SIMULATION

Simulation is a technique in which a model of a process is developed and then experiments are conducted on the model to evaluate the existing or proposed system. Simulation is not a technique to obtain the optimum (best) solution. Instead, simulation enables decision makers to test their solutions on a model that reasonably duplicates a real process; simulation models enables decision makers to experiment with decision alternatives using a *what if* approach.

The use of simulation as a decision-making tools is fairly widespread. For example, space engineers simulate space flight in laboratories to permit future astronauts to become accustomed to working in a weightless environment. Similarly, airline pilots often undergo extensive training with simulated landings and takeoffs before being allowed to try the real thing. Universities use management games as a means of simulating business environments.

Simulation has applications across a broad spectrum of operations management problems. Their usefulness depends on the degree to which decision makers are able to successfully answer their *what if* questions. Some of the common problems in which the simulation is used are the inventory problems with probabilistic (uncertain) demand, waiting-line (queuing) problems, product design, and facility layout problems. Queuing analysis is used for many practices such as banks, amusement parks, sporting events, emergency rooms, and doctor's offices.

The reasons for the popularity of simulation include

a) Many situations are too complex to use a mathematical solution. In contrast, simulation models are often able to capture the richness of a situation.
b) Simulation models are fairly simple to use and understand
c) Simulation models allow the decision makers to experiment on a model to understand the process behavior.
d) Extensive computer software packages make it easy to use the model
e) Simulation can be used for a wide variety of situations

Monte Carlo Simulation

There are many kinds of simulation techniques. The discussion in this note will focus on the Monte Carlo simulation, which is used for simulating processes with probabilistic behavior. The technique gets its name from the famous Mediterranean resort associated with the games of chance (uncertainty). The chance (probabilistic) element is an important aspect of Monte Carlo simulation, and this approach can be used only when a process has a probabilistic, or chance, component.

Deterministic vs Probabilistic Models

To understand the probabilistic nature of Monte Carlo simulation, let's first consider a deterministic scenario where the value of every variable is known with certainty. In such cases, a simulation is not needed.

Example 1.
Consider the following deterministic problem:
Imagine you are the marketing manager for a firm that is planning to introduce a new product. You need to calculate the first year net profit from this product,

Net profit will be calculated as
Net Profit = Sales Volume* (Selling Price - Unit cost) - Fixed costs.
Fixed costs (for overhead, advertising, etc.).

Suppose the values of these variables are known with certainty to be as follows:

Sales Volume = 75,000
Selling Price = $9.67
Unit Cost = $ 6.50
Fixed Costs = $ 100,000
Net Profit = 75,000(9.67 – 6.50) – 100,000 = $137,750.

As given above, all the values are known with certainty in the deterministic models. Therefore, a simulation is not needed to solve deterministic problems.

In practice, we would not know the exact values of these variables as these variables involve uncertainty. Sales volume (in units) can cover quite a range, and the selling price per unit will depend on competitor actions. Unit costs will also vary depending on vendor prices and production experience. Sales Volume, Selling Price and Unit Cost are all uncertain variables, so Net Profit is an uncertain function**.**

For problems involving uncertainty, Monte Carlo simulation is useful because it considers all possible values using a random variable to capture the uncertainty.

Introducing Uncertainty in a Model

We generally rely on randomly generated values to consider the uncertainty in the actual problems. Using the randomly generated numbers we try to generate values representing the uncertainty.

R comes with a set of random number generators that allow you to simulate the most common probability distributions such as:

- Uniform
- Normal
- Exponential
- Binomial
- Triangular (this is not a built-in function in R)

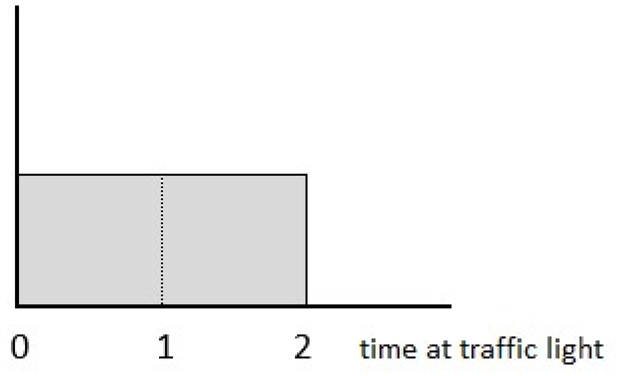
The past observed values of the variables determine which distribution to use in the simulation.

Seed in Random Number Generators

A random number generator is an algorithm based on a starting point called "seed". If you want to perform an exact replication of your program, you have to specify the seed using the function `set.seed()`. The argument of set.seed has to be an integer.

Uniform Distribution

Uniform numbers are ones that are "equally likely" to be in the specified range. An example might be the time you wait at a traffic light. This might be uniform on [0,2].



To generate random numbers from a uniform distribution you can use the `runif()` function to generate random numbers from a uniform distribution.

The syntax: `runif(n,a,b)` , where

n is the number of random numbers to be generated, a is the minimum value and b is the maximum value of the interval. The runif() functions generates continuums values.

Generate 1 random number from a uniform distribution between 0 and 2

```
runif(1,0,2)  # wait at traffic light
[1] 0.9296358
```

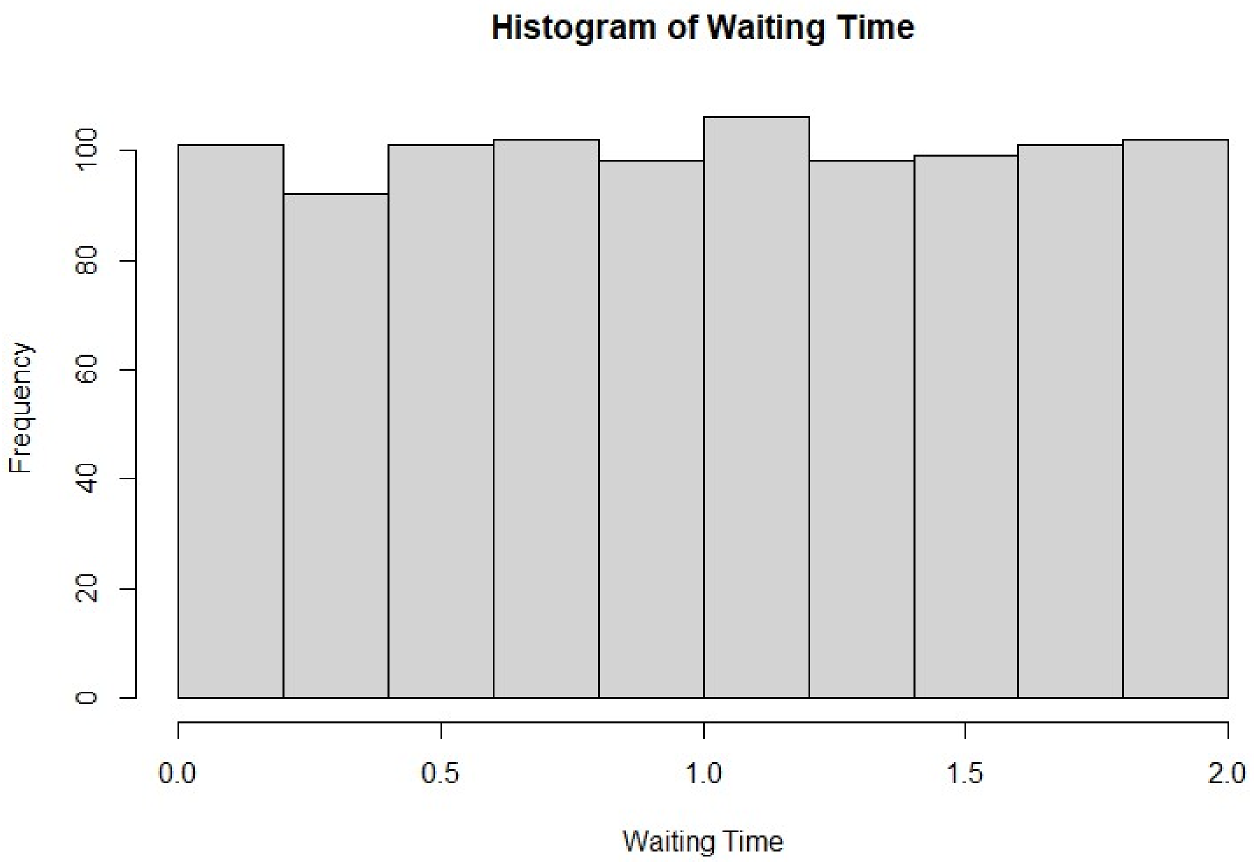
Generate 5 random numbers from a uniform distribution between 0 and 2.

```
runif(5,0,2)  # wait at 5 traffic lights

[1] 1.5385039 0.2546731 0.4109268 1.9589693 0.9906476
```

Draw the histogram of 1000 uniformly generated random numbers between 0 and 2.

```
hist(runif(1000,0,2),main="Histogram of Waiting Time",
     xlab="Waiting Time")
```

The histogram below is fairly uniform.



`sample()` function

Alternatively, you can use `sample()` to take a random sample using with or without replacements. The difference between runif() and sample() functions is that runif () function generates continuous random numbers and the sample () function generates integer numbers. You will use the sample() function to choose the numbers 1 through 6 in the dice rolling example. The sampling can be done with replacement (like dice rolling) or without replacement (like a lottery). By default, sample() function selects the numbers without replacement. You need to specify replace=TRUE if you want to sample with replacement.

`sample(range, n)`: select n random numbers from this range.

Here are some examples:

# Roll a die once
```
sample(1:6,1)
[1] 3
```

Roll the die 5 times.

```
sample(1:6,5,replace=TRUE)
[1] 5 1 6 4 6
```

You could also generate 5 rolls as follows:

```
for (i in 1:5){
  num <- sample(1:6,1,replace=TRUE)
  print(num)
}
```

You might need to use for loop in some cases. If you have an option, the first option (vectorization) is executed faster and, therefore, it is preferred.

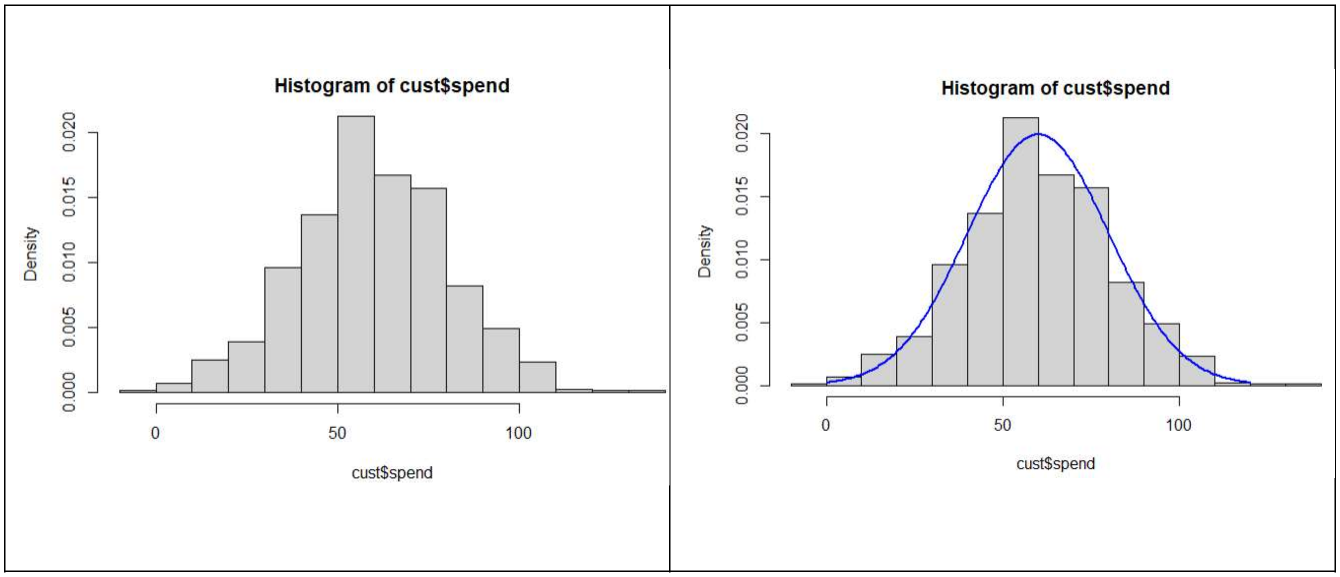
Pick 6 of 54 (a lottery)

```
sample(1:54,6)       # without replacement - default
[1] 50 33 28 44 19  4
```

Normal Distribution

The normal distribution is the most common and well known distribution. The normal distribution has two parameters: mean (μ) and a standard deviation (σ).

A grocery store manager wants to analyze customer spending. The manger has selected 120 customers randomly, recorded their spending and obtained the following histogram for the customer spending.



The customer spending can be approximated by a mean of $60 and a standard deviation of $20 as shown on the second graph. Random numbers selected from a normal distribution (μ=60, σ=10) can be used to simulate the customer spending for this grocery store.

Random numbers from a normal distribution can be generated using `rnorm()` function.

The syntax: `rnorm(n,mean,sd)` , where

n is the number of random numbers to be generated, mean and sd are the mean and the standard deviation of the normal distribution.

Generate one random number from a normal distribution with mean=60 and standard deviation = 20.

```
rnorm(1, 60, 20)
[1] 51.29222
```

If you want to round the number to 2 decimal places, you can write the code as follows:

```
round(norm(1, 60, 20), 2)
[1] 51.29
```

or

```
demand <- rnorm(1, 60, 20)
round(demand,2)
[1] 51.29
```

Generate 7 random numbers from a normal distribution with mean=60 and standard deviation = 20.

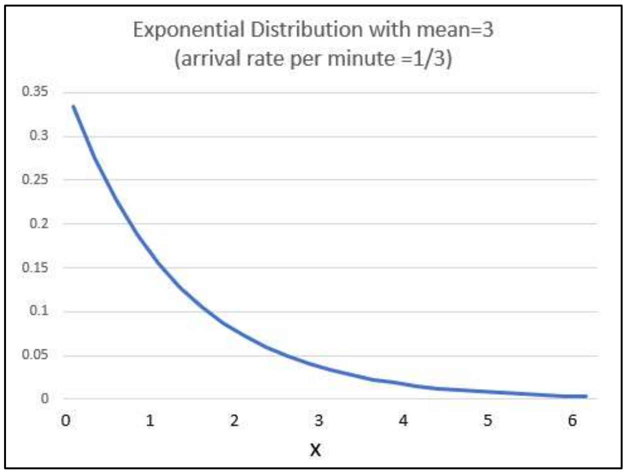
```
round(norm(7, 60, 20),2)
[1] 49.18  55.67  27.56  30.98  67.02  56.51  48.17
```

Exponential Distribution

The Exponential probability distribution describes the time between the occurrence of two random events.

Suppose the customers arrive at a checkout counter with a mean time of 3 minutes. That is, the mean interarrival time between two customers is 3 minutes. The one parameter in exponential distribution is the rate = 1/mean. This interarrival time is random with exponential distribution with an arrival rate of 1/3 per minute. (Arrival rate= mean arrival in one minute in this case).

The following graph displays this exponential distribution.



Here are some additional examples for the exponential distribution.

- If the mean life of a light bulb is 2500 hours one may think its lifetime is random with exponential distribution having mean 2500. We specify the lifetime (time between the lifetime of two bulbs) is exponential distribution with the rate being 1/2500 per hour.
- Suppose the mean checkout time of a supermarket cashier is three minutes. The service time has an exponential distribution with a rate of 1/3 service completion per minute.
- On the average, a certain computer part lasts 10 years. The length of time the computer part last is exponentially distributer with a rate of 1/10 years.
- Suppose that an average of 30 customers per hour (60 minutes) arrive at a store and the time between arrivals is exponentially distributed with a rate of 1/2 customers per minute.

Random numbers from an exponential distribution can be generated using rexp() function.

The syntax: `rexp(n,rate)`, where

n is the number of random numbers to be generated and rate is the number of occurrences per unit time.

Consider that the customers arrive at a checkout counter with a mean time of 3 minutes. The number of arrivals per minute (arrival rate) is 1/3. Generate a random number to simulate the interarrival time for this checkout counter.

```
rexp(1,1/3)
[1] 1.053933
```

Generate 5 random interarrival times for this checkout counter.

```
rexp(5,1/3)
[1] 0.8021578 4.7531355 2.6317482 5.8630475 3.6086256
```

Binomial Distribution

The binomial random numbers are discrete random numbers. The binomial distribution is used to obtain the probability of observing x successes in n independent trials where each trial results in success or failure, success with probability p.

The formula for the binomial probability mass function is

$$P(X = x) = \frac{n!}{(n-k)!\,k!}(p)^k(1-p)^{n-k}$$

Suppose you are interested in the probability of observing 5 heads when you toss a fair coin 10 times. This experiment has 10 trials and, hence, n=10. Each trial results in success or failure. The success in this trial is observing heads and the probability of success (p) is ½.

$$P(X = 5) = \frac{10!}{(10-5)!\,5!}(0.5)^5(1-0.5)^{10-} \quad = 0.2460907$$

In R, `dbinom()` function is used to calculate this binomial probability.

```
dbinom(5,10,0.5)
[1] 0.2460938
```

Random numbers from a binomial distribution can be generated using `rbinom()` function. The syntax: `rbinom(k, n, p)`, where

k is the number of random numbers to be generated, n is the number of trials, and p is the probability of success.

Consider the experiment of tossing a fair coin 10 times and you repeat this experiment 4 times. Generate random number representing the number observed in each of 4 trials.

```
rbinom(4,10,0.5)
[1] 5 7 5 6
```

Generate a vector of length 10 (generate 10 random numbers) displaying the number of successes in an experiment with 100 trials and the probability of success is 0.50.

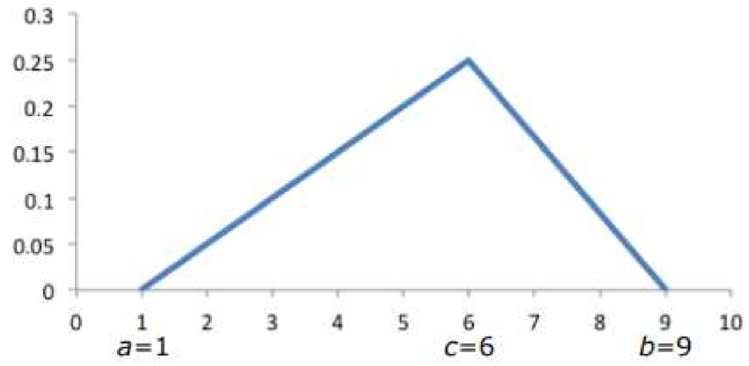
```
rbinom(10, 100, 0.5)
[1] 55 55 45 52 48 46 56 52 54 50
```

Triangular Distribution

A triangular distribution is a continuous probability distribution with a probability density function shaped like a triangle. It is defined by three values: the minimum value **a**, the maximum value **b**, and the most likely value **c**.

The triangular distribution is often used in business decision making, particularly in simulations. Generally, when not much is known about the distribution of an outcome (say, only its smallest and largest values), it is possible to use the uniform distribution. But if the most likely outcome is also known, then the outcome can be simulated by a triangular distribution.

The Figure below displays a triangular distribution with minimum=1, maximum=9 and the most likely value (mod) = 6.



Random numbers from a triangular distribution can be generated using `rtriangle()` function.
The syntax: `rtriangle(n,a,b,c)`,
where n is the number of random numbers to be selected, a is the minimum vale, b is the maximum value and c is the most likely value (mod).

The `rtriangle()` function is not a built-in function in R. To generate random numbers from a triangular distribution, you need to install "triangle" package and load the library functions.

```
install.packages("triangle")
library(triangle)
```

Generate 5 random numbers from a triangular distribution with a=1, b=9, and c=6.

```
set.seed(1)
rtriangle(5,1,9,6)
[1] 4.258887 4.858103 5.786871 7.515745 3.840295
```

Examples – Generating Random Numbers

Example 1.

Generate 10 random numbers from a uniform distribution on [0,10]. Find the minimum, maximum and mean values of these random numbers.

```
set.seed(1)
x <- runif(10,0,10)
max(x); min(x); mean(x)
[1] 0.6178627    [1] 9.446753    [1] 5.515139
```

Example 2.

Generate 100 random normal numbers with mean 100 and standard deviation 10. How many are 2 standard deviations away from the mean (smaller than 80 or bigger than 120)?

```
set.seed(1)
x <- rnorm(100,100,10)
count <- 0
for(i in 1:100){
  if (x[i] < 80 | x[i] > 120)
      count <- count + 1
  }
print(count)
[1]  3
```

Example 3.

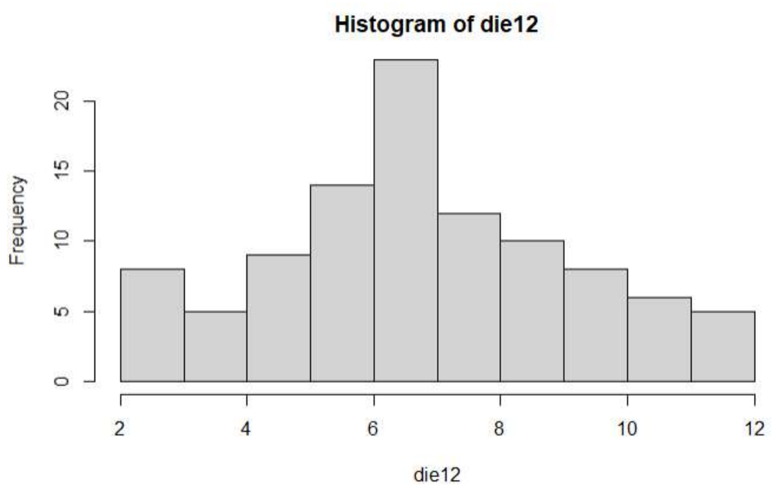
Roll a die'' 100 times. How many 6's did you see?

```
set.seed(1)
x <- sample(1:6, 50, replace=TRUE)
count <- 0
for(i in 1:50){
  if (x[i] == 6)
      count <- count + 1
  }
print(count)
[1]  9
```

Example 4.

Roll two dice 100 times. Calculate the total of two dice and histogram the totals in these 100 rolls.

```
set.seed(13)
die1 <- sample(1:6, 100, replace=TRUE)
die2 <- sample(1:6, 100, replace=TRUE)
die12 <- die1 + die2
hist(die12)
```



Example 5.

Select 6 numbers from a lottery containing 49 balls. What is the largest number? What is the smallest?

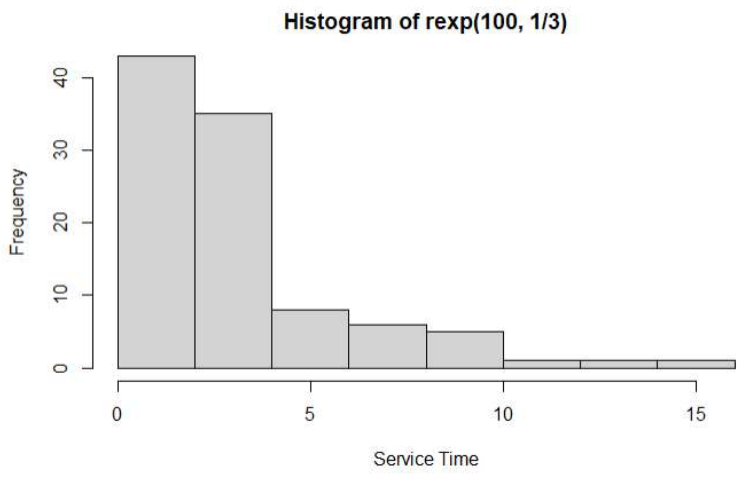
```
set.seed(1)
y <- sample(1:49, 6, replace=FALSE)
cat(" The minimum number is", min(y),"\n",
    "The minimum number is", max(y))
```

```
 The minimum number is 1
 The minimum number is 43
```

Example 6.

Suppose the mean service time to prepare the drink and the food at Starbucks is 3 minutes. Make a histogram of the random service times for 100 customers. Note the service time is exponentially distributed with a rate of 1/3 customers per minute.

```
set.seed(1)
hist(rexp(100,1/3))    # in a compact form
```



Example 7.

The market research shows that there are equal chances that the market will be **Slow**, **OK**, or **Hot** next year. Use a simulation of 40 trials to determine the market scenarios next year. Set the seed to 1.

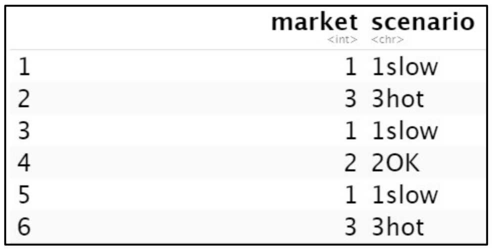
This can be simulated by selecting a number from the set [1,2,3] with 1=slow market, 2=OK market, and 3=hot market.

(a) Create a vector "market" that will contain the selected 40 numbers. Create another vector "scenario" containing the associated market scenario as "1slow", "2OK", and "3hot".

(b) Create a data frame with the columns "market" and "scenario" and display the first 6 rows of this data frame

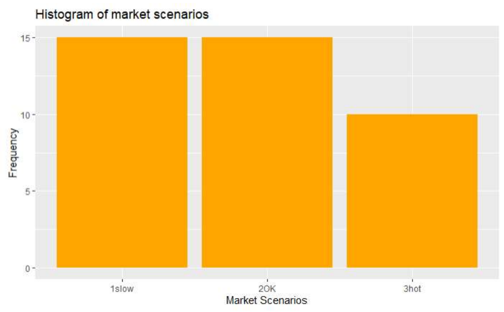
(c) Develop histogram for the "scenario" column using the ggplot2.

```
# (a) Create the vectors "market" and "scenario"
set.seed(1)
market <- sample(1:3,40, replace=TRUE)
scenario <- character(40)   # create an empty vector
scenario <- ifelse(market==1,"1slow",
ifelse(market==2, "2OK","3hot"))
# (b) Create the data frame and display the first 6 rows.
df <- data.frame(market,scenario)
head(df)
```



```
  market scenario
   <int> <chr>
1      1 1slow
2      3 3hot
3      1 1slow
4      2 2OK
5      1 1slow
6      3 3hot
```

```
# (c) Develop histogram for "scenario" using the ggplot2
library(ggplot2)
ggplot(df, aes(x = factor(scenario))) +
geom_bar(fill="orange") +
    labs(title = "Histogram of market scenarios",
             x = "Market Scenarios", y = "Frequency")
```



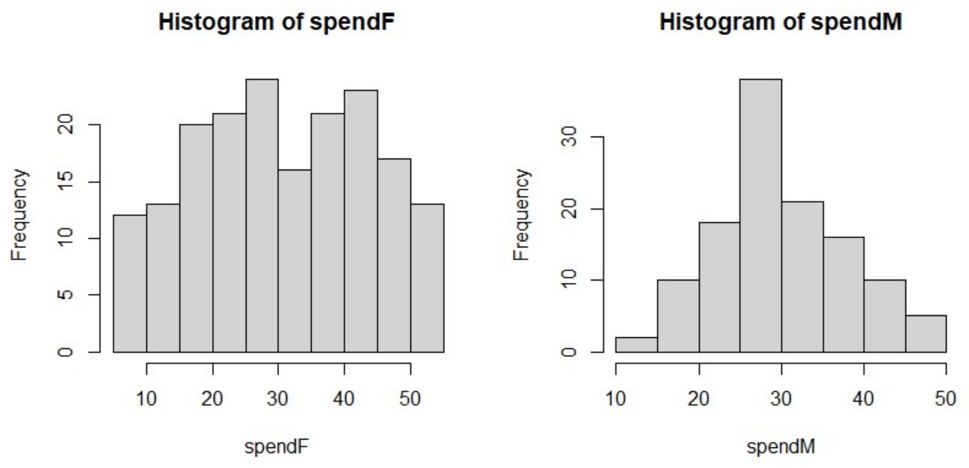
Examples _Simulation Examples

Example 8.

A manager at a local grocery store learns that 60% of her customers are females and 40% are males. On average, approximately 300 customers make a trip to the grocery store each data. For each shopping trip, the spending by the female shoppers tend to follow a continuous uniform distribution between $5 and $55. The spending by the male shoppers is normally distributed with a mean (μ) $30 and a standard deviation (σ) of $8. Develop a Monte Carlo simulation to analyze the customer spending. Simulate this system to provide the summary information about the daily spending.

Number of female shoppers = 0.60*300 = 180
Number of male shoppers = 0.40*300 = 120

```
set.seed(1)
#Determine the spending amounts by 180 female customers
spendF <- runif(180,5,55)
#Determine the spending amounts by 120 male customers
spendM <- rnorm(120, 30, 8)
#Summary information on spending
summary(spendF)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  5.654  20.530  29.954  30.712  41.596  54.634
summary(spendM)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  11.72   25.00   29.02   30.17   35.40   49.98

# Histogram
par(mfrow=c(1,2))
hist(spendF)
hist(spendM)
```



Example 9.

Your firm's production manager advises you that unit costs for one of your products may be anywhere from $5.50 to $7.50, with a most likely cost of $6.00. Develop a simulation of 10 trials to generate the random unit costs. Set the seed to 1.

This scenario will be simulated by using the triangular distribution.

```
install.packages("triangle") # if not installed before
library(triangle)
set.seed(1)
unitCost <- rtriangle(10,5.5,7.5,6.0)
(unitCost <- round(unitCost,2))   # print with 2 decimals
```

```
[1] 6.02 6.13 6.37 6.98 5.95 6.95 7.09 6.49 6.45 5.75
```

Examples – Simulation Applications

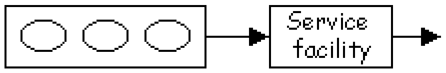
Simulation has applications across a broad spectrum of operations management problems. Some of the common problems in which the simulation is used are the inventory problems with probabilistic (uncertain) demand, waiting-line (queuing) problems, product design, and facility layout problems. Queuing analysis is used for many practices such as banks, amusement parks, sporting events, emergency rooms, and doctor's offices.

This note presents several examples and the R codes for these example problems.

Simulation for Waiting Line Problems

A waiting line system, also known as a queuing system, is exactly what it sounds like. It's when a person or object spends time waiting in a line for an activity to happen. For example, a customer may be waiting in line to deposit a check at bank, waiting in line to see a doctor at a hospital, waiting in line to check out at a cash register, or a car (driver) waiting on Kennedy Expressway to get to work in the morning on a rainy day in Chicago.

A simple (single server, single phase) waiting line problem is shown below.



Waiting times depend on the number of factors such as the arrival pattern of the customers (interarrival time), the number of service people, and the service time (time to complete service).

The summary measures used to evaluate a service system are the average time spent in service, average waiting time, number of customers waited, and the average idle time

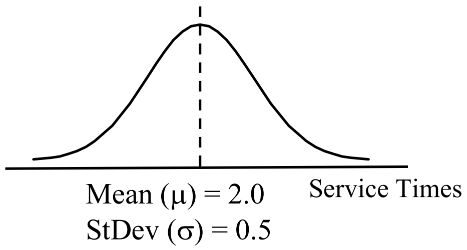
A goal of many waiting line problems is to help a firm find the ideal level of services that minimize the cost of waiting and the cost of providing the service.

Example 10.

New recruits at a military base must visit the medical lab and have their blood drawn by phlebotomist (person who draws blood samples for medical testing).

The recruits arrive at the lab with a mean time of 3 minutes. That is, the mean interarrival time between two recruits 3 minutes. This implies that interarrival times has an exponential distribution with a rate of 1/3 recruits per minute.

The service times to have their blood drawn follow a normal distribution with a mean (μ) of two minutes and a standard deviation (σ) of 0.5 minutes (30 seconds) as shown on the side.



Determine queuing and service statistic for the 300 recruits who visit the lab over a 5-hour period. Calculate the summary statistics such as the number of recruits waited, average waiting time, average time in service and the average idle time by the server (phlebotomist).

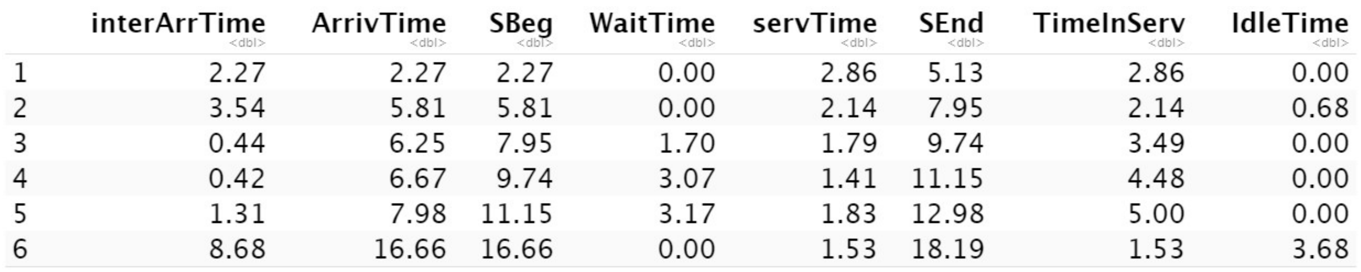
The R codes and the summary results for this example are given on the following pages.

Generate inter arrival times and service times (round to 2 decimals)
```{r}
ncust <- 300   # number of customers to simulate
set.seed(1)    # to generate the same random numbers in each run
interArrTime <- round(rexp(ncust,1/3),2) # inter arrival times
servTime <- round(rnorm(ncust,2, 0.5),2) # service times
```

Create empty vectors for the variables
```{r}
SBeg <- numeric(ncust)          # service begin empty vector
SEnd <- numeric(ncust)          # service end - empty vector
WaitTime <- numeric(ncust)      # Waiting time before service
ArrivTime <- numeric(ncust)     # Arrival time
TimeInServ <- numeric(ncust)    # Time spent in service (system)
IdleTime <- numeric(ncust)      # Idle Time by Server
```

# initialize for the 1st customer
```{r}
ArrivTime[1] <- interArrTime[1]    # Arrival time for Cust1
SBeg[1] <- ArrivTime[1]            # Service Begin for Cust1
SEnd[1] <- SBeg[1]+servTime[1]     # Service end for Cust1
WaitTime[1] <- SBeg[1]-ArrivTime[1] # Waiting time for Cust1
TimeInServ[1] <- servTime[1]        # Time spent in service - Cust1
```

# Calculate the values for customers 2 to ncust (300)
```{r}
for(i in 2:ncust)
{
  ArrivTime[i] <- ArrivTime[i-1]+interArrTime[i]
  SBeg[i] <- max(ArrivTime[i],SEnd[i-1])
  SEnd[i] <- SBeg[i]+servTime[i]
  WaitTime[i] <- SBeg[i]-ArrivTime[i]
  TimeInServ[i] <- SEnd[i]-ArrivTime[i]
  IdleTime[i] <- SBeg[i]-SEnd[i-1]
}
```

~~~
# Create data frame containing the variables
```{r}
table <- data.frame(interArrTime,ArrivTime,SBeg,WaitTime,
        servTime,SEnd,TimeInServ,IdleTime)
head(table)
~~~
```



	interArrTime	ArrivTime	SBeg	WaitTime	servTime	SEnd	TimeInServ	IdleTime
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2.27	2.27	2.27	0.00	2.86	5.13	2.86	0.00
2	3.54	5.81	5.81	0.00	2.14	7.95	2.14	0.68
3	0.44	6.25	7.95	1.70	1.79	9.74	3.49	0.00
4	0.42	6.67	9.74	3.07	1.41	11.15	4.48	0.00
5	1.31	7.98	11.15	3.17	1.83	12.98	5.00	0.00
6	8.68	16.66	16.66	0.00	1.53	18.19	1.53	3.68

```
# Calculate summaries (round to three decimals)
```{r}
avgWT <- round(mean(WaitTime),3)
avgTS <- round(mean(TimeInServ),3)
avgIT <- round(mean(IdleTime),3)
numWaited <- sum(WaitTime>0)
avgTinServ <- round(mean(TimeInServ),3)
```

Print summary results
```{r}
cat("********* Simulation Summary Results
*********","\n",
    "average time in service    =",avgTS,"\n",
    "average waiting time       =",avgWT,"\n",
    "number of customers waited =",numWaited,"\n",
    "average idle time          =", avgIT)
```

********* Simulation Summary Results *********
 average time in service   = 3.498
 average waiting time   = 1.502
 number of customers waited = 192
 average idle time   = 0.993
```
```

Example 11.

Consider a newspaper vendor selling papers on the corner. For each Sunday, the vendor must decide how many Sunday papers to buy at the wholesale price. The vendor pays $1.00 for each newspaper and sells them for $3.00 each. The shortage cost is considered to be equal to the profit foregone for not having enough papers to meet the demand. At the end of the day (Sunday), any unsold papers are sold back to the wholesaler at half of the purchasing price.

The demand for the Sunday papers is normally distributed with a mean of 200 and a standard deviation of 10 papers.

The vendor needs to decide how many newspapers to order for each Sunday. The order quantities (Q) considered are 190, 200, 210, and 220 newspapers. Simulate these ordering policies for 104 Sundays (about 2 years). Based on the total profit generated by each ordering policy, which ordering policy do you recommend to the newspaper vendor?

```
# Simulation Example11 Solution
ntrials <- 104   # simulate for 104 Sundays

# Cost Data (Per unit)
purchaseC <- 1                      # Purchase cost per unit
sellPrice <- 3                      # Selling price per unit
shortC <- sellPrice - purchaseC     # shortage cost per unit
salvageV <- 0.5*purchaseC           # salvage value per unit

# Generate random demand quantities
set.seed(1)
demand <- round(rnorm(104,200, 10), 0)

# Order quantities/policies (Q) to be evaluated
Q <- c(190, 200, 210, 220)
profitQ <- numeric(length(Q))  # initialize profit

# Evaluate different order quantities (Q)
for (i in 1:length(Q)) {       # start of the for loop

# Determine order quantity, sold, short, extra
orderQ <- rep(Q[i], times=104)
sold <- pmin(demand, orderQ)
extra <- orderQ - sold
short <- demand - sold
#Revenue and Costs (Vectors with 104 elements/Sundays)
revenue <- sold*sellPrice   # vector of revenue for each Sunday
salvageRev <- extra*salvageV    # vector of salvage revenue
purchasecost <- orderQ*purchaseC # vector of purchase cost/Sunday
shortagecost <- short*shortC     # vector of shortage cost/Sunday
sundayprofit <- revenue + salvageRev - purchasecost-
shortagecost

#Create a data frame
df <- data.frame(demand,orderQ,sold,extra,short,revenue,
      purchasecost, shortagecost,salvageRev,sundayprofit)

# Results Totals
TotalSold <- sum(sold)
TotalPurch <- sum(orderQ)
TotalSalvage <- sum(extra)
TotalShort <- sum(short)
TotalRevenue <- sum(revenue) + sum(salvageRev)
TotalCost <- sum(purchasecost) + sum(shortagecost)
TotalProfit <- sum(sundayprofit)
# Or TotalProfit <- TotalRevenue - TotalCost
AverageProfit = mean(sundayprofit)
profitQ[i] <- TotalProfit

# Print Results:
cat("\n*** Summary Results ***",
"\n-----------------------",
"\nTotal Sold    =",TotalSold,"\nTotal Purchased =",
TotalPurch,
"\nTotal Extra   =", TotalSalvage,"\nTotal Shortage =",
 TotalShort,
"\n\nTotal Revenue =",TotalRevenue,"\nTotal Cost =",
 TotalCost,
"\n-----------------------",
"\nTotal Profit    =",TotalProfit,
"\nAverage Profit  =",AverageProfit,"/Sunday")
}    # close of the for loop
```

```
cat("\nThe recommended order quantity =  ",
Q[which.max(profitQ)],
"\nWith Total Profit              =",
profitQ[which.max(profitQ)])
```

Given below is the output of this programs:

```
*** Summary Results for = 190 ***
-----------------------
Total Sold      = 19707
Total Purchased = 19760
Total Extra     = 53
Total Shortage  = 1194

Total Revenue   = 59147.5
Total Cost      = 22148
-----------------------
Total Profit    = 36999.5
Average Profit  = 355.7644 /Sunday


*** Summary Results for = 200 ***
-----------------------
Total Sold      = 20482
Total Purchased = 20800
Total Extra     = 318
Total Shortage  = 419

Total Revenue   = 61605
Total Cost      = 21638
-----------------------
Total Profit    = 39967
Average Profit  = 384.2981 /Sunday
```

```
*** Summary Results for = 210 ***
-----------------------
Total Sold      = 20826
Total Purchased = 21840
Total Extra     = 1014
Total Shortage  = 75

Total Revenue   = 62985
Total Cost      = 21990
-----------------------
Total Profit    = 40995
Average Profit  = 394.1827 /Sunday


*** Summary Results for = 220 ***
-----------------------
Total Sold      = 20895
Total Purchased = 22880
Total Extra     = 1985
Total Shortage  = 6

Total Revenue   = 63677.5
Total Cost      = 22892
-----------------------
Total Profit    = 40785.5
Average Profit  = 392.1683 /Sunday


The recommended order quantity =   210
With Total Profit              = 40995
```

Example 12.

A manufacturer of a smartphone battery estimates that the monthly demand follows a normal distribution with a mean of 500 units and a standard deviation of 20. Material cost is uniformly distributed between $7.0 and $8.50. Fixed costs are $2,700 per month, regardless of the production rate. The selling price is $15 per unit.

Simulate 1,000 trials. Set the seed to 1. Demand values need to be rounded to integers, and two decimal places for the material cost.

(a) Display the following values for the first 6 trials.

Month	Demand	Revenue	Cost	Profit
1				
2				
3				
4				
5				
6				

(b) Print the expected monthly profit and standard deviation.

(c) Print the best and worst profit scenarios for the company.

Profit Model

Monthly Values:

Revenue = Q*SellingPrice

Cost = Q*Material Cost – Fixed Cost

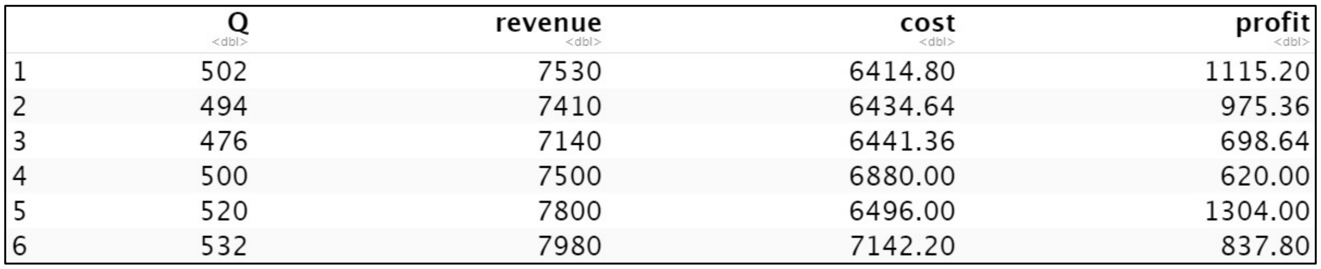
Profit = Revenue – Cost

The production quantity (Q) is equal to the demand.

The R program for the solution is given on the next pages.

~~~
Simulation Example 12 Solution
```{r}
ntrials <- 1000      # number of trials to simulate
```

Generate demand Quantities and material cost
```{r}
set.seed(1)      # to generate the same random numbers in each trial
Q <- round(rnorm(ntrials,500, 20),0)     # demand quantities
matCost <- round(runif(ntrials,7.0, 8.50),2)    # material cost
```

Selling Price and Fixed Cost
```{r}
sellPrice <-  15
fixedCost <- 2700
```

Create empty vectors for the variables
```{r}
revenue <- numeric(ntrials)
cost <- numeric(ntrials)
profit <- numeric(ntrials)
```

Calculate the revenue, cost, and profit
```{r}
revenue <- Q*sellPrice
cost <-  Q*matCost + fixedCost
profit <- revenue-cost
```

Calculate summary values
```{r}
avgProfit <- round(mean(profit),2)
stDev <- round(sd(profit),2)
```

Calculate summaries (round to three decimals)
```{r}
min(profit)
max(profit)
sum(profit<0)
```
~~~

~~~~
Create the data frame containing demand Quantity, revenue,
cost, profit and display the first 6 rows.
```{r}
df <- data.frame(Q,revenue,cost,profit)
head(df)
```
~~~~



	Q <dbl>	revenue <dbl>	cost <dbl>	profit <dbl>
1	502	7530	6414.80	1115.20
2	494	7410	6434.64	975.36
3	476	7140	6441.36	698.64
4	500	7500	6880.00	620.00
5	520	7800	6496.00	1304.00
6	532	7980	7142.20	837.80

~~~~
Print summary results
```{r}
cat("********* Simulation Summary Results *********","\n",
    "Expected monthly profit    =",avgProfit,"\n",
    "Stand Deviation of profits =",stDev,"\n\n",
    "Best profit scenario       =",max(profit),"\n",
    "Worst profit scenario      =",min(profit),"\n")
```
~~~~

```
********* Simulation Summary Results *********
 Expected monthly profit    = 921.67
 Stand Deviation of profits = 262.11

 Best profit scenario       = 1617.18
 Worst profit scenario      = 179.7
```

Example 13.

Imagine you are the marketing manager for a firm that is planning to introduce a new product. You need to calculate the first year net profit from this product, which will depend on:

- Sales volume in units
- Price per unit
- Unit cost
- Fixed costs

Net profit will be calculated as

Net Profit = Sales Volume* (Selling Price - Unit cost) - Fixed costs.

Fixed costs (for overhead, advertising, etc.) are known to be $100,000. But the other factors all involve some ***uncertainty***. Sales volume (in units) can cover quite a range, and the selling price per unit will depend on competitor actions. Unit costs will also vary depending on vendor prices and production experience.

Uncertain Variables

To build a risk analysis model, we must first identify the **uncertain variables** -- also called random variables. While there's some uncertainty in almost all variables in a business model, we want to focus on variables where the range of values is significant.

Market, Sales and Price

Based on your market research, you believe that there are equal chances that the market will be Slow**,** OK**, or** Hot**.**

- In the "Slow market" scenario, you expect to sell 70,000 units at an average selling price of $12.00 per unit.
- In the "OK market" scenario, you expect to sell 85,000 units, but you'll likely realize a lower average selling price of $10.00 per unit.
- In the "Hot market" scenario, you expect to sell 100,000 units, but this will bring in competitors who will drive down the average selling price to $8.00 per unit.

	Slow Market	OK Market	Hot Market
Sales	70,000	85,000	100,000
Price	$12.0	$10.0	$8.0

Market scenarios will be determined by randomly selecting the numbers 1, 2, and 3 from the set of [1,2,3] since each scenario is equally likely.

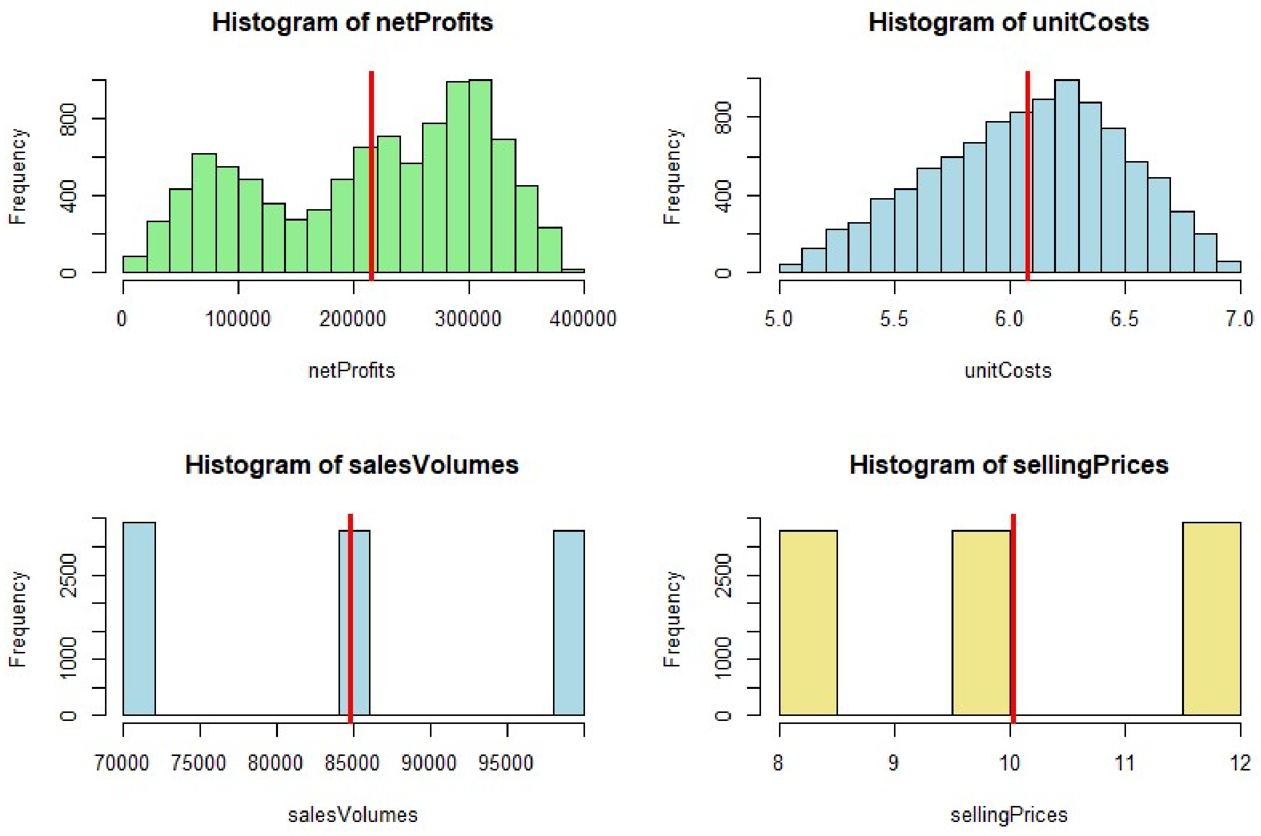
Unit Cost

Another uncertain variable is Unit Cost. Your firm's production manager advises you that unit costs may be anywhere from $5.0 to $7.0, with a most likely cost of $6.25. This scenario will be simulated by using the triangular distribution

```
# Simulation Example13 Solution - Version 1 (Using vectors)
startTime <- Sys.time()  # to calculate the CPU time
options(scipen=999) # not to have scientific notation
ntrials <-  10000
salesVolumes <- numeric(ntrials)
sellingPrices <- numeric(ntrials)
unitCosts <- numeric(ntrials)
netProfits <- numeric(ntrials)
set.seed(1)
ms <- sample(1:3,ntrials, replace=TRUE)  # market scenarios (ms)
# Determine salesVolumes and sellingPrices
salesVolumes <- ifelse(ms==1,70000,
                ifelse(ms==2,85000,100000))
sellingPrices <- ifelse(ms==1,12,
                 ifelse(ms==2,10,8))
# Create triangular distribution for unit cost and Fixed Cost
library(triangle)
unitCosts <- rtriangle(ntrials, 5, 7, 6.25)
fixedCost <- 100000

netProfits = salesVolumes * (sellingPrices - unitCosts) -
fixedCost
endTime <- Sys.time()
(timeTaken <- endTime - startTime)  # CPU time needed to solve
Time difference of 0.031003 secs
# Create a data frame containing salesVolumes, sellingPrices
# unitCosts, and netProfits and display first 6 rows
df <- data.frame(salesVolumes, sellingPrices, unitCosts,
netProfits)
head(df)
```

```
  salesVolumes sellingPrices unitCosts netProfits
1        70000            12  5.754452  337188.38
2       100000             8  6.330054   66994.60
3        70000            12  5.906920  326515.63
4        85000            10  6.316217  213121.55
5        70000            12  6.810579  263259.44
6       100000             8  6.246397   75360.32
```

```
# Display numerical summary results
summary(netProfits)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3852  122862  231975  214615  298179  389679
summary(salesVolumes)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  70000   70000   85000   84790  100000  100000
# Display visual summary results
par(mfrow=c(2,2))
hist(netProfits, col="lightgreen")
abline(v=mean(netProfits), col="red", lw=3)

hist(unitCosts, col="lightblue")
abline(v=mean(unitCosts), col="red", lw=3)

hist(salesVolumes, col="lightblue")
abline(v=mean(salesVolumes), col="red", lw=3)

hist(sellingPrices, breaks=8.5,9.5,10., col="khaki")
abline(v=mean(sellingPrices), col="red",lw=3)
```

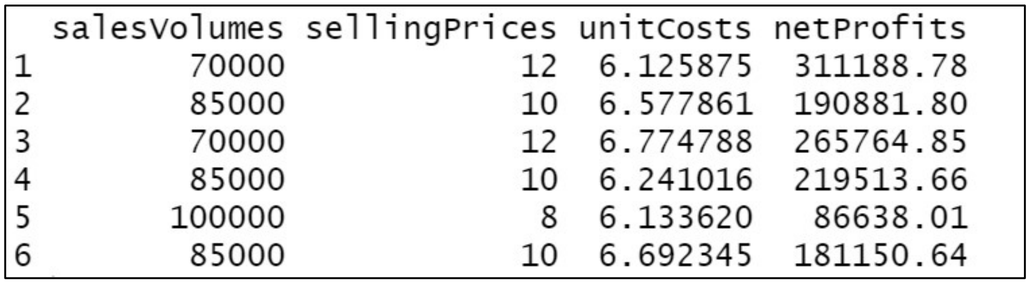


```
# Simulation Example13 Solution - Version 2 (Using for loop)
# Create random uniform integer for market scenario (ms)
# and sales volumes
startTime <- Sys.time()    # to measure CPU time
options(scipen=999)   # not to have scientific notation
ntrials <-  10000     # 10000 trials
# Create empty vectors for the variables
salesVolumes <- numeric(ntrials)
sellingPrices <- numeric(ntrials)
unitCosts <- numeric(ntrials)
netProfits <- numeric(ntrials)

# Repeat the simulation for 10000 trials
for (i in 1:ntrials) {
ms <- sample(1:3,1)
if (ms==1){      # 1=slow market
  salesVolume <- 70000
  sellingPrice <-  12
}
if (ms==2){      # 1=OK market
  salesVolume <- 85000
  sellingPrice <-  10
}
if (ms==3){      # 1=hot market
  salesVolume <- 100000
  sellingPrice <-  8
}

# Create triangular distribution for unit cost
library(triangle)
unitCost <- triangle(1, 5, 7, 6.25)
fixedCost <- 100000

netProfit =salesVolume* (sellingPrice - unitCost) - fixedCost
salesVolumes[i] <- salesVolume
sellingPrices[i] <- sellingPrice
unitCosts[i] <- unitCost
netProfits[i] <- netProfit
}
```

```
endTime <- Sys.time()   # To calculate the CPU time to solve
(timeTaken <- endTime - startTime)
```

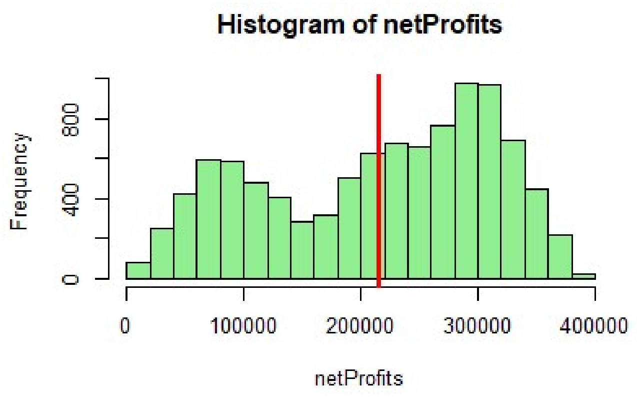
```
Time difference of 3.80528 secs
```

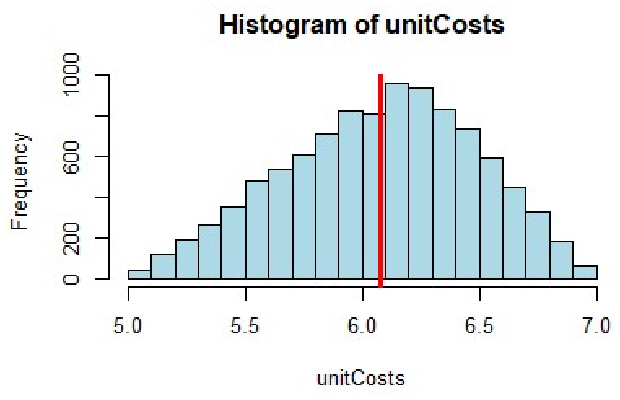
```
# Create a data frame containing salesVolumes,sellingPrices,
# unitCosts, and netProfits and display first 6 rows
df <- data.frame(salesVolumes, sellingPrices, unitCosts,
netProfits)
head(df)
```

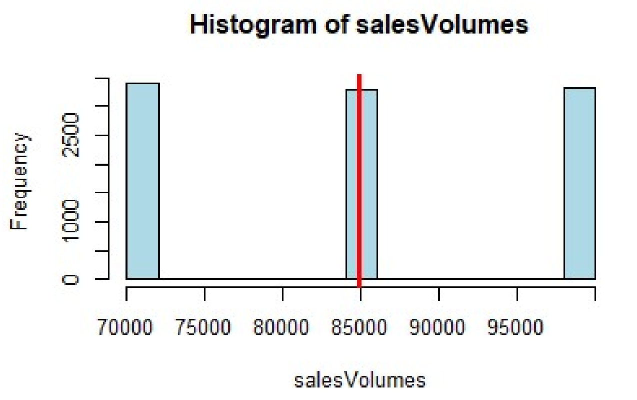


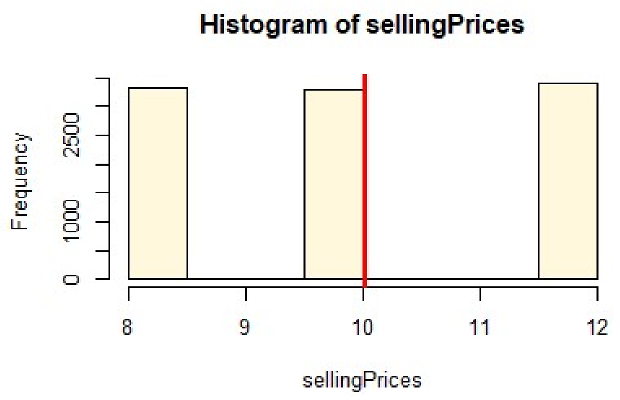
```
  salesVolumes sellingPrices unitCosts netProfits
1        70000            12  6.125875  311188.78
2        85000            10  6.577861  190881.80
3        70000            12  6.774788  265764.85
4        85000            10  6.241016  219513.66
5       100000             8  6.133620   86638.01
6        85000            10  6.692345  181150.64
```

```
# Display numerical summary results
summary(netProfits)
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1091  124138  232223  214491  297193  388770

summary(salesVolumes)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 70000   70000   85000   84859  100000  100000

# Display visual summary results
par(mfrow=c(2,2))

hist(netProfits, col="lightgreen")
abline(v=mean(netProfits), col="red", lw=3)

hist(unitCosts, col="lightblue")
abline(v=mean(unitCosts), col="red", lw=3)

hist(salesVolumes, col="lightblue")
abline(v=mean(salesVolumes), col="red", lw=3)

hist(sellingPrices, breaks=8.5,9.5,10.5, col="cornsilk")
abline(v=mean(sellingPrices), col="red",lw=3)


# See next page for the histogram.
```


Histogram of netProfits
Frequency
0
400
800
0
100000
200000
300000
400000
netProfits


Histogram of unitCosts
Frequency
0
200
600
1000
5.0
5.5
6.0
6.5
7.0
unitCosts


Histogram of salesVolumes
Frequency
0
1000
2500
70000
75000
80000
85000
90000
95000
salesVolumes


Histogram of sellingPrices
Frequency
0
1000
2500
8
9
10
11
12
sellingPrices