16”, that means that it’s a very small decimal number (16 places to the right of the decimal point), and is essentially 0.

Take the standard deviation of `normalized_data` and assign it to a variable called `normalized_sd`.

What is the value of `normalized_mean`? What is the value of `normalized_sd`? You should get a vector that is mean zero and has a standard deviation of one, because the data has been normalized.

Making a Histogram of Z-scores

Let’s plot the z-scores and see if our values are still skewed. How does this compare to the histogram of `random_numbers`? Run the following code:

```
hist(normalized_data)
```

Is the resulting data skewed?

Calculating a T-Score

T-tests are used to determine if two sample means are equal. The formula for calculating a t-score is as follows:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

where $\bar{x}_i$ is the mean of the first or second set of data, s_i is the sample standard deviation of the first or second set of data, and n_i is the sample size of the ith set of data.

We'll first create two data sets of random numbers following a normal distribution:

```
set.seed(1)
data_1 <- rnorm(1000, 3)
data_2 <- rnorm(100, 2)
```

Here's how we'll calculate the mean (`x_1`), standard deviation (`s_1`), and sample size (`n_1`) of the first data set:

```
x_1 <- mean(data_1)
s_1 <- sd(data_1)
n_1 <- length(data_1)
```

What numeric types do you get from doing this? Try running the `typeof()` function on each of `x_1`, `s_1`, and `n_1`. We have you started with `x_1`.

```
typeof(x_1)
```

```
## [1] "double"
```

What object type is `n_1`?

Can you calculate the same values for `data_2`, assigning mean, standard deviation, and length to the variables of `x_2`, `s_2`, and `n_2`, respectively?

What values do you get for `x_2` and `s_2`?

Now, you should be able to translate the t-score formula ($\frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$) into code, based on the above calculated values.

What did you get for the t-score? You should have gotten 9.243, if not, double check your code!

The t-score's meaning depends on your sample size, but in general t-scores close to 0 imply that the means are not statistically distinguishable, and large t-scores (e.g. t > 3) imply the data have different means.

Performing a T-Test

Once again, R has a built in function that will perform a T-test for us, aptly named `t.test()`. Look up the arguments the function `t.test()` takes, and perform a T-test on `data_1` and `data_2`.

What are the sample means, and are they distinguishable from each other?

Well done! You've learned how to work with R to calculate basic statistics. We've had you generate a few by hand, but be sure to use the built-in functions in R in the future.

Want to improve this tutorial? Report any suggestions/bugs/improvements on here! We're interested in learning from you how we can make this tutorial better.