```
#PO 841
#9/24/14
#Lab 3

#TYPES OF DISTRIBUTIONS

#Let's briefly review the different types of distributions we will come across in
 this course since I've received a few questions on this topic from students
 (needless to say, these are only the tip of the iceberg). No need to fully
 understand all the distributions yet, but since I was asked to go over them, I
 will. A probability distribution, essentially, tells us how data are arranged; it
 tells us the shape of our data.

#Binomial = models n number of independent Bernoulli trials (e.g., flipping a coin
 n number of times, approving or disapproving of Obama, being right about something
 or being wrong)
hist(rbinom(1000000,10,.8))

#Normal = models a sum of independent and identically distributed variables with a
 finite mean and variance (e.g., test scores, income)
hist(rnorm(1000000))

#Poisson = models the number of times an event occurs within certain time period
 (assuming we know these events occur independently and at a known average rate)
 E.g., number of texts a person receives per day, number of wars in a given time
 period, number of SCOTUS nominees in a presidential term)
hist(rpois(1000000,3))

#Chi-squared = the sum of the squares of independent, normally-distributed random
 variables; used to test whether or not an observed distribution of data is a good
 fit for a theoretical distribution.
hist(rchisq(1000000,10))

#*************************************************************************

#PROBABILITY FUNCTIONS

#R has built-in functions for dealing with probability distributions. There are
 four types of functions--the PDF, the CDF, the inverse CDF (which gives you
 quantiles,i.e., the 70th percentile), and one that gives you random draws from
 that distribution. The names of these functions take the form of the letter d, p,
 q, or r followed by the (usually abbreviated) name of the distribution. So for the
 "Claire" distribution, dclaire would give the PDF, pclaire would give the CDF,
 qclaire would give quintiles, and rclaire would give us random numbers from that
 distribution.

#Let's use a uniform distribution on the interval [1,5]. We can evaluate the PDF
 for any number.
```

```r
dunif(3,min=1,max=5) #Specify the number for which you want to evaluate the probability and the limits of the interval. Here, we're asking what is the probability of drawing a 3 from a uniform distribution ranging from 1-5.
dunif(pi,1,5) #Probabilites for all values on the interval are identical
dunif(-1,1,5) #Probability of a number outside the specified interval is, of course, 0

#In addition to evaluating the PDF at a single number, we can evaluate it at every value of a vector.

k<-0:7
k
fx.k<-dunif(k,1,5) #Specify the your vector and the interval
fx.k
#Could also specify interval this way:
fx.k<-dunif(k,min=1,max=5)
fx.k

#Plot this distribution:
plot(k,fx.k,type='p')

#We can also evaluate the CDF, and we can use the difference between the CDF at two different values to find the probability of X's being between those values.

punif(pi,1,5) #Remember, the CDF is evaluated as <=X (probability of X's being less than or equal to whatever value you specify). This tells us the probability of X's being less than or equal to pi (3.14159...)

punif(4,1,5)-punif(pi,1,5) #This tells us the probability of X's being between pi and 4 (prob of x's being less than or equal to 4 minus the probability of x's being less than or equal to pi)

#What if we wanted to know which values of X give us different percentiles (or quantiles) of our uniform distribution on [1,5]?

qunif(.5,1,5) #The 50th percentile, or median. Our answer of "3" tells us that 50% of the possible values of X are less than or equal to 3

qunif((1:10)/10,1,5) #This gives us the values of X for each decile -- the 10th percentile, 20th percentile, etc., up to the 100th.

#What if you wanted the value of X for all the quartiles? Or quintiles?
qunif((1:4)/4,1,5)
qunif((1:5)/5,1,5)


#Finally, what if we want R to give us random numbers drawn from this distribution? We need to specify how many numbers R should randomly draw, along with the interval.
```

```
numbers<-runif(100,1,5) #Randomly samples 100 numbers from a uniform distribution
 along the interval 1-5.
unique(numbers)
mean(numbers)


#EXERCISES: The functions for the binomial distribution are dbinom, pbinom, etc.
 For the normal distribution they are dnorm, pnorm, etc. Use these commands to find
 answers to the following questions:

#a). What is the probability of getting exactly 6 heads out of 10 coin flips?

dbinom(6,10,.5)

#b.) Problem from lecture: In a game of blackjack, where the deck is reshuffled
 after each hand, and the probability of winning each hand is .43, what is the
 probability of winning exactly 7 hands out of 10?

dbinom(7,10,.43)

#b-2.) What is the probability of winning either 4, 5, 6, or 7 hands out of 10?

pbinom(7,10,.43) - pbinom(3,10,.43)

#Is equivalent to:

dbinom(7,10,.43) + dbinom(6,10,.43) + dbinom(5,10,.43) + dbinom(4,10,.43)


#c.) If 15 questions on a test are all equally hard, and each one can be answered
 correctly with p = .4, how many does Bob have to get right to be in the 99th
 percentile of students?

qbinom(.99,15,.4)

#d.) You randomly select 50 Americans and ask them if they approve of Obama. You do
 this 100 different times. If Obama's nationwide approval rating is .41, what is
 the proportion of people in your simple random sample who approve of Obama?

#rbinom(number of simulations, number of observations per simulation, probability
 of success).

obama<-rbinom(100000, 50,.41)
mean(obama/50)

#e.) Standardized tests like the SAT follow a normal distribution with a mean of
 500 and a standard deviation of 125. What is the probability of scoring between a
 650 and a 750 (on one of the sections)?
```

```
#NOTE: While the parameters of a normal distribution are mean and variance, R's
 default is to ask for the standard deviation, not the variance. So while you will
 identify a normal distribution by its variance, in R you will evaluate it
 according to its SD (which is of course just the square oot of the variance).
 Sorry for any confusion in class re: this difference!

pnorm(750,500,125) - pnorm(650,500,125)

#What is the probability of scoring above a 790?

1-pnorm(790,500,125)
#OR
pnorm(790,500,125, lower.tail=FALSE)

#What score would you have to get in order to do better than 95% of students?
qnorm(.95,500,125)

#What is the mean score of a randomly-selected sample of 100 test-takers?
mean(rnorm(100,500,125))


#Now, let's assume that this year the SAT is being administered at 5 different
 sites across the country. You randomly select 100 students from each of the test
 sites and ask them their scores on the math portion.

site1<-rnorm(100,500,125)
site2<-rnorm(100,500,125)
site3<-rnorm(100,500,125)
site4<-rnorm(100,500,125)
site5<-rnorm(100,500,125)

sites<-matrix(cbind(site1, site2, site3, site4, site5), ncol=5, nrow=100)
sites

colmeans<-colMeans(sites) #Takes the mean of each column and gives you a vector of
 length 5
colmeans
mean(colmeans)

#As you can see, after randomly selecting students at each of the five test sites,
 we get mean scores that don't quite equal our expected value of 500. But if we
 kept doing this experiment -- if we randomly sampled 100 students  at 10,000
 different test sites as opposed to just 5, we would approach the expected
 population mean of 500.

#But there has to be a way to perform these experiments more quickly and
 efficiently. Which brings us to....
```

```
#*************************************************************************

#APPLY FUNCTIONS

#We want to tell R to perform a certain number of random samples/trials/experiments
 a certain umber of times. The most efficient way to do these kinds of probability
 experiments is to use the APPLY function or its variants (lapply, sapply, tapply).
 You can literally apply a function to a vast set of data  using one command.

#Let's take a simple example first. Remember the homework problem where you were
 asked to do various things to a vector (and then a matrix) of exam grades? Let's
 now assume that you have a matrix of grades on three exams and you want to
 calculate each student's average grade.

test1<-c(65,75,98,79,85,87)
test2<-c(70,78,95,84,86,86)
test3<-c(71,76,97,88,90,91)
m<-matrix(cbind(test1,test2,test3), ncol=3, nrow=6)
m

#Let's calculate each students average grade without having to index each student's
 scores.

avg<-c(mean(m[1,]), mean(m[2,]), mean(m[3,]), mean(m[4,]), mean(m[5,]),
 mean(m[6,]))
avg

#OR

avg<-apply(m, MARGIN=1, mean)
avg

m2<-matrix(cbind(m,avg), nrow=6, ncol=4)
m2

#MARGIN=1 tells R to apply the function to each ROW (in this case, each row
 represents 1 student and we want the avergae test grade for each student).
 MARGIN=2 says to apply the function to each COLUMN; you could also do MARGIN=1:2
 to indicate that R should perform the function on each element (both rows and
 colums). In this case, of course, where the function=MEAN, that would just mean
 the elements would remain as is.

#You can apply one of R's canned functions or define your own:

apply(m, 1, function(x) sum(x)/ncol(m))

#Say we want to inflate the average exam grade by 10%:
apply(m,1,function(x) ((sum(x)/ncol(m)) * .1) + (sum(x)/ncol(m)))
```

```
#Say we want to inflate the average exam grade by 10% and then give every student a
 1-point extra-credit bump.

apply(m,1,function(x,y) mean(x)*1.1 + y, y=1)

#Now round up every grade:
finalgrade<-apply(m,1,function(x,y) round(mean(x)*1.1 + y), y=1)
finalgrade

grades<-matrix(cbind(m,finalgrade), nrow=6)
grades

#The variants of the apply function (sapply, lapply, tapply) are similar in that
 the command performs a specified function for the items we want. Sapply works on a
 list or a vector of data, while lapply works on a list. A list in R is just a
 glorified vector that may contain objects other than numbers. For example:

x<-c(1,2,3)
y<-c("a","b","c","d")
z<-c(TRUE, FALSE, FALSE, TRUE, TRUE)
l<-list(x,y,z)
l
unlist(l)

#Let's use the apply function to run a probability experiment, drawing randomly
 from a certain distribution. Let's go back to the normally-distributed SAT scores
 example.

#Randomly select 100 students' scores on a standardized test with mean 500 and
 standard deviation 125, for 100 test sites around the country (i.e., repeating an
 experiment 100 times)

scores<-lapply(1:100, function(x) rnorm(100,500,125)) #We use lapply here because
 we'll be creating a list
scores
scores<-unlist(scores)
length(scores)
#Gives us a long, 10,000-length vector that includes all our randomly-drawn test
 scores

#Transform this vector into a matrix that arrays our experimental data better than
 the list did:

scores<-matrix(scores, ncol=100, nrow=100)
dim(scores)

#Now let's see how many of our randomly-selected scores were above 500:

propover500<-sum(scores>=500)/length(scores)
```

```
propover500 #This should be approximately .50, as we would expect.

#*************************************************************************
********

#LOOPS

#Loops do essentially the same thing as the apply function; I prefer using apply
 but either way can get the job done in probability experiments like the former
 example.

#Let's do the exact same thing we did in the former example -- we want to randomly
 select 100 test-takers where the exam mean is 500 and the SD is 125, and we want
 to do this 100 times.

x<-NULL #You can tell R either x=NULL or x=c(); the point is to tell R that your
 "for" loop will fill in the elements of the vector that you have left "blank."

#Let's write code that tells R that we want to perform the functions inside the
 brackets 100 times. Explicitly, we're saying that for every Xi in 1:100, we want
 to perform what's inside the brackets. You should get a vector of length = 100 as
 a result, of which we can take the mean to get the value of X, which we've defined
 as the proportion of randomly-selected test scores from a normal distribution with
 parameters 500,125 that fall at or above 500.
for(i in 1:100) {scores<-rnorm(100,500,125)
    x[i]<-sum(scores>=500)/length(scores)}
mean(x) #Should be very close to 50%

#We can obviously use the "for" loop for simpler (or more complicated) functions as
 well; just define the function you want to perform within the brackets and tell R
 on which vector you want to perform it. For example:

x<-c()
for(i in 1:100) {x[i]<-i+1}
x
mean(x)

x<-c()
for (i in 2:10) {x[i]<-i^2}
x
```