| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks 2: Workspace and Files 3: Sequences of Numbers

4: Vectors 5: Missing Values 6: Subsetting Vectors

7: Matrices and Data Frames 8: Logic 9: Functions

10: lapply and sapply 11: vapply and tapply 12: Looking at Data

13: Simulation 14: Dates and Times 15: Base Graphics

Selection: 13

| | 0%

| One of the great advantages of using a statistical programming language like R is its vast collection

| of tools for simulating random numbers.

...

|=== | 3%

| This lesson assumes familiarity with a few common probability distributions, but these topics will

| only be discussed with respect to random number generation. Even if you have no prior experience with

| these concepts, you should be able to complete the lesson and understand the main ideas.

...

|====== | 6%

| The first function we'll use to generate random numbers is sample(). Use ?sample to pull up the

| documentation.

> ?sample

| Keep working like that and you'll get there!

|========= | 9%

| Let's simulate rolling four six-sided dice: sample(1:6, 4, replace = TRUE).

> sample(1:6, 4, replace = TRUE)

[1] 1 4 4 4

| That's correct!

|============ | 12%

| Now repeat the command to see how your result differs. (The probability of rolling the exact same

| result is (1/6)^4 = 0.00077, which is pretty small!)

> sample(1:6, 4, replace = TRUE)

[1] 2 2 3 4

| Nice work!

|=============== | 16%

| sample(1:6, 4, replace = TRUE) instructs R to randomly select four numbers between 1 and 6, WITH

| replacement. Sampling with replacement simply means that each number is "replaced" after it is

| selected, so that the same number can show up more than once. This is what we want here, since what

| you roll on one die shouldn't affect what you roll on any of the others.

...

|================== | 19%

| Now sample 10 numbers between 1 and 20, WITHOUT replacement. To sample without replacement, simply

| leave off the 'replace' argument.

> sample(1:20, 10)

[1] 16 12 7 6 8 9 17 20 15 1

| Excellent job!

|===================== | 22%

| Since the last command sampled without replacement, no number appears more than once in the output.

...

|======================== | 25%

| LETTERS is a predefined variable in R containing a vector of all 26 letters of the English alphabet.

| Take a look at it now.

> LETTERS

[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y"

[26] "Z"

| You are doing so well!

|=========================== | 28%

| The sample() function can also be used to permute, or rearrange, the elements of a vector. For

| example, try sample(LETTERS) to permute all 26 letters of the English alphabet.

> sample(LETTERS)

[1] "K" "P" "J" "C" "D" "M" "S" "O" "X" "U" "Z" "I" "R" "A" "N" "W" "Q" "B" "L" "F" "E" "V" "H" "T" "G"

[26] "Y"

| You got it right!

|============================== | 31%

| This is identical to taking a sample of size 26 from LETTERS, without replacement. When the 'size'

| argument to sample() is not specified, R takes a sample equal in size to the vector from which you are

| sampling.

...

|================================= | 34%

| Now, suppose we want to simulate 100 flips of an unfair two-sided coin. This particular coin has a 0.3

| probability of landing 'tails' and a 0.7 probability of landing 'heads'.

...

|==================================== | 38%

| Let the value 0 represent tails and the value 1 represent heads. Use sample() to draw a sample of size

| 100 from the vector c(0,1), with replacement. Since the coin is unfair, we must attach specific

| probabilities to the values 0 (tails) and 1 (heads) with a fourth argument, prob = c(0.3, 0.7). Assign

| the result to a new variable called flips.

> flips <- sample(c(0, 1), 100, replace = TRUE, prob = c(0.3, 0.7))

| You are doing so well!

|======================================= | 41%

| View the contents of the flips variable.

> flips

[1] 1 0 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1

[51] 1 1 0 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 0

| Excellent work!

|========================================== | 44%

| Since we set the probability of landing heads on any given flip to be 0.7, we'd expect approximately

| 70 of our coin flips to have the value 1. Count the actual number of 1s contained in flips using the

| sum() function.

> sum(flips)

[1] 74

| You nailed it! Good job!

|============================================= | 47%

| A coin flip is a binary outcome (0 or 1) and we are performing 100 independent trials (coin flips), so

| we can use rbinom() to simulate a binomial random variable. Pull up the documentation for rbinom()

| using ?rbinom.

> ?rbinom

| You're the best!

|================================================ | 50%

| Each probability distribution in R has an r\*\*\* function (for "random"), a d\*\*\* function (for

| "density"), a p\*\*\* (for "probability"), and q\*\*\* (for "quantile"). We are most interested in the r\*\*\*

| functions in this lesson, but I encourage you to explore the others on your own.

...

|================================================== | 53%

| A binomial random variable represents the number of 'successes' (heads) in a given number of

| independent 'trials' (coin flips). Therefore, we can generate a single random variable that represents

| the number of heads in 100 flips of our unfair coin using rbinom(1, size = 100, prob = 0.7). Note that

| you only specify the probability of 'success' (heads) and NOT the probability of 'failure' (tails).

| Try it now.

> rbinom(1, size = 100, prob = 0.7)

[1] 71

| You are amazing!

|===================================================== | 56%

| Equivalently, if we want to see all of the 0s and 1s, we can request 100 observations, each of size 1,

| with success probability of 0.7. Give it a try, assigning the result to a new variable called flips2.

> flips2 <- rbinom(100, size = 1, prob = 0.7)

| Perseverance, that's the answer.

|======================================================== | 59%

| View the contents of flips2.

> flips2

[1] 0 1 1 1 0 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1

[51] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0

| You got it!

|=========================================================== | 62%

| Now use sum() to count the number of 1s (heads) in flips2. It should be close to 70!

> sum(flips2)

[1] 74

| All that hard work is paying off!

|============================================================== | 66%

| Similar to rbinom(), we can use R to simulate random numbers from many other probability

| distributions. Pull up the documentation for rnorm() now.

> ?rnorm

| Perseverance, that's the answer.

|================================================================= | 69%

| The standard normal distribution has mean 0 and standard deviation 1. As you can see under the 'Usage'

| section in the documentation, the default values for the 'mean' and 'sd' arguments to rnorm() are 0

| and 1, respectively. Thus, rnorm(10) will generate 10 random numbers from a standard normal

| distribution. Give it a try.

> rnorm(10)

[1] 1.3063720 -1.7483433 -1.5831615 -0.7094477 1.9121656 0.1223980 -0.0847721 1.4154465 -0.8078720

[10] -0.1628545

| You are quite good my friend!

|==================================================================== | 72%

| Now do the same, except with a mean of 100 and a standard deviation of 25.

> rnorm(10, mean = 100, sd = 25)

[1] 100.62423 98.30238 54.10093 62.93520 110.19069 84.28261 100.20019 83.21382 139.55675 85.26494

| Excellent job!

|======================================================================= | 75%

| Finally, what if we want to simulate 100 \*groups\* of random numbers, each containing 5 values generated

| from a Poisson distribution with mean 10? Let's start with one group of 5 numbers, then I'll show you

| how to repeat the operation 100 times in a convenient and compact way.

...

|========================================================================== | 78%

| Generate 5 random values from a Poisson distribution with mean 10. Check out the documentation for

| rpois() if you need help.

> ?rpois

> rpois(5, lambda = 10)

[1] 9 11 12 16 9

| Keep working like that and you'll get there!

|============================================================================= | 81%

| Now use replicate(100, rpois(5, 10)) to perform this operation 100 times. Store the result in a new

| variable called my\_pois.

> my\_pois <- replicate(100, rpois(5, 10))

| Excellent work!

|================================================================================ | 84%

| Take a look at the contents of my\_pois.

> my\_pois

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18]

[1,] 6 10 10 7 12 8 23 14 12 12 6 7 12 9 11 10 11 9

[2,] 15 10 8 13 6 9 8 9 8 12 8 11 12 11 10 6 5 10

[3,] 10 9 6 12 8 7 11 7 12 8 10 7 10 8 6 13 7 5

[4,] 6 10 9 13 8 12 6 6 7 9 13 9 7 6 6 12 15 12

[5,] 8 13 10 12 11 7 9 11 7 10 17 8 8 12 10 5 12 7

[,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34]

[1,] 12 8 8 13 7 8 13 6 13 7 8 14 7 5 12 13

[2,] 12 12 13 6 10 8 13 13 13 5 14 11 4 15 8 10

[3,] 7 8 15 8 12 10 11 10 17 14 7 4 11 8 10 7

[4,] 10 8 11 7 13 9 6 15 9 10 11 15 7 8 10 13

[5,] 9 12 8 12 10 10 9 9 14 13 9 8 10 10 10 6

[,35] [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]

[1,] 14 11 5 13 11 14 7 13 6 7 10 7 10 9 7 9

[2,] 12 12 4 11 9 13 3 16 5 7 8 7 12 9 4 11

[3,] 11 9 7 8 6 9 7 8 14 8 14 9 6 8 9 12

[4,] 9 12 8 0 12 16 10 13 11 12 10 12 9 10 8 13

[5,] 6 15 6 15 12 6 8 13 11 9 6 10 17 11 8 8

[,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62] [,63] [,64] [,65] [,66]

[1,] 14 6 9 14 11 13 7 11 15 11 12 4 6 16 9 16

[2,] 11 10 11 8 12 11 7 8 11 11 9 12 6 6 12 12

[3,] 7 5 12 11 10 17 8 10 9 11 17 6 7 9 8 14

[4,] 11 10 12 12 14 8 5 7 13 7 13 15 13 12 4 5

[5,] 8 7 10 6 9 8 13 13 11 11 12 10 7 13 12 14

[,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82]

[1,] 12 12 8 9 13 8 4 10 11 15 10 8 13 10 9 10

[2,] 6 3 12 15 7 14 10 10 10 11 10 13 9 11 17 11

[3,] 11 11 9 8 12 9 13 5 15 10 11 4 7 7 5 12

[4,] 15 9 8 9 11 12 13 16 5 10 13 13 14 16 6 8

[5,] 10 11 7 10 10 6 10 12 14 12 8 9 13 4 9 13

[,83] [,84] [,85] [,86] [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]

[1,] 9 13 9 11 8 12 5 7 9 15 12 9 11 16 12 12

[2,] 7 7 7 10 10 11 5 10 15 9 8 12 8 4 18 7

[3,] 12 13 9 8 9 13 8 5 13 6 11 15 10 5 14 8

[4,] 6 8 10 15 14 9 12 12 13 9 5 9 4 4 9 7

[5,] 18 10 9 13 11 12 10 6 16 11 15 5 5 10 11 5

[,99] [,100]

[1,] 8 11

[2,] 10 12

[3,] 14 7

[4,] 14 12

[5,] 9 7

| Excellent work!

|=================================================================================== | 88%

| replicate() created a matrix, each column of which contains 5 random numbers generated from a Poisson

| distribution with mean 10. Now we can find the mean of each column in my\_pois using the colMeans()

| function. Store the result in a variable called cm.

> cm <- colMeans(my\_pois)

| That's the answer I was looking for.

|====================================================================================== | 91%

| And let's take a look at the distribution of our column means by plotting a histogram with hist(cm).

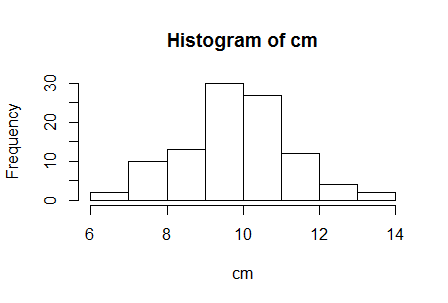
> hist(cm)

| You are really on a roll!

|========================================================================================= | 94%

| Looks like our column means are almost normally distributed, right? That's the Central Limit Theorem at

| work, but that's a lesson for another day!



...

|============================================================================================ | 97%

| All of the standard probability distributions are built into R, including exponential (rexp()),

| chi-squared (rchisq()), gamma (rgamma()), .... Well, you see the pattern.

...

|===============================================================================================| 100%

| Simulation is practically a field of its own and we've only skimmed the surface of what's possible. I

| encourage you to explore these and other functions further on your own.

...