Lab Session 3 Solution: Vectors and Data Types

Ari Anisfeld

8/31/2020

1 Warm-up: Vector creation.

1. Create an R script or Rmd to capture your work. Load the `tidyverse` libraries.
2. 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 6 with the items "Nothing" "will" "work" "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.

```
# SOLUTION
a <- 7:70
b <- runif(60)
c <- rep("x", 1980)
angelou_quote <- c("nothing", "will", "work", "unless", "you", "do")
e <- rnorm(1e4)
f <- seq(0, 20, by = 2)
```

1. Run this code and explain why we get an error. (Make sure you did question 1.d above first!)

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

   Solution: We get an error because we can't add characters!

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

   ```
   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.

3. Try these lines of code using `paste0` (or it's `tidyverse` synonym `str_c`)[2] .

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

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

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

  a. Explain to your partner what paste0 is doing.

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

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

   This is an example where the error is not so helpful. I get this one a lot, because it's easy to forget to type a comma!

2 Calculating Mean and Standard Deviation with vectors

2.1 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 sampling functions. Let's start by flipping coins.

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

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

`sample()` is a function that takes up to four arguments. (Check out the help `?sample`)

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

1. We hope the following code will give us 100 independent die rolls (i.e. random numbers between 1 and 6), but we get an error. Run the code to reproduce the error.

   a. Interpret the error. I.e. why does the code fail?
   b. Adjust the code so that you simulate 100 independent die rolls.

```
# SOLUTION: the default is replace = FALSE which means we would need at least
# 100 numbers to get a sample of size 100.
die_rolls <- sample(c(1, 2, 3, 4, 5, 6), 100, replace = TRUE)
```

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.[3]

```
# SOLUTION
ten_flips <- sample(0:1, 10, replace = TRUE)
thousand_flips <- sample(0:1, 1000, replace = TRUE)
million_flips <- sample(0:1, 1e6, replace = TRUE)
```

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

   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. What does this represent?

   Solutions: They are integer; sum() tells how many heads there are (assuming heads = 1); length() tells us the number of die rolls.

3. A fair coin assigns equal probability to heads and tails. Thus, the probability of heads or tails is 50 percent or 0.5. 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 denoted $\hat{p}$ (read as: "p hat"). Use the starter code to calculate the estimated probability of heads from your `ten_flips` sample.

```
# SOLUTION
n_heads <- sum(ten_flips)
n_flips <- length(ten_flips)
p_hat_ten <- n_heads / n_flips
```

4. Repeat the code from part 3 to find the estimated probability of heads from your `thousand_flips` sample and `million_flips` sample.

   a. Re-run all the code from parts 2 through 4 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")

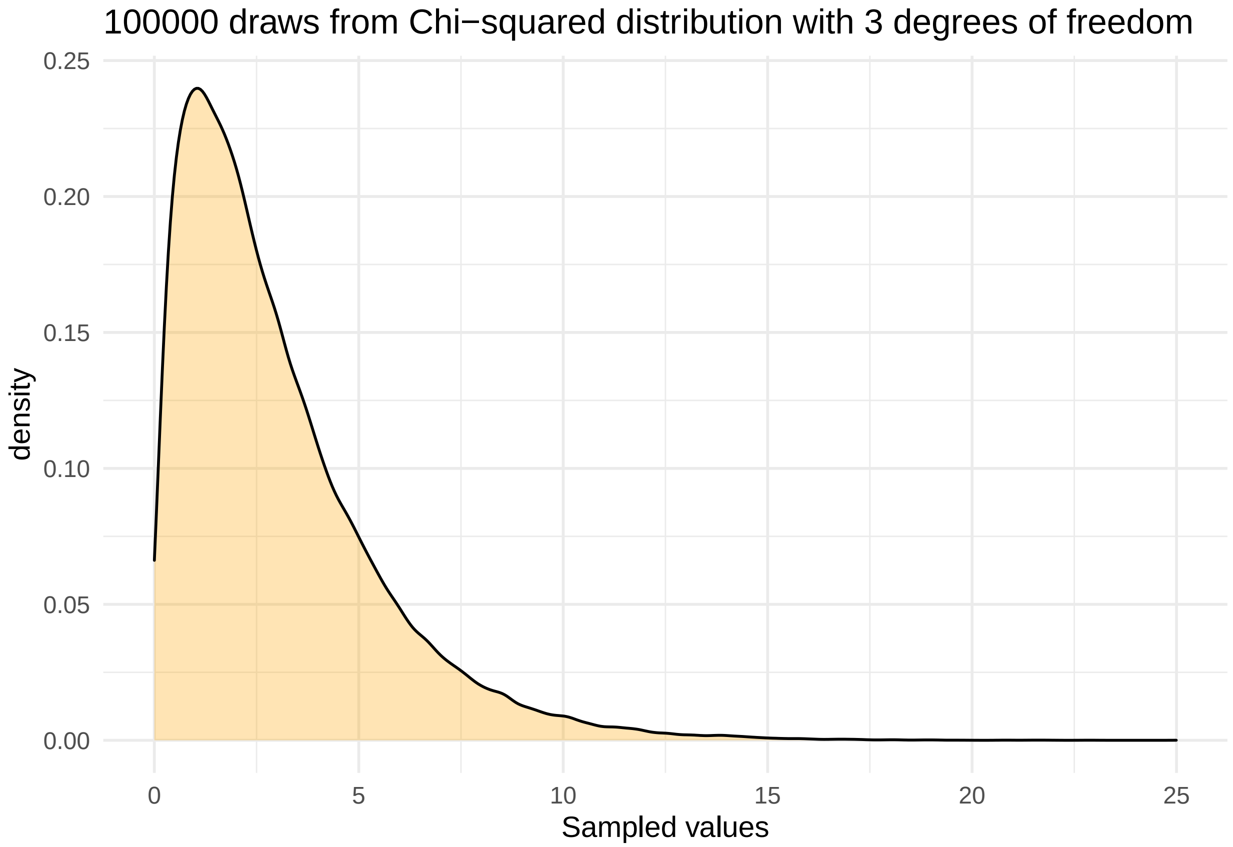
   Solution: When we use more flips, we get numbers closer to .5 for p__hat

5. 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()`.

```
# Solution: example
p_hat_ten <- mean(ten_flips)
```

2.2 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 approaching `Inf`-inity.



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

1. *way 1: 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. (`mean() = sum() / length()`)
   - n is the sample size and
   - $\sum$ means we add up

```
# SOLUTION
# 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 <- mean(x)
n <-  length(x)
# 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((x - x_bar)^2) / (n - 1)
var_x
```

```
## [1] 5.991139
```

2. Standard deviation is the square root of Variance, i.e. $sd(x) = \sqrt{Var(x)}$. Calculate the standard deviation.[4]

   ```
   # Solution
   sqrt(var_x)
   ```

   ```
   ## [1] 2.44768
   ```

3. *way 2: built-in with vectors:* 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`.[5]

   ```
   # Solution
   var(x)
   ```

   ```
   ## [1] 5.991139
   ```

   ```
   sd(x)
   ```

   ```
   ## [1] 2.44768
   ```

4. *way 2: built-in with data.frames / tibbles:* we can do this in a `tibble` (or `data.frame`) and use `summarize()` (we'll go over this soon). You will need to load a package the `tidyverse`.

Using a tibble provides two services 1) the results print as an organized table. 2) We can do further data processing with it.

```
‘‘‘r
# Solution
tibble(x = rchisq(100000, 3) ) %>%
  summarize(mean = mean(x),
            variance = var(x),
            ‘standard deviation‘ = sd(x))
‘‘‘

‘‘‘
## # A tibble: 1 x 3
##    mean variance ‘standard deviation‘
##   <dbl>    <dbl>                <dbl>
## 1  3.01     5.97                 2.44
‘‘‘
```

1. Copy your code from the previous problem, but replace `summarize` with `mutate`. Can you explain the result to your group.

   ```
   # Solution
   tibble(x = rchisq(100000, 3) ) %>%
     mutate(mean = mean(x),
            variance = var(x),
            `standard deviation` = sd(x)) %>%
     head()
   ```

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

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

```
## # A tibble: 6 x 4
##       x  mean variance ‘standard deviation‘
##   <dbl> <dbl>    <dbl>                <dbl>
## 1 4.84   3.00     5.97                 2.44
## 2 1.85   3.00     5.97                 2.44
## 3 2.66   3.00     5.97                 2.44
## 4 1.16   3.00     5.97                 2.44
## 5 6.12   3.00     5.97                 2.44
## 6 0.690  3.00     5.97                 2.44
```

mutate() adds a new column to the data with the mean, variance and sd repeated in each row

2.3 Challenge problems

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

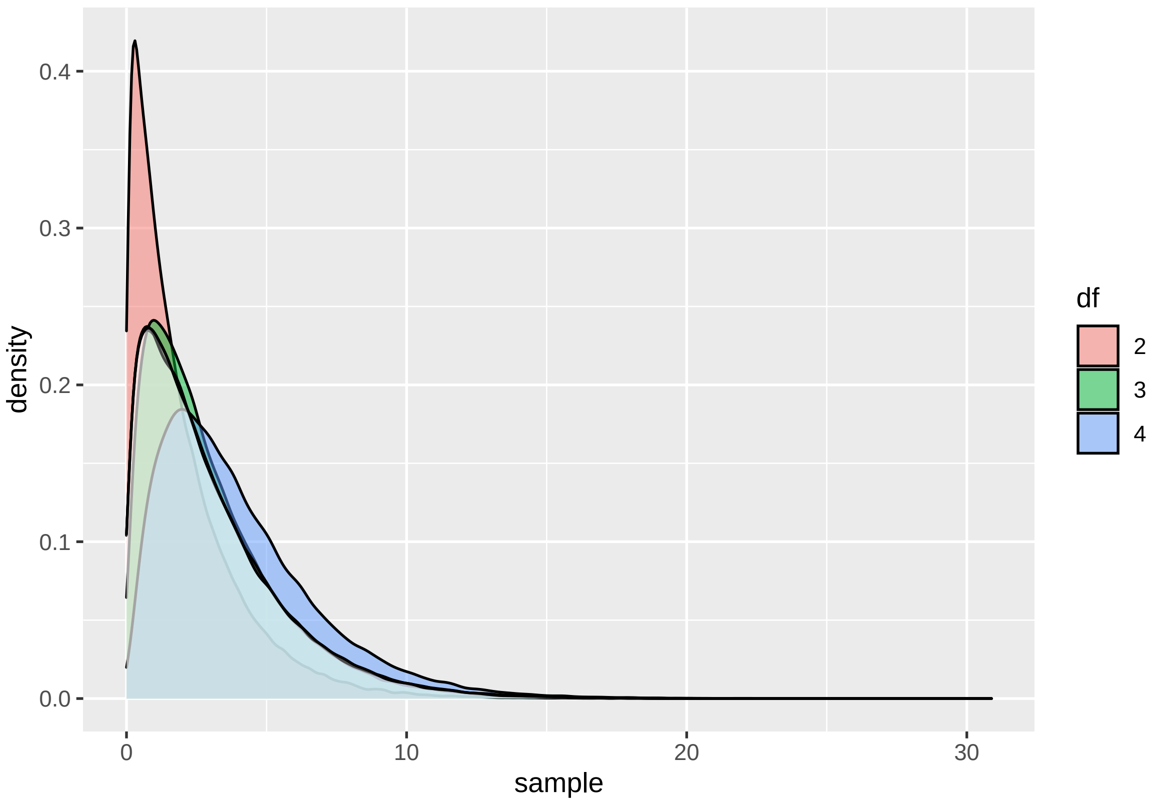
```
# I parameterize the number of draws so it's easy to experiment
# with different values.
B <- 1e5

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

mixed_chi_sq_samples <-
    chi_sq_samples %>%
        mutate(df = rep(c("2", "3", "4"), B))



chi_sq_samples %>%
  ggplot(aes(x = x, group = df, fill = df)) +
  geom_density( alpha = .5) +
  geom_density(data = mixed_chi_sq_samples,
               fill = "white", alpha = .3) +
  labs(fill = "df", x = "sample")
```



2. 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.

   Solution: there are 300000 rows (1e5 * 3). `x` is made up of 1e5 draws from the different chi sq distributions. (It may not obvious that `rchisq(100000, 1) + rchisq(100000, 1)` is still from a chi sq). And we have a `df` column that repeats "2" 100000 times and then "3" 100000 times and then "4" 100000 times.

3. Temporarily delete `each =` (keep `1e5`) and re-run the code. How does the `df` column change?

   Solution: when remove `each` we end up repeating the vector `c("2", "3", "4")` times. So the quoted numbers are mixed-up

4. Can you explain why the graph looks the way it does when you replace each?

   **Solution: First, we must explain the plotting code. When we assign a `group = df` we are telling R to make three distinct densities from our data–one for each `df`. When we include `each` in our code the rows of the data frame are not "mixed-up" so every row with df = "3" includes a draw from chi square with 3 degrees of freedom (looking at `?rchisq` shows us that the second number is `df`!).

   Now, when we remove `each` we "mix-up" the df column so rows with df = "3" now corresponds to a mixture of each of the three distributions. Notably, this mixture is almost perfectly mixed (e.g. 1 of 3 of the rows labelled "2" correspond to chi sq with 4 df and so forth). Hence, each `group = df` refers to the same mixture of random draws from different chi sq distributions. The mixture turns out to be similar to the chi sq with 3 degree of freedom. **

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.