Vectors and data types

Contents

Introduction to Rmds

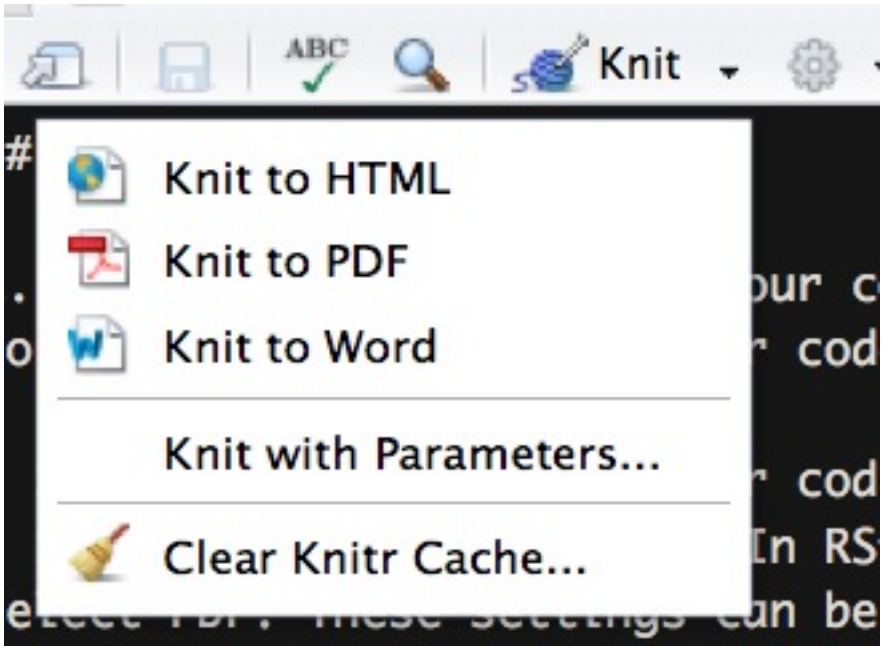
Intro to Rmds (R Markdown documents).

1. Before getting started, run the following code in the console. This gives R the necessary tools to make pdfs from your Rmds.

```
install.packages("tinytex")
tinytex::install_tinytex()
```

Note: You never want to include include code that installs packages or use `View()` in a code chunk that is **evaluated** in an Rmd. When knitting you will get an error or worse create unusual behavior without an error.

1. Create an `Rmd` file. (In RStudio's menu `File > New File > R Markdown`). Name the document and select PDF. These settings can be changed later.
2. Save the `Rmd` in your coding lab folder as `fall_lab_2.Rmd`.
3. New `Rmd`s come with some example code. Read the document and then knit to pdf by clicking the knit button. If `pdf` doesn't work, tell us. (You can usually knit to `html` as a worst case scenario.)



Pay attention to the syntax and ask your group or TA about anything you don't understand. `Rmd`s start with meta information which provides instructions to `knitr` on how to knit. After that, there's a normal code chunk which runs, but you won't see because of the `include=FALSE` bit at start of the code chunk.

```
---
title: "Lab Session 1: Read and manipulate data"
author: "Ari Anisfeld"
date: "8/26/2020"
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

Keep the part shown in the image above and erase the rest of the code and text in the document. This is how we start our own `Rmd` code.

4. Make a new code chunk by pressing Ctrl + Alt + I (Cmd + Option + I on macOS).[1] Then, add code to load the `tidyverse` in that chunk.

[1] You can also type three tick marks with `{r}` and then another three tick marks.

Warmup

1. Which of the following does not create vector with the numbers 1 to 5.

a. `rep(1, 5)`
b. `seq(1, 5)`
c. `1:5`
d. `c(1, 2, 3, 4, 5)`

2. Try to guess the output of the following code:

```
a <- 3
b <- a^2 + 1
```

3. Now try to guess again:

```
a <- c(1, 2, 3)
b <- a^2 + 1
```

4. Will this one run? If it does, what will it return? Will R complain?

```
a <- c(1, 2, 3)
b <- a^2 + c(1, 2)
```

5. Finally, what about this one?

```
a <- c(1, 2)
b <- a^2 + c(1, 2, 3, 4)
```

Calculating Mean and Standard Deviation

Calculating the Mean (by hand)

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)
```

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

2. **Note**: You don't automatically see the output of `numbers_sum` when you assign it to a variable. Add a line that is just `numbers_sum` and run it to print the value assigned to it.

3. 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: If you forgot the function, google "find the length of a vector in R".

4. 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!

5. Of course, R has a built in function to calculate the mean for you. 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:

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

Where x is a vector of length n and $\bar{x}$ is the mean of x (i.e. `this_mean`).

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.

1. Subtract `this_mean` from the `random_numbers` vector. Did each number in `random_numbers` change?
2. 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!

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!

3. 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.

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

```
hist(random_numbers)
```

2. You could also visualize this with `ggplot` but there are some extra steps. `ggplot` typically expects a data frame (tibble) in the first position so we need to explicitly tell it that the first position has the aesthetic `mapping`.

```
ggplot(mapping = aes(x = random_numbers)) +
geom_histogram()
```

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:

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

where $\bar{x}$ is `this_mean` and `s` is `this_sd`, the standard deviation of `x`.

1. 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 do this in one simple line of code!

2. 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.[2]

3. Take the standard deviation of `normalized_data` and assign it to a variable called `normalized_sd`.
4. What is the value of `normalized_mean`? What is the value of `normalized_sd`? We expect our vector has 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)
```

1. 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:

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

where $\overline{x}_i$ 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.

[2] this is scientific notation.

1. 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)
```

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)
```

3. 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"
```

4. What object type is `n_1`?
5. 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?
6. What values do you get for `x_2` and `s_2`?
7. Now, you should be able to translate the t-score formula ($\frac{\overline{x}_1 - \overline{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$) 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.

The tibble connection

Recall, the columns of `tibbles` / `data.frames` are vectors!

1. Create a tibble with random data in it. We're using a new distribution (the uniform distribution), but don't worry about the particulars. We have 500 observations from each group with a covariate `x` drawn from different uniform distributions.

```r
library(tidyverse) # we really just need dplyr
our_data <- tibble(
  group = rep(c("1", "2"), each = 500),
  x = c(runif(500), runif(500, min = 0, max = 2))
)
```

2. What data type is `group`? Find this two ways. First, use `glimpse()` and read the data type off the print out. Second, `pull()` and `typeof()`.[3].

3. Use `mutate()` to change groups data type to numeric. Remember, you need to assign the output to a name to "capture" the change.

4. Use `summarize()` to find the mean value of `x` for the whole population.

5a. Use `group_by()` and `summarize()` to find the mean of `x` for each group.

5b. Use `group_by()` and `summarize()` to find the standard deviation of `x` for each group.

6. Challenge / Preview: Can you normalize your data while it's in a tibble?

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

[3] `our_data %>% pull(group) %>% typeof()`