Stat 2994: Statistical Computing

Introduction to R: 7

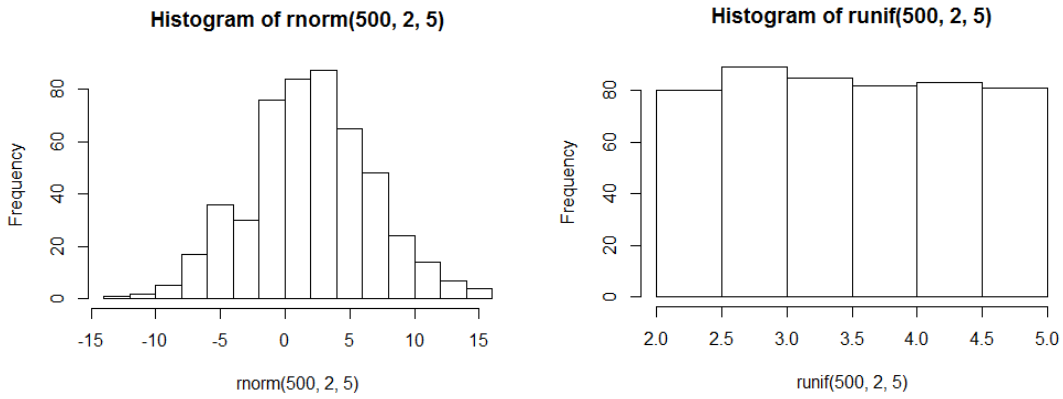
Generating Data

Often when performing statistical computations and assessing statistical tests we need to generate random data directly in R. We are able to do this for all of the well-known data distributions, like Normal, T, Chi-square, Uniform, and so on. The general form for the function of these distributions is *rnorm(), rt(), runif().* In order to use the built-in R functions to generate random data, we should know the arguments. The first argument for all of these will determine the sample size, or how many data points we would like to generate. Then we need to include the parameters for the distribution. Parameters control the shape and scale of the distribution, for instance Normal data will always need a mean and standard deviation specified (so will the T distribution).

```
> x <- rnorm(10, mean=0, sd=1)
> x
 [1] -0.614066666 -0.008102138 -0.560143207  0.653050969  1.544758969
 [6]  0.970197665  0.755382606 -0.662576014  1.110972906 -1.989826880
> mean(x)
[1] 0.1199648
> sd(x)
[1] 1.080345
```

We can use graphical functions directly with randomly generated data as well.

```
> hist(rnorm(500, 2,5))
> hist(runif(500, min=2, max=5))
```



One thing you may notice is that since these data points are random, every time they are generated you will get different values. That is helpful in the sense that we want them to be truly random, but occasionally we need to be able to recreate results for consistency or comparing two techniques. Random values are generated using a *seed*, a starting point from which to create the data. By default the seed is set by the time and date so it will be different each time you use one of the random functions. To replicate randomly generated data, the seed can be set so others can create the same values.

```
> mean(rnorm(100, 0,1))
[1] 0.104169
> mean(rnorm(100, 0,1))
[1] -0.1441567
> mean(rnorm(100, 0,1))
[1] -0.1752492

> set.seed(10)
> mean(rnorm(100, 0,1))
[1] -0.1365489
> set.seed(10)
> mean(rnorm(100, 0,1))
[1] -0.1365489
```

Exercise 1: Plot a histogram of a random sample of size 100 generated from the Exponential distribution with rate = 5.

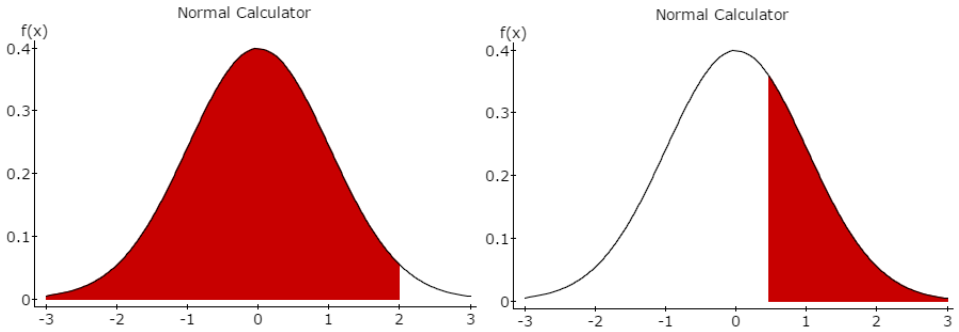
Plot a boxplot of 30 values generated from a Gamma distribution with shape parameter = 3 and scale parameter = 1.

The distribution functions in R can be used for more than generating data. We will consider the Normal for example, but the idea is the same for all of the common distributions. Along with *rnorm()*, we have *pnorm(), qnorm(),* and *dnorm().*

pnorm(q, mean = , sd = , lower.tail = TRUE/FALSE)

This function will return the probability above or below a specified cutoff point q for any Normal distribution. By default the parameters are for the Standard Normal distribution (mean = 0, sd = 1), but you can specify any values in the function. Also by default, the lower tail is calculated, this is changed to the upper tail by using the lower.tail = FALSE argument.

```
> pnorm(2, mean=0, sd=1)
[1] 0.9772499
> pnorm(0.45, mean=0, sd=1, lower.tail=FALSE)
[1] 0.3263552
```



(This isn't R output but it demonstrates what we are finding)

qnorm(p, mean = , sd = , lower.tail = TRUE/FALSE)

This version of the function is the opposite of *pnorm()*, instead of supplying the cutoff point to find the probability, we will give a percentage and the quantile is returned. Again the default for mean and standard deviation is for a Standard Normal distribution and the lower tail is calculated unless specified otherwise.

```
> qnorm(0.70, mean=10, sd=3)
[1] 11.5732
> qnorm(0.3264, mean=0, sd=1, lower.tail=FALSE)
[1] 0.4498758
```

dnorm(x, mean = , sd =)

The density of a Normal distribution can be found using this variation. The density refers to the "height" of the distribution at a single value. This is related to how the values in the distribution are concentrated. Default values are also for the Standard Normal distribution unless explicitly changes.

```
> dnorm(2, mean=0, sd=1)
[1] 0.05399097
> dnorm(0, mean=0, sd=1)
[1] 0.3989423
```

Exercise 2: Find the following probabilities and percentiles.

For a Normal distribution, $X \sim N(3.5,\ 1.2)$, find:

$P(X > 2.7) =$ _____ $P(X < 1.03) =$ _____

$P(X <$ _____$) = 0.75$ $P(X >$ _____$) = 0.142$

$P(1.6 < X < 4.6) =$ _____ $P($_____$< X <$ _____$) = 0.95$

For a Uniform distribution, $Y \sim Uniform(min = 10.5,\ max = 23.4)$

$P(Y > 15) =$ _____ $P(Y < 20) =$ _____

$P(Y <$ _____$) = 0.34$ $P(Y >$ _____$) = 0.82$

$P(13.8 < Y < 19.8) =$ _____ $P($_____$< Y <$ _____$) = 0.90$

Importing Data

There are two main types of files that are imported into R. The first is a basic text file, opens in programs like Notepad or TextEdit and has a .txt extension. Values in this type of file are often separated by a space, tab, or comma. As long as the data is the file is fairly neat and in order, the read.table() function can handle importing the data. Be sure you know the location of the file before using the function. Below is an example of importing a file bld.txt which is available in D2L.

```
> bld <- read.table("U:\\Stat 2090 Computing\\bld.txt", header=T)
> bld
   Diet Time
1     1   62
2     1   60
3     1   63
4     1   59
5     2   63
6     2   67
7     2   71
8     2   64
9     2   65
10    2   66
```

Your files will be in a different location and they will not always include a header row listing the variable names. Be sure that folders are separated by a double slash "\\", and know that Mac users will have a slightly different look to their file location.

The other type of file to import into R is an excel file. In order to get the data as easily as possible, you should save any excel file as a *.csv*, this is easily read by R while *.xlsx* files are not. The process is similar to a text file, only the function is now read.csv(). Here is the same data as above, only importing it as a .csv file from excel.

```
> blood <- read.csv("U:\\Stat 2090 Computing\\blood.csv", header=T)
> blood
  diet time
1    1   60
2    1   63
3    1   62
4    1   64
5    2   71
6    2   59
7    2   65
8    2   66
9    2   63
```

From here once we attach the data set, we can use the variable names independently throughout the R session.

```
> Diet
Error: object 'Diet' not found
> attach(bld)
> Diet
 [1] 1 1 1 1 2 2 2 2 2 2
> attach(blood)
> time
[1] 60 63 62 64 71 59 65 66 63
```