Lab Session 2: Vectors and Data Types

Ari Anisfeld

8/31/2020

We expect you to review the `class 2` material, here prior to lab. If you find yourself in lab without R installed, you can use google colab colab.fan/r) to run R in a notebook environment. You'll lose some of the RStudio functionality, but it'll work.

TODO– directions on Rmd | Script

Warm-up: Vector creation.

1. In the lecture, we covered `c()`, `:`, `rep()`, `seq()`, `rnorm()`, `runif()` among other ways to create vectors. Use each of these functions once as you create the vectors required below.

    a. Create an integer vector from seven to seventy.
    b. Create a numeric vector with 60 draws from the `r`andom `unif`orm distribution
    c. Create a character vector with the letter "x" repeated 1980 times.
    d. Create a character vector of length 5 with the items "Nothing" "works" "unless" "you" "do". Call this vector `angelou_quote` using `<-`.
    e. Create a numeric vector with 1e4 draws[1] from a standard `norm`al distribution.
    f. Create an integer vector with the numbers 0, 2, 4, … 20.

2. Run this code and explain why we get an error.

```r
# make sure you followed direction in part d above.
sum(angelou_quote)
```

3. If we want `angelou_quote` to be a single string, we can use `paste0`.

```r
paste0(angelou_quote, collapse = " ")
```

    a. We gave collapse the argument `" "` i.e. a character string that is a blank space. Try a different character string.

4. `paste0` (or it's `tidyverse` synonym `str_c`)[2] .

```r
paste0(angelou_quote, ".com")
paste0(angelou_quote, c("!", "!", "?", " :(", "!!"))
```

    a. Explain to your partner what paste0 is doing.

5. Common error alert. Run the following code and explain why it throws an error.

```r
c(1, 2) + c(1 2)
```

[1] This is scientific notation. Try `1e4 - 1 + 1` in the console.

[2] `tidyverse` synonyms are usually preferrable since they have fewer quirky behaviors. For example, try `str_c(c("bob", NA, "maya"), "@gmail.com")` vs `paste0(c("bob", NA, "maya"), "@gmail.com")`

This is an example where the error is not so helpful. I get this one a lot, because I forget to put a comma where it should be.

Calculating Mean and Standard Deviation with vectors

Is the coin fair?

In this exercise, we will calculate the mean of a vector of random numbers. To get started, we'll generate some fake data using built-in random[3] sampling functions. Let's start by flipping coins.

```
(coin_flips <- sample(c("Heads", "Tails"), 10, replace = TRUE))
```

```
##  [1] "Tails" "Heads" "Heads" "Heads" "Heads" "Heads" "Tails" "Heads" "Tails"
## [10] "Tails"
```

`sample()` is a function that requires two arguments.

- In the first position, we have a vector of any type. We sample *from* this vector.
- In the second position, we have `size` which is the number of items to choose.

If we want to have independent draws from our sampling vector, we say `replace = TRUE`. By default `replace` is `FALSE`.

1. Run the following code and get an error.

   a. Interpret the error, i.e. explain
   b. Adjust the code so that you simulate 100 independent die rolls.

   ```
   die_rolls <- sample(c(1, 2, 3, 4, 5, 6), 100)
   ```

2. In my coin-toss simulation above, I sample from a character vector. Doing so, makes it easier to interpret the outcome, but difficult to do stuff with the results. Replace the characters with 1 and 0. Now, you'll be able to do math, but the results are more abstract. You can choose whether 1 represents heads or tails, just be consistent. Collect samples of size 10, 1000 and 1000000.[4]

   ```
   # replace ... with suitable code
   ten_flips <- ...
   thousand_flips <- ...
   million_flips <- ...
   ```

   a. What data type are your `xxx_rolls` vectors?
   b. Use `sum()` on your vectors. What does this represent?
   c. Use `length()` on your vectors to verify the vectors are the right length. What does this represent?

3. A fair coin assigns equal probability to heads and tails. Thus, the probability of heads or tails is 50 percent or .5. We can run experiments or simulations to see if our "coins" are fair. In particularly, we can calculate an estimate of the probability of heads by computing estimated probability heads $= \hat{p}(\text{heads}) = \frac{\text{n heads}}{\text{n flips}}$. The estimated probability is often called $\hat{p}$. Use the starter code to calculate the estimated probability of heads from your `ten_flips` sample.

   ```
   n_heads <- ...
   n_flips <- ...
   p_hat_ten <- n_heads / n_flips
   ```

[3] Techinically, "pseudo-random", but who's asking.

[4] Note: you can use scientific notation `1e6` is short for 1 with 6 zeros.

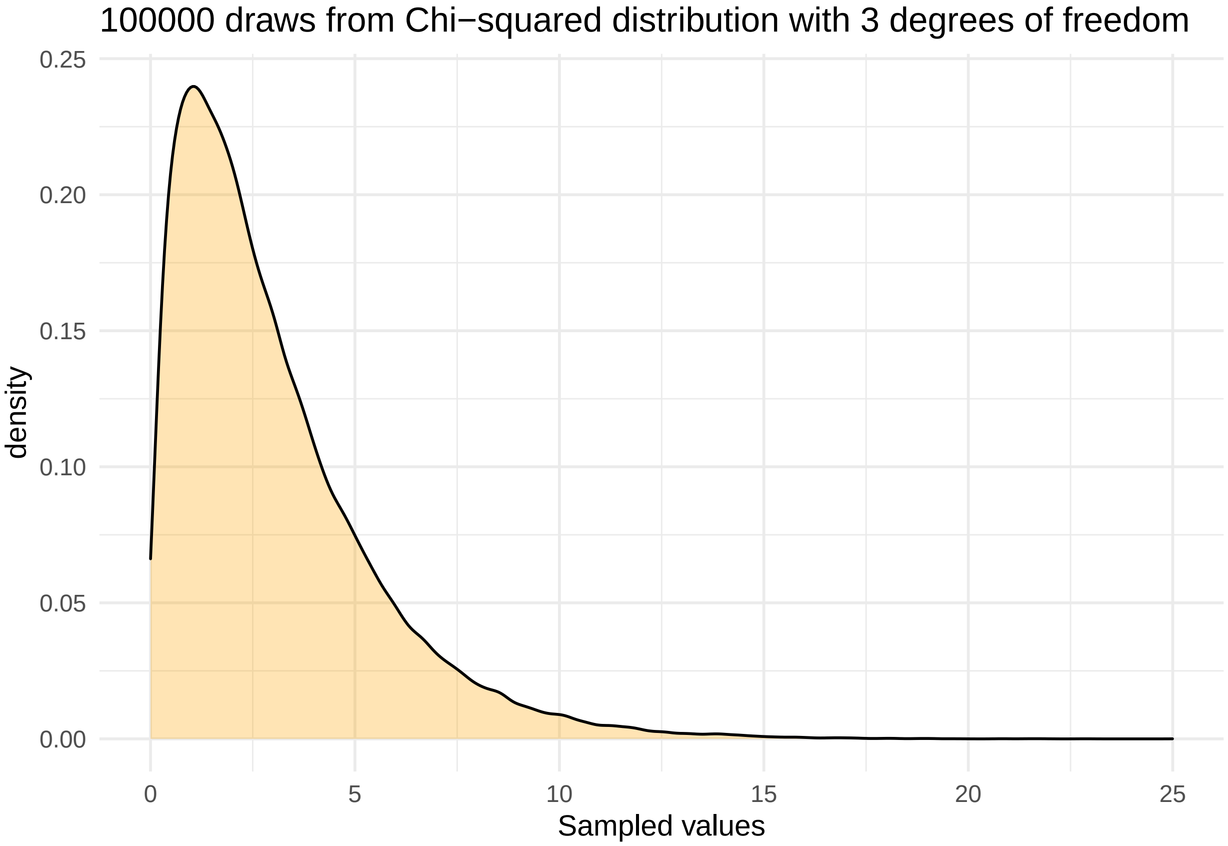
4. Repeat the code to find estimated probability of heads from your `thousand_flips` sample and `million_flips` sample.[5]

a. Run all the code a few times. Notice that the random number generator will give a different sequence of flips each time.
b. What do you notice about the estimated probabilities as the sample size gets larger? (This is an example of the "Law of Large Numbers")

1. We had you calculate the estimated probability with `sum() / length()`. R also has a function `mean()` built in. Simplify the computation for `p_hat_xxx` by using `mean()`.

A new distribution.

Now we are going to take random samples from a chi-squared distribution with 3 degrees of freedom. Do not worry about what the distribution's name means, but be aware of that it looks something like the picture below. It's possible–but highly unlikely–to get values up to `Inf`, which is R for infinity.



We are going to calculate the mean, variance and standard deviation of the distribution using vectors in three different ways.

1. First, we'll do it "by hand". The formula for sample variance is $Var(x) = \frac{\sum(x-\bar{x})^2}{n-1}$. where
   - $\bar{x}$ is the sample mean.
   - n is the sample size and
   - $\sum$ means we add up

[5] In the fall, we'll discuss ways to write this type of code more efficiently without copy paste.

```r
# fill in the ... with appropriate code.
x <- rchisq(100000, 3)

# this one should be straight forward!
# (See what we did with coin flips)
x_bar <- ...
n <-  ...
# The formula in R will be exactly the same as the
# fomula in math thanks to vectorization!
# If you aren't sure the code will work the way you want
# try with a simpler x. x <- c(1, 0, 1, 1)
var_x <- sum(...) / ...
```

2. Standard deviation is the square root of Variance, i.e. $sd(x) = \sqrt{Var(x)}$. Calculate the standard deviation.[6]

3. Now, we'll check your work using built in R functions. To calculate variance use `var()`. To calculate standard deviation use `sd()`. Try them out. If you disagree with your previous results, it's most likely a coding error in the definition of `var_x`.[7]

4. Finally, we can do this in a `tibble` setting and use summarize. You may need to load a package.[8] Using a tibble provides two services 1) the results print as an organized table. 2) We can do further `tidyverse` processing with it.

```r
# replace the ... with suitable code.
tibble(x = ... ) %>%
  summarize(mean = mean(x),
            variance = ...,
            `standard deviation` = ...)
```

5. Copy your code from the previous problem, but replace `summarize` with `mutate`. Explain the result to your group.

Challenge problems

1. Run the code below. The resulting graph shows three chi-sq distribtions determined by their degrees of freedom.

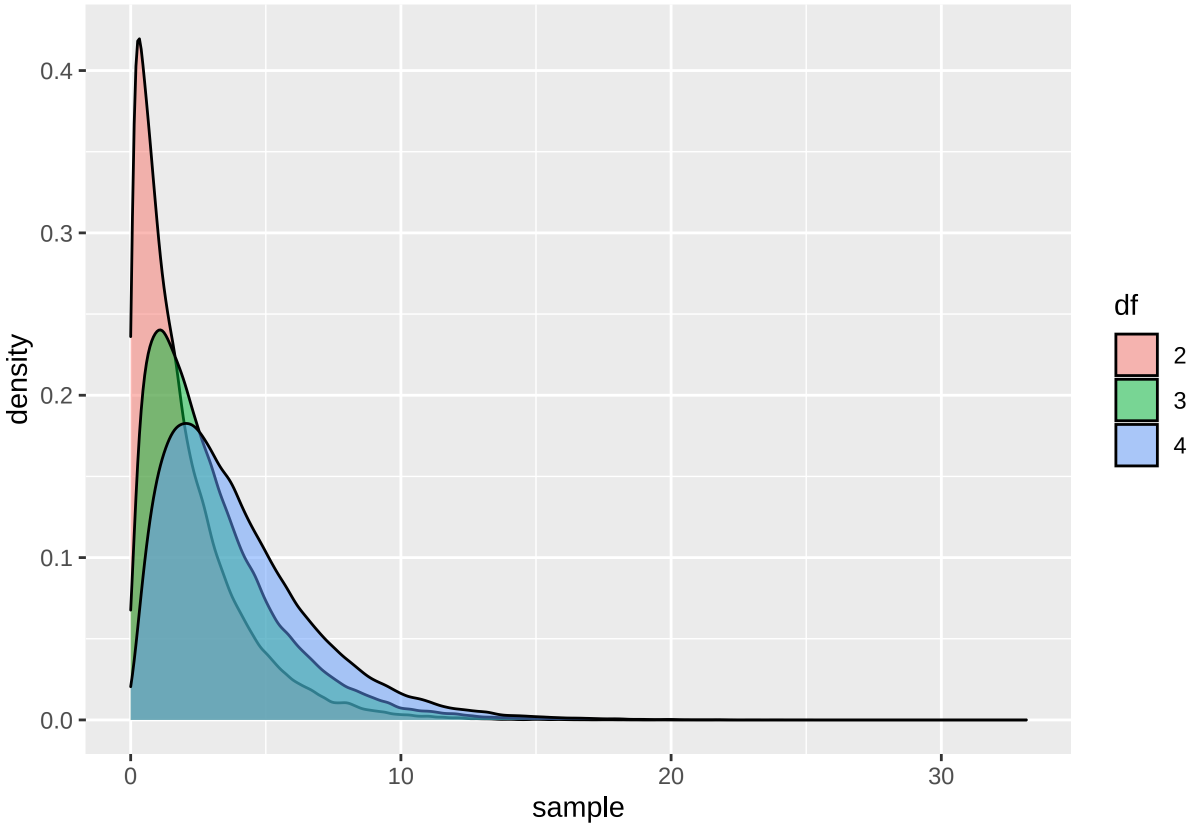
```r
chi_sq_samples <-
 tibble(x = c(rchisq(100000, 1) + rchisq(100000, 1),
             rchisq(100000, 3),
             rchisq(100000, 4)),
        df = rep(c("2", "3", "4"), each = 1e5))

chi_sq_samples %>%
  ggplot(aes(x = x, group = df, fill =df)) +
  geom_density( alpha = .5) +
  labs(fill = "df", x = "sample")
```

[6] Hint: we have the function `sqrt()`

[7] The most common errors are about where you put your parentheses. The second most common error is where you put the power i.e. ^.

[8] Hint: it's the `tidyverse`.



a. How many rows are in the tibble? Explain how the code that defines `x` and the code that defines `df` make vectors that are the right length.

b. Temporarily delete `each =` (keep `1e5`). Explain why the resulting graph looks the way it does. Make sure you put `each =` back.

c. David and Rohen told me that when we add two independent `chi sq` distributions with degrees of freedom df_1 and df_2 the result is a `chi sq` with df $= df_1 + df_2$. I'm not sure whether or not they're right. Adjust the graph code to provide visual evidence for or against their claim. There's an easy way to mess this up that is difficult to explain until after lesson 4. Ask your TA to check your result for you.