STATISTICAL SIMULATION

A Project Presentation By

Abraha Shaik(STA212013)

Beauty Abedin(STA212030)

Snehana Basu(STA212024)

Snehadrita Das(STA212008)

What is Statistical Simulation?

Simulation is a way to model random events, such that simulated outcomes closely match real-world outcomes. By observing simulated outcomes, researchers gain insight on the real world. For the ease of computation, we use Computer simulation. A simulation uses a mathematical description, or model, of a real system in the form of a computer program.
A simulation is the execution of a model, represented by a computer program that gives information about the system being investigated. he activities of the model consist of events, which are activated at certain points in time and in this way affect the overall state of the system. The points in time that an event is activated are ***randomized***, so no input from outside the system is required.

Why should we consider Simulating Data?

- In comparison to mathematical model Simulation technique is easier to use and it is superior technique to the mathematical analysis.
- It saves cost, time and man power because it uses random numbers generated rather than data that are collected.
- Testing of statistical models are easier with simulated data.
- Simulation models are comparatively flexible and can be changed according to the changing environments of the real situation.

Some Basic Simulations

Simulating data from Normal,Chi-square, F and t distribution

General Procedure:

- Generated random number from $U(0,1)$
- Used mathematical relations between uniform and other distributions to simulate data.
- Compared the histogram of the data generated by mathematical relations and density curves for the particular distribution.

Standard Normal Distribution : use of Box Muller Transformation

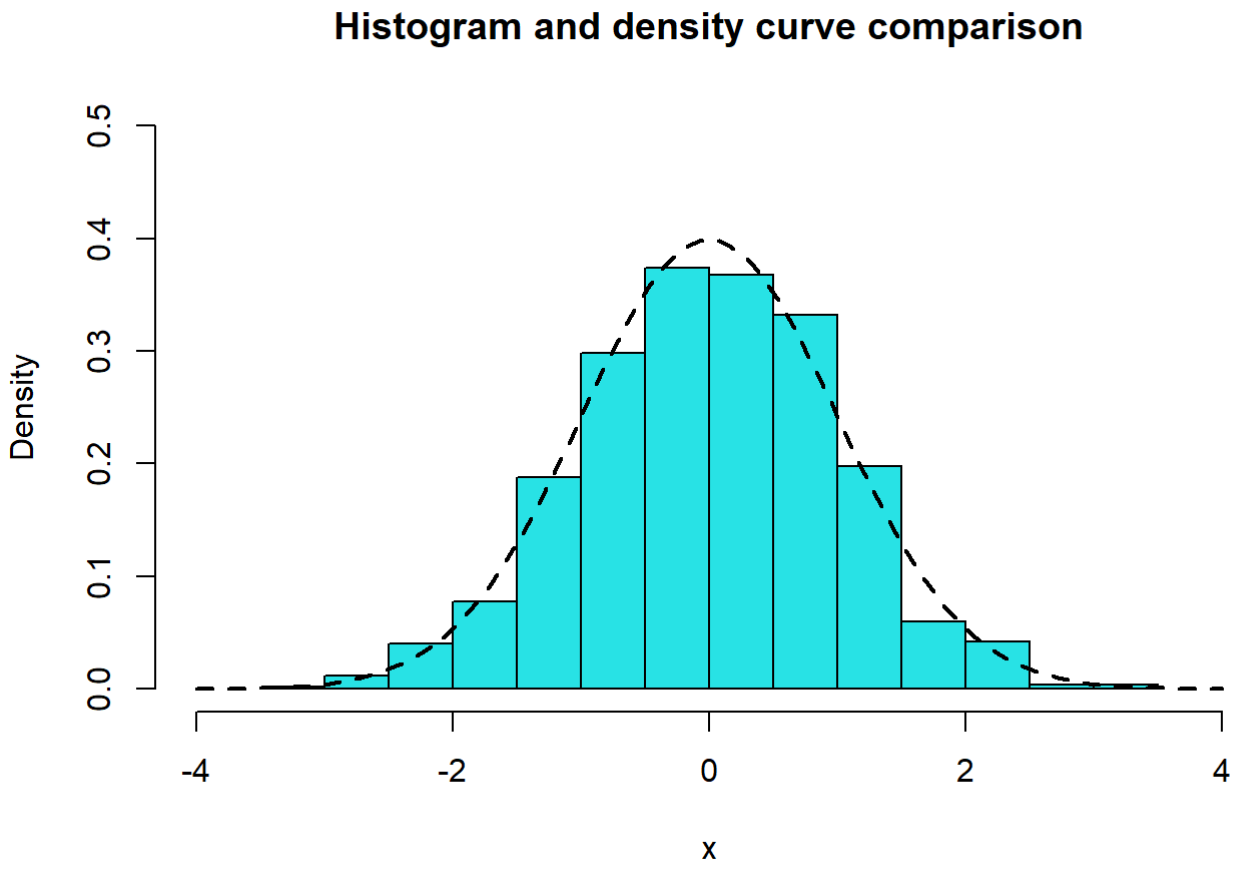
One way to achieve normal random variable simulation using a transform is with the Box–Muller algorithm, devised for the generation of $N(0,1)$ variables.

Let X_1 & X_2 are independent $U(0,1)$. So $Y = \sqrt{}((-2logX_1)\sin(2\pi X_2)) \sim N(0,1)$

```
normal=function(n){ #n being the number of random numbers we wish to generate
    y=rep(0,n)
  for(i in 1:n){
    y[i]=sqrt(-2*log(runif(1,0,1)))*sin(2*pi*(runif(1,0,1)))
  }
  return(y)
}
x=normal(1000)
head(x)
```

```
[1] -0.2503551 -0.9222167  0.8623202 -0.7209191 -1.5884140  1.8097989
```

```
hist(x,col=5,xlim=c(-4,4),ylim=c(0,0.5), main="Histogram and density curve comparison",probab
ility=TRUE)
curve(dnorm(x,0,1), lty = 2, lwd = 2, add = TRUE)
```



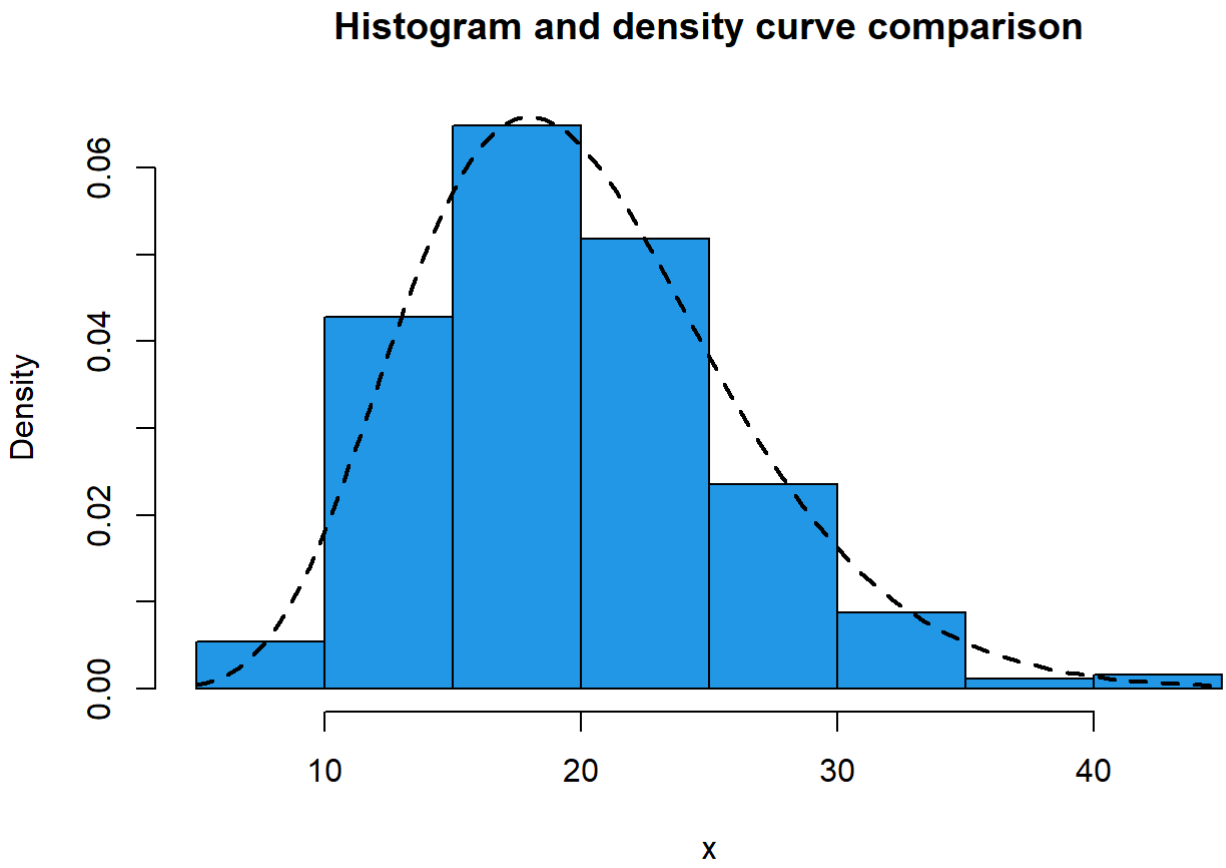
Central Chi Square Distribution with df multiple of 2

Here we have used the relation between uniform and Chi square.We know that,when $X \sim U(0,1)$, $Y = -2logX \sim \chi^2_2$

```
chisqr=function(n,d){ #n being the number of random numbers we wish to generate
  r=rep(0,n)
  set.seed(11)
  for(i in 1:n){
    u=runif(d,0,1) #d,the number of samples drawn from U(0,1),so that we have chi with df 2d
    y=-2*log(u)
    r[i]=sum(y)
  }
  return(r)
}
x=chisqr(1000,10)
head(x)
```

```
[1] 44.95206 16.20671 31.16934 25.39699 21.74935 17.92028
```

```
hist(x,col=4,main="Histogram and density curve comparison", probability=TRUE)
curve(dchisq(x, 20), lty = 2,lwd=2, add = TRUE)
```



Central t-Distribution

Here we have used the relation of Standard normal and Chi square for t distribution.We know that when X be independent $N(0,1)$ distribution & Y_1,Y_2....Y_n be independent $N(0,1)$ then

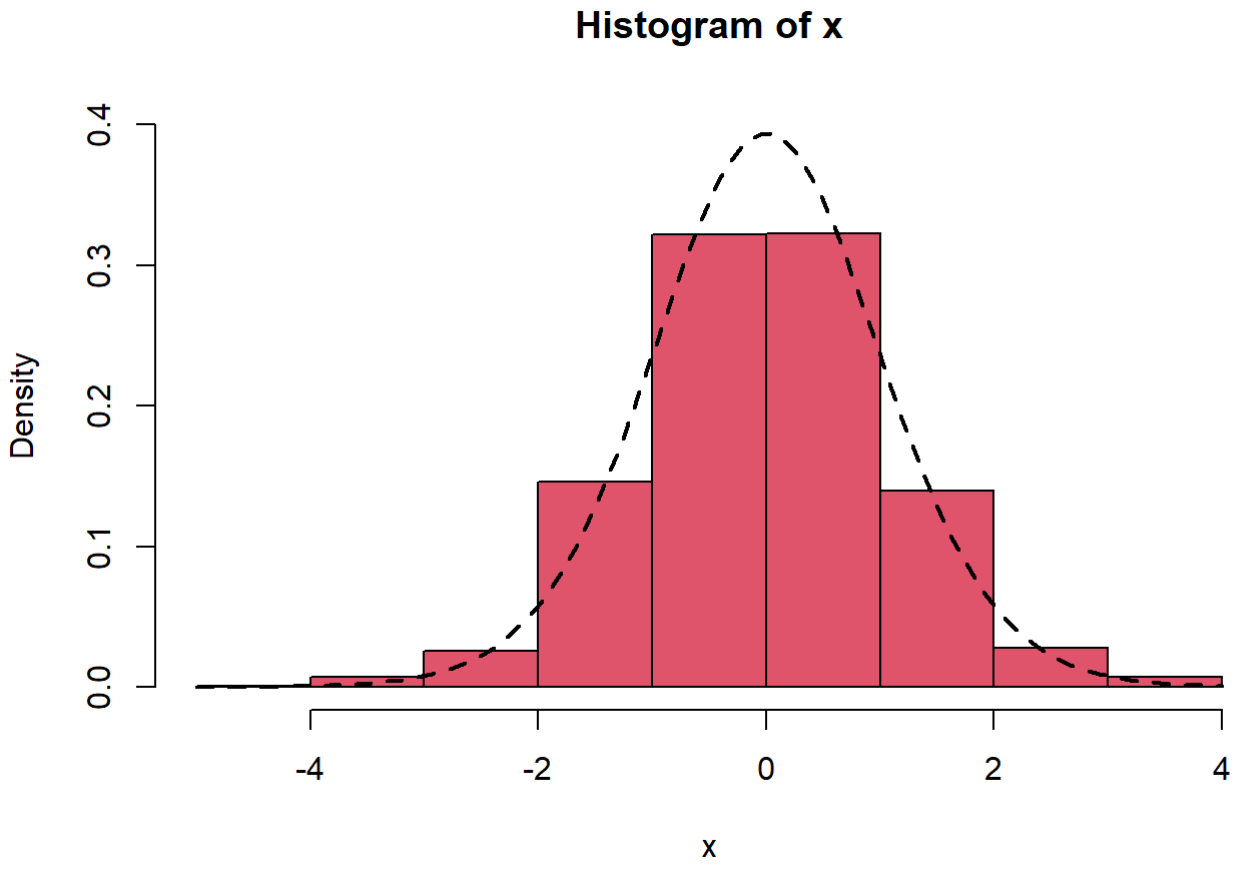
$$\sum_{i=1}^{n} Y_i^2 \sim \chi_n^2$$

and $\frac{N(0,1)}{\sqrt{(\sum_{i=1}^{n} Y_i^2/n)}} \sim t_n$. Where n is degrees of freedom

```
t_distribution=function(n,d){
  t=rep(0,n)
set.seed(11)
for(i in 1:n){
  y=sum((normal(d))^2)
   t[i]=normal(1)/(y/d)^(1/2)
}
return(t)
}
x=t_distribution(1000,10)
head(x)
```

```
[1] -1.9255752 -0.9876022  0.6298407 -1.5745710  0.8701880 -0.2545109
```

```
hist(x,col=10,ylim=c(0,0.4), probability=TRUE)
curve(dt(x,20),  lty = 2, lwd = 2, add = TRUE)
```



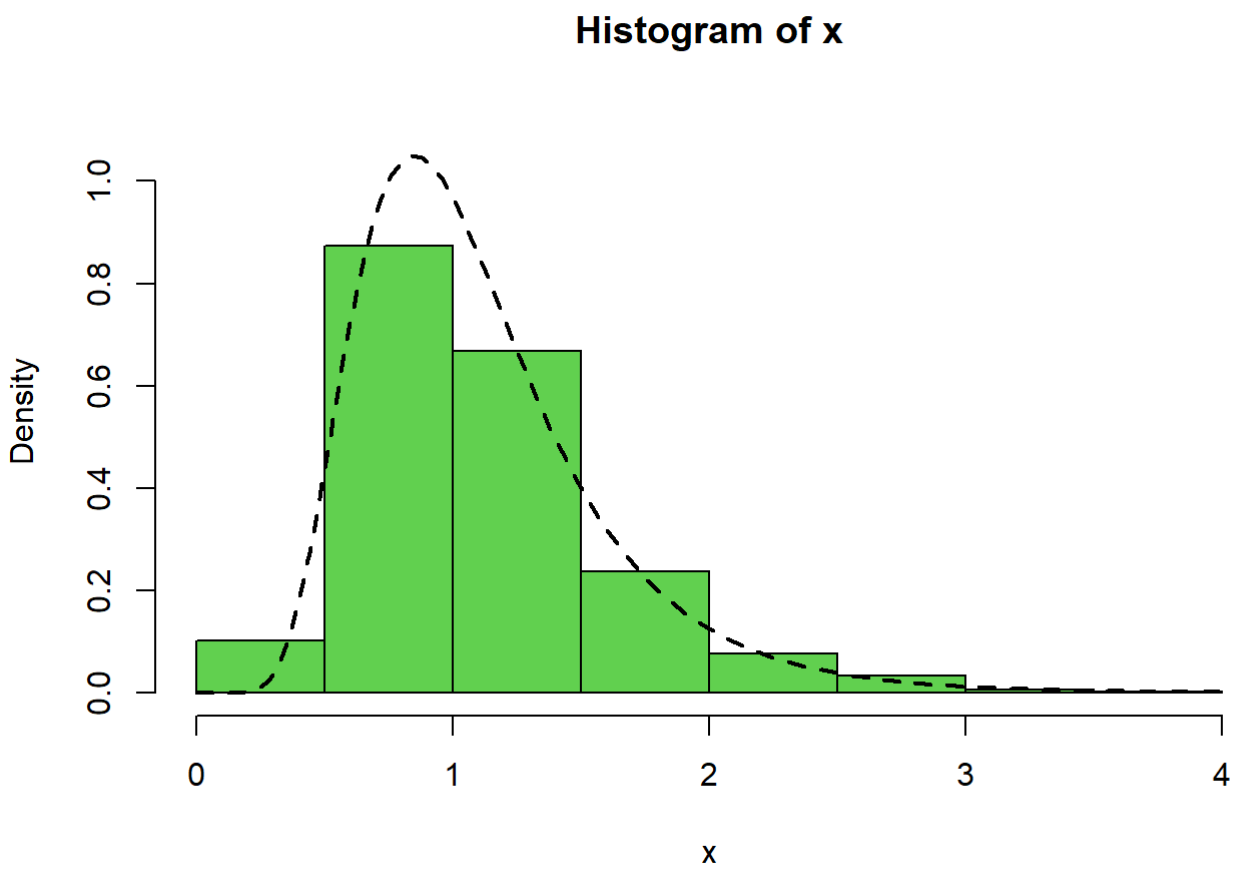
F-Distribution

Here we have used the relation of Standard normal and Chi square for F distribution. We know that when $X_i(\forall i = 1, 2....n)$, $Y_j(\forall j = 1, 2....m)$ are independent $N(0,1)$ & $\sum_{i=1}^{n} X_i^2 \sim {\chi_n}^2$, $\sum_{j=1}^{m} Y_j^2 \sim {\chi_m}^2$ where n & m are degrees of freedom then $F = \frac{\sum_{i=1}^{n} X_i^2/n}{\sum j=1^m Y_j^2/m} \sim F_{n,m}$

```
F_distribution=function(n,p,q){
  f=rep(0,n)
  set.seed(19)
  for(i in 1:n){
    v1=sum((normal(p))^2)
    v2=sum((normal(q))^2)
    f[i]=(v1/p)/(v2/q)
  }
  return(f)
}
x=F_distribution(1000,30,20)
head(x)
```

```
[1] 0.6532710 1.7721703 0.8833027 1.4150308 0.9960572 1.5874290
```

```
hist(x,col=3,ylim=c(0,1.1), probability=TRUE)
curve(df(x,30,20), lty = 2, lwd = 2, add = TRUE)
```



Simulating data from Bernoulli(p) Distribution

Simulate tossing a coin with probability of heads p.
Let X be a $Uniform(0,1)$ random variable. We can write Bernoulli random variable Y as $Y = \{1$ if $X<p$ or 0 if $X \geqslant p\}$
Thus, $PH = P(Y = 1) = P(X < p) = p$
Therefore, X has $Bernoulli(p)$ distribution. The R code for $Bernoulli(0.4)$ is:

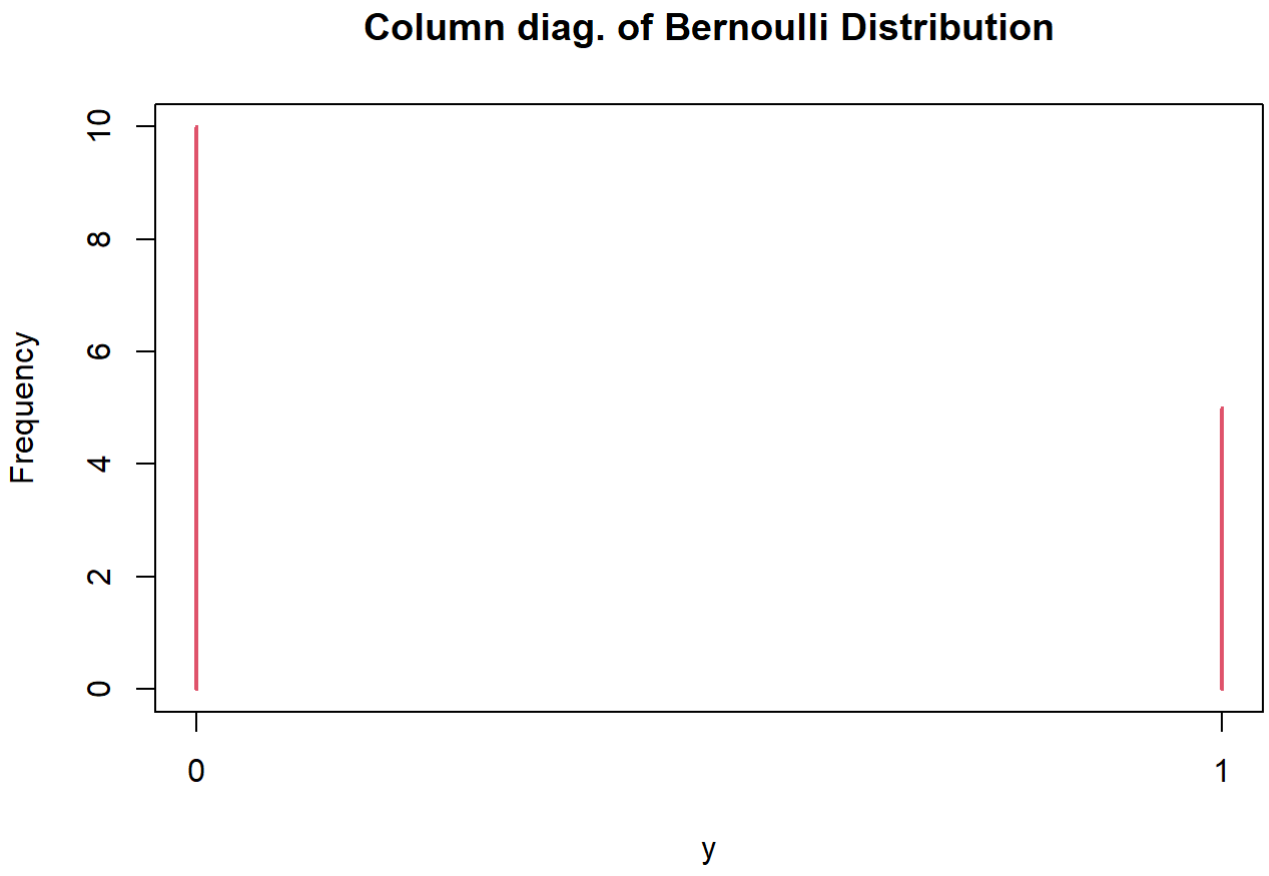
```
set.seed(2)
p=0.4
x=runif(15,0,1) # 15 sample from U(0,1) distribution
round(x,4)
```

```
 [1] 0.1849 0.7024 0.5733 0.1681 0.9438 0.9435 0.1292 0.8334 0.4680 0.5500
[11] 0.5527 0.2389 0.7605 0.1808 0.4053
```

```
y=(x<p)*1  #15 sample from Bernoulli Distribution
table(y )
```

```
y
 0  1
10  5
```

```
plot(table(y),type="h",col=2,ylab="Frequency",main="Column diag. of Bernoulli Distribution")
```



Simulating samples from Binomial(n,p) Distribution

If $X1$, $X2$, ..., Xn are independent $Bernoulli(p)$ random variables, then the random variable X defined by $X = X1 + X2 + \ldots + Xn$ has a $Binomial(n, p)$ distribution.
We have generated a random variable $X \sim Binomial(30, 0.7)$, we can toss a coin 30 times and count the number of heads.

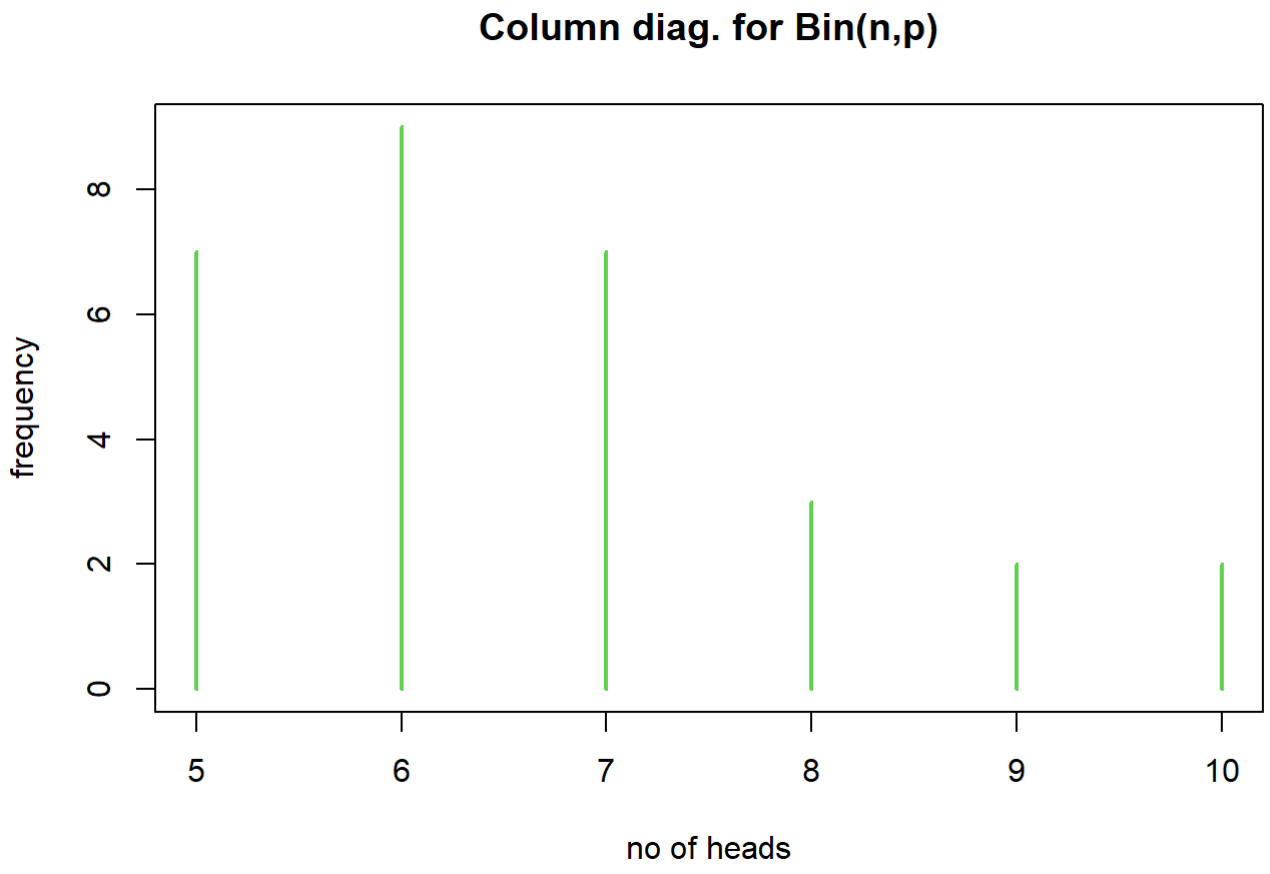
```
set.seed(21)
b=rep(0,30)
for(i in 1:30){
  u=runif(10)
  b[i]=sum(u<=0.7)     #sum of 10 Bernoulli(0.7) random sample
}
b
```

```
 [1]  5  9  5  6  7  7  5  6  7  7  5  5  6  7  7  8  8  6  6  8 10  6  6 10  9
[26]  7  5  6  5  6
```

```
table(b)
```

```
b
 5  6  7  8  9 10
 7  9  7  3  2  2
```

```
plot(table(b),type="h",col=3,main="Column diag. for Bin(n,p)",xlab="no of heads",ylab="frequency")
```



Conclusion

- Counting the number of heads is exactly the same as finding $X1 + X2 + \ldots + Xn$, where each Xi is equal to one if the corresponding coin toss results in heads and zero otherwise.
- Since we know how to generate Bernoulli random variables, we can generate a $Binomial(n, p)$ by adding n independent $Bernoulli(p)$ random variables.

Demonstration of the Frequency Definition of Probability by simulating coin toss experiment

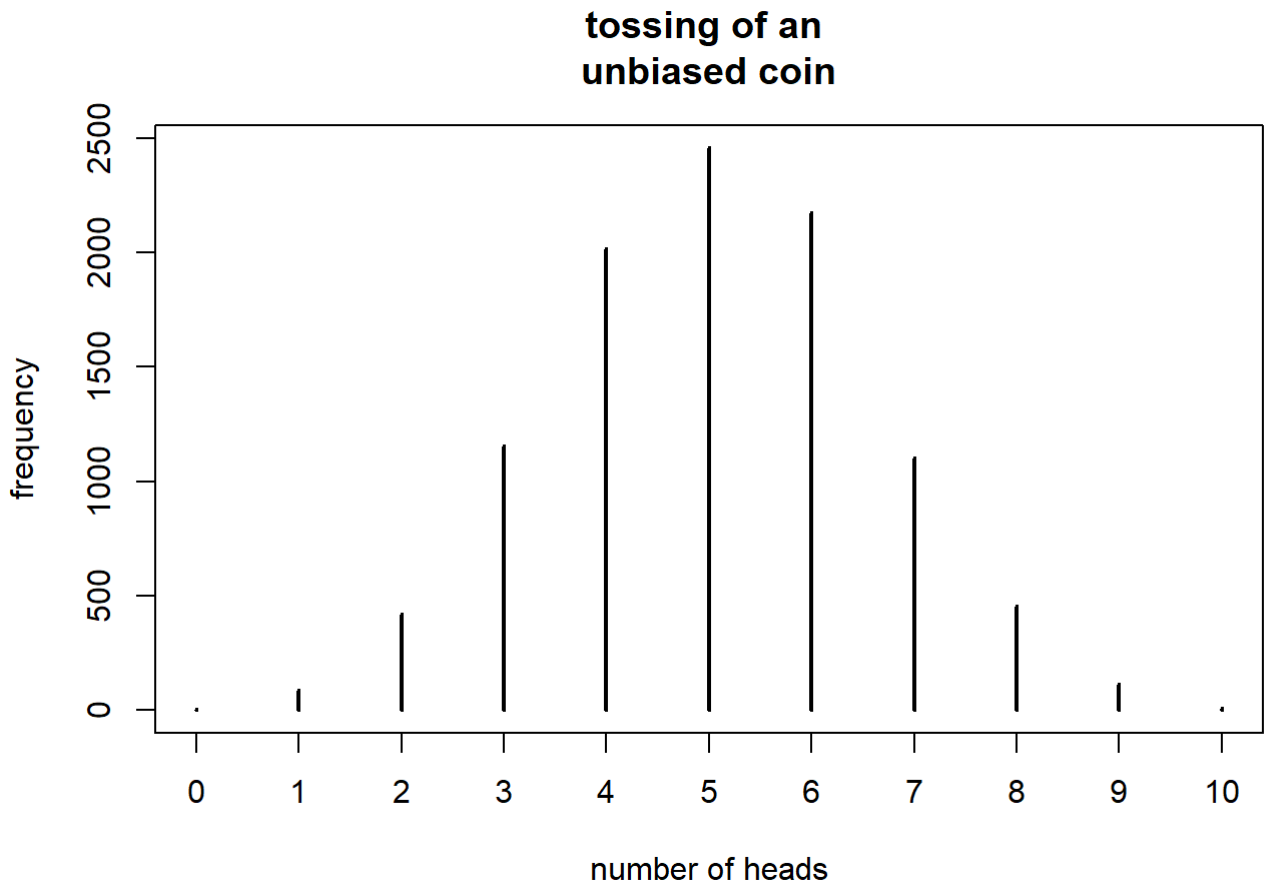
Frequency definition of Probability:
The finite frequency theory of probability defines the probability of an outcome as the frequency of the number of times the outcome occurs relative to the number of times that it could have occurred.

Let take a look for an *unbiased coin* We may toss an unbiased coin with success probability 0.5 for 10 times and then simulate the process for 10000 times and we are interested to observe the number of heads.
If X be the random variable denoting the number of heads, them X ~ Bin(10,0.5).

```
set.seed(11)
x=rbinom(10000,10,0.5) #simulating data from binomial distribution
table(x) #the frequency table of number of heads
```

```
x
   0    1    2    3    4    5    6    7    8    9   10
   6   87  421 1156 2017 2457 2173 1104  457  114    8
```

```
plot(table(x),ylab="frequency",xlab="number of heads",main="tossing of an
unbiased coin")
```



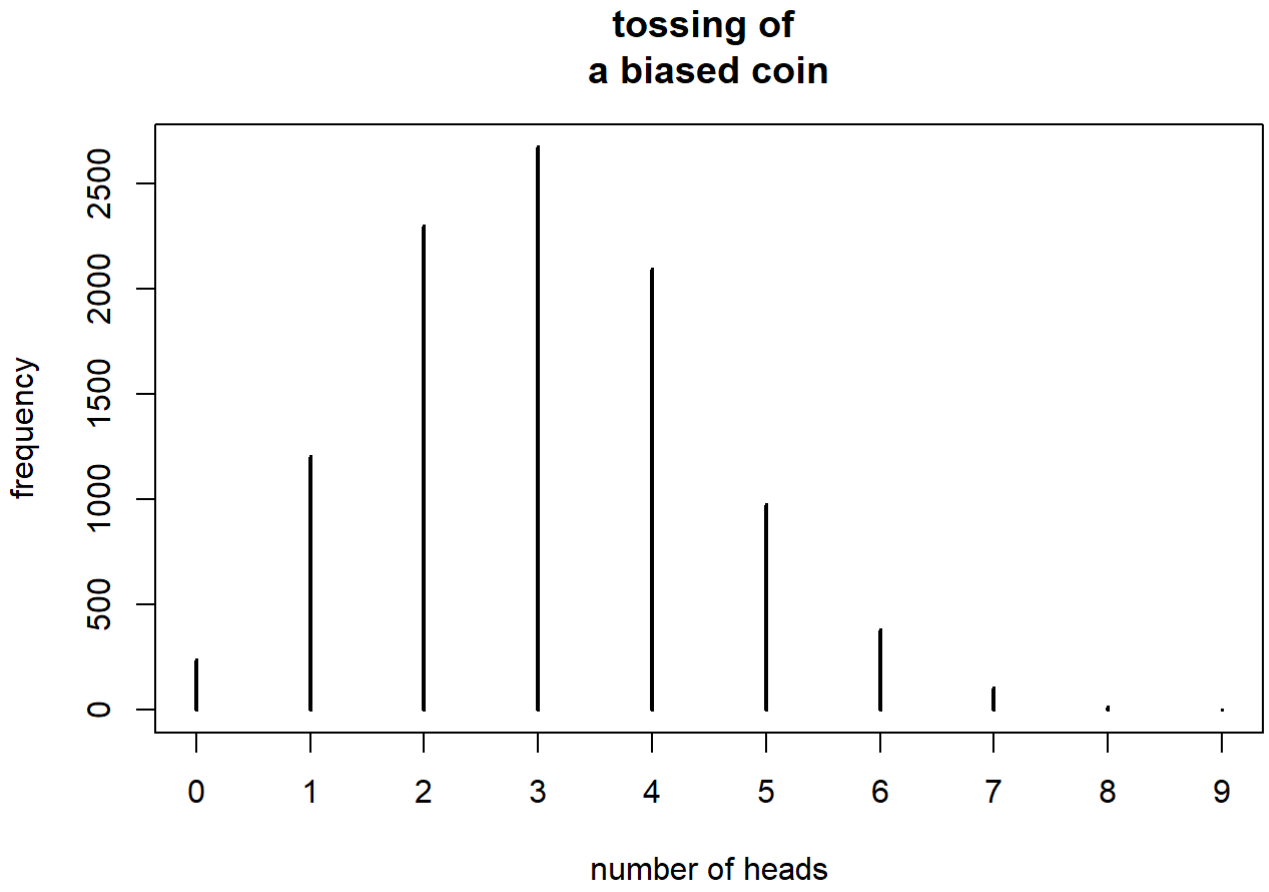
Now, theoretically P(X=2) which is approximately equal to 0.0439 And from the table,for the frequency probability, we can calculate the relative frequency for X=2, which is 0.0421. Thus we can demonstrate Frequency definition of Probability by comparing the theoritical probability and the relative frequency.

Now let us take a look for a *Biased coin*. We shall repeat the same experiment and simulate the data from a binomial distribution but this time with different success probability, say, 0.3.

```
set.seed(11)
y=rbinom(10000,10,0.3) #simulating data from binomial distribution
table(y) #the frequency table of number of heads
```

```
y
   0    1    2    3    4    5    6    7    8    9
 240 1205 2301 2676 2098  976  383  105   15    1
```

```
plot(table(y),lwd=2,ylab="frequency",xlab="number of heads",main="tossing of
a biased coin")
```



Here we shall do the same comparison for X=7 which is approximately equal to 0.009 and the relative frequency for X=7 is 0.010. For the diagram the peak has been attained for X=3 and we have a right tailed which positively skewed distribution of number of heads. Hence our demonstration for biased coin.

Illustration Of CLT using Simulated Data

Under certain conditions, in large samples, the sampling distribution of the sample mean can be approximated by a normal distribution. The Central Limit Theorem (CLT) states that the distribution of sample means approximates a normal distribution as the sample size gets larger, regardless of the population's distribution.

Outline

- We have simulated data from certain continuous distributions.
- We have obtained the distribution of sample means for each.

- Illustrated the distribution of sample means with histogram and the normal distribution it approximates to.

Demonstrating CLT on Chi-Square Distribution χ^2

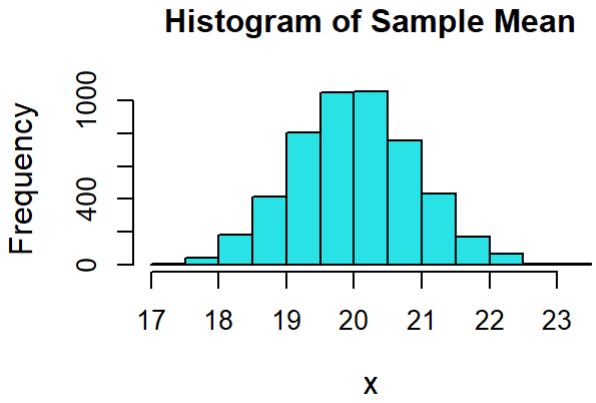
```
#n=20 #df of cho square (say)
#mean= 20 and variance =40
set.seed(19)

x=rep(0,5000)
for(i in 1:5000){
  x[i]=mean(rchisq(50,20))
}
head(x,10)
```

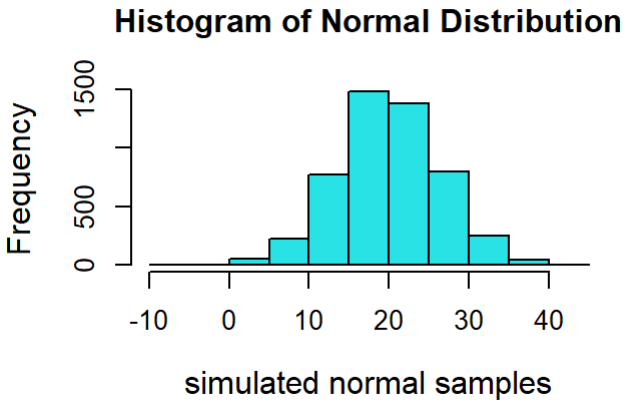
```
 [1] 20.32117 20.69464 18.87245 19.47377 21.48812 20.66191 21.07103 19.57742
 [9] 19.68984 20.62013
```

```
mean(x)  #Mean of the sample means
```

```
[1] 20.00638
```

```
par(mfrow=c(2,2))
#Illustrating Histogram of Sample Means and the Normal Distribution it approximates to
hist(x,col=5,main="Histogram of Sample Mean",cex.lab=1.2)
hist(rnorm(5000,20,sqrt(40)),col=5,xlab="simulated normal samples",main="Histogram of Normal
 Distribution",cex.lab=1.2)
```





Demonstrating CLT on t- Distribution(Students's t)

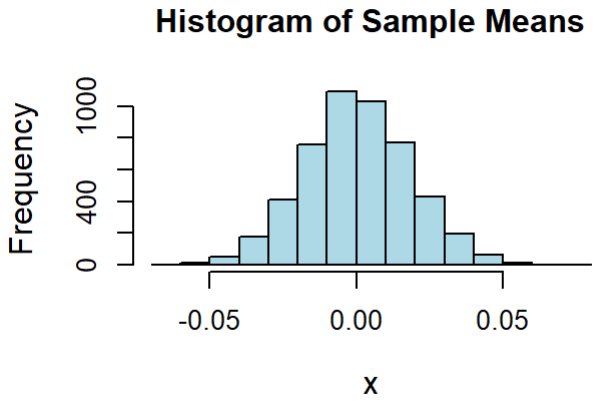
```
#n=5 df of t and variance is 5/3
x=rep(0,5000)
set.seed(11)
for(i in 1:5000){
  x[i]=mean(rt(5000,5))
}
head(x)
```

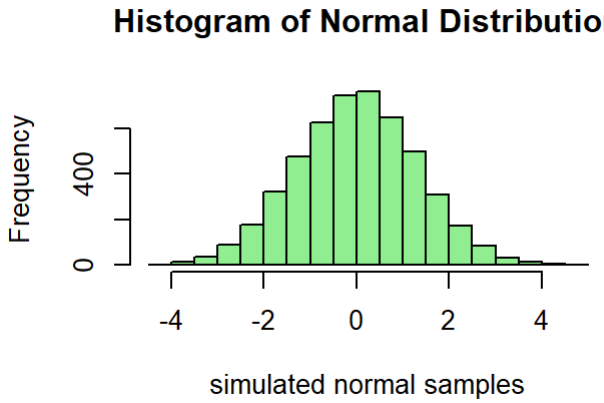
```
[1]  0.030513039 -0.008547241  0.011978132  0.012460881 -0.008820000
[6] -0.005119580
```

```
mean(x)  #Mean of the samples evaluated from the loop
```

```
[1] 0.0002325168
```

```
par(mfrow=c(2,2))
#Illustrating Histogram of Sample Means and the Normal Distribution it approximates to
hist(x,col="lightblue",main="Histogram of Sample Means",cex.lab=1.2)
hist((rnorm(5000,0,sqrt(5/3))),xlab="simulated normal samples",col="lightgreen",main="Histogr
am of Normal Distribution")
```





Demonstrating CLT on Exponential Distribution

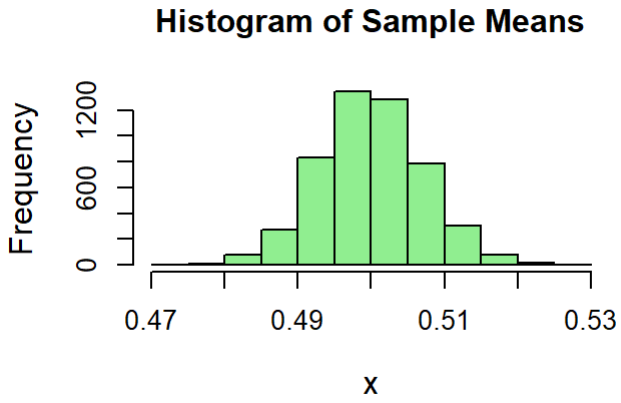
```
x=rep(0,5000)
set.seed(11)
for(i in 1:5000){
  x[i]=mean(rexp(5000,2))
}
head(x)
```

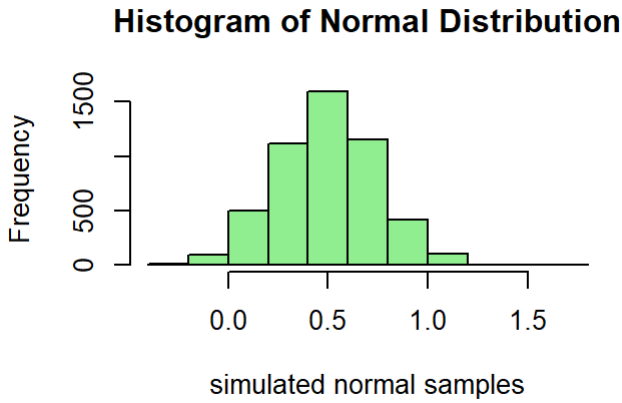
```
[1] 0.4967987 0.4987808 0.5000982 0.4954560 0.5018513 0.5028053
```

```
mean(x)  #Mean of the samples evaluated from the loop
```

```
[1] 0.4999645
```

```
par(mfrow=c(2,2))
#Illustrating Histogram of Sample Means and the Normal Distribution it approximates to
hist(x,col="lightgreen",main="Histogram of Sample Means",cex.lab=1.2)
hist(rnorm(5000,0.5,1/(2^2)),xlab="simulated normal samples",col="lightgreen",main="Histogram
of Normal Distribution")
```





Demonstrating CLT on Poission Distribution

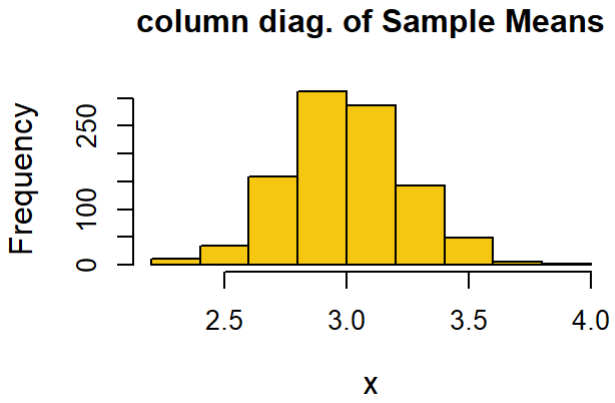
```
#taking the parameter lam=3
x=rep(0,1000)
set.seed(11)
for(i in 1:1000){
  x[i]=mean(rpois(50,3))
}
head(x)
```

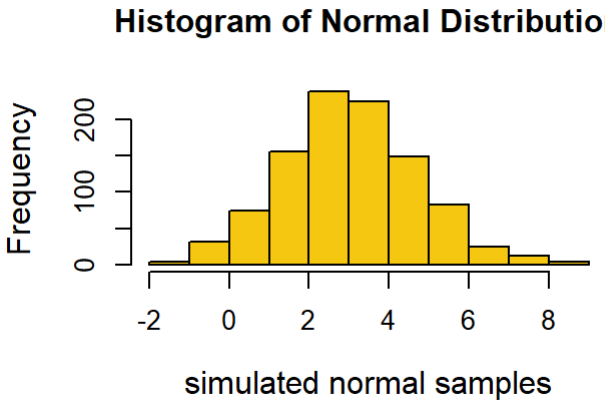
```
[1] 2.42 2.58 3.14 2.96 3.42 2.76
```

```
mean(x)  #Mean of the samples evaluated from the loop
```

```
[1] 3.00698
```

```
#Illustrating Histogram of Sample Means and the Normal Distribution it approximates to

par(mfrow=c(2,2))
hist(x,col=7,main="column diag. of Sample Means",cex.lab=1.2)
hist(rnorm(1000,3,sqrt(3)),col=7,xlab="simulated normal samples",main="Histogram of Normal Di
stribution",cex.lab=1.2)
```





The Resampling Method: Jackknife

What is a Jackknife method?

The jackknife method is a popular **resampling method** that provides estimates of the bias and a standard error of an estimate by recomputing the estimate from subsamples of the available sample. It involves a leave-one-out strategy of the estimation of a parameter (e.g., the mean,median.etc) in a data set of N observations. Ideally, N − 1 models are built on the data set with different factors left out of each new dataset.

Pros and Cons of Jackknife:

- Computationally simpler than bootstrapping, more orderly as it is computationally simpler than bootstrapping, more orderly as it is iterative.
- Still fairly computationally intensive.
- Does not perform well for nonlinear statistics.
- Requires observations to be independent of each other hence,it is not suitable for time series analysis.

Procedure

- The jackknife deletes each observation and calculates an estimate based on the remaining n − 1 of them.
- This collection of estimates to do things like estimate the bias and the standard error of *Median* of the data.
- We've considered the univariate data of height of 30 people.

```
x=c(83.9,83.6,81.3,85.4,83.9,81.1,84.9,81.1,81.9,79.6,77.7,76.4,77.2,80.1,76.9,81.5,80.9,83.1
,84.2,80.3,83.4,79.1,86.1,78.6,80.1,81.6,78.6,82.5,84.1,87.1)

sam.median=median(x) #median of the data set
sam.median
```

```
[1] 81.4
```

```
jk.estimator=rep(0,30)
for(i in 2:31){
  jk.estimator[i-1]=median(x[-i+1]) #jk estimator for ith sample
}
jk.estimator
```

```
 [1] 81.3 81.3 81.5 81.3 81.3 81.5 81.3 81.5 81.3 81.5 81.5 81.5 81.5 81.5 81.5
[16] 81.3 81.5 81.3 81.3 81.5 81.3 81.5 81.3 81.5 81.5 81.3 81.5 81.3 81.3 81.3
```

```
JK=mean(jk.estimator) #the jackknife estimate
JK
```

```
[1] 81.4
```

```
bias=29*(JK-sam.median) #the bias of estimator, median
bias
```

```
[1] 0
```

```
s=sum((jk.estimator-JK)^2) #standard error of the estimator
se=sqrt((29/30)*s)
se
```

```
[1] 0.5385165
```

The Resampling Method : BOOTSTRAP

What is a Bootstrap?

A Bootstrap is a very useful **resampling technique** when we have too small sample size to draw inference. Also it is surprisingly helpful for constructing confidence intervals and calculating standard errors for difficult statistics.
In Bootstrapping, samples are drawn with replacement. It consists in taking multiples samples out of our original sample and study the resulting distribution. It allows us to see how much variation there would have been and gives further understanding of the sample that we took in the first place.

Why should we use Bootstrap?

- It is easy to implement.

- It doesn't rely in any kind of assumption of population distribution.
- Enables us to draw inference even when we have sample size like 10.

Our Illustration:

Here we have considered two data sets, consisting the test scores of 10 students on a certain subject,before and after a teaching method was introduced.
Object: We want to see if the teaching method was useful at all.
Procedure:
* We consider Bootstrapping from the sets of data.
* Defining the absolute difference of means of two groups, our *test statistics*.
* Setting our null and alternative hypothesis, H_0: The method is effective and H_1:The method is not effective.

```
library(readxl)
dt1=read_excel("C:/Users/Hp/Desktop/scores.xlsx",sheet="scores")
View(dt1)

obs.diff=abs(mean(dt1$method1)-mean(dt1$method2)) #the crical value
obs.diff
```

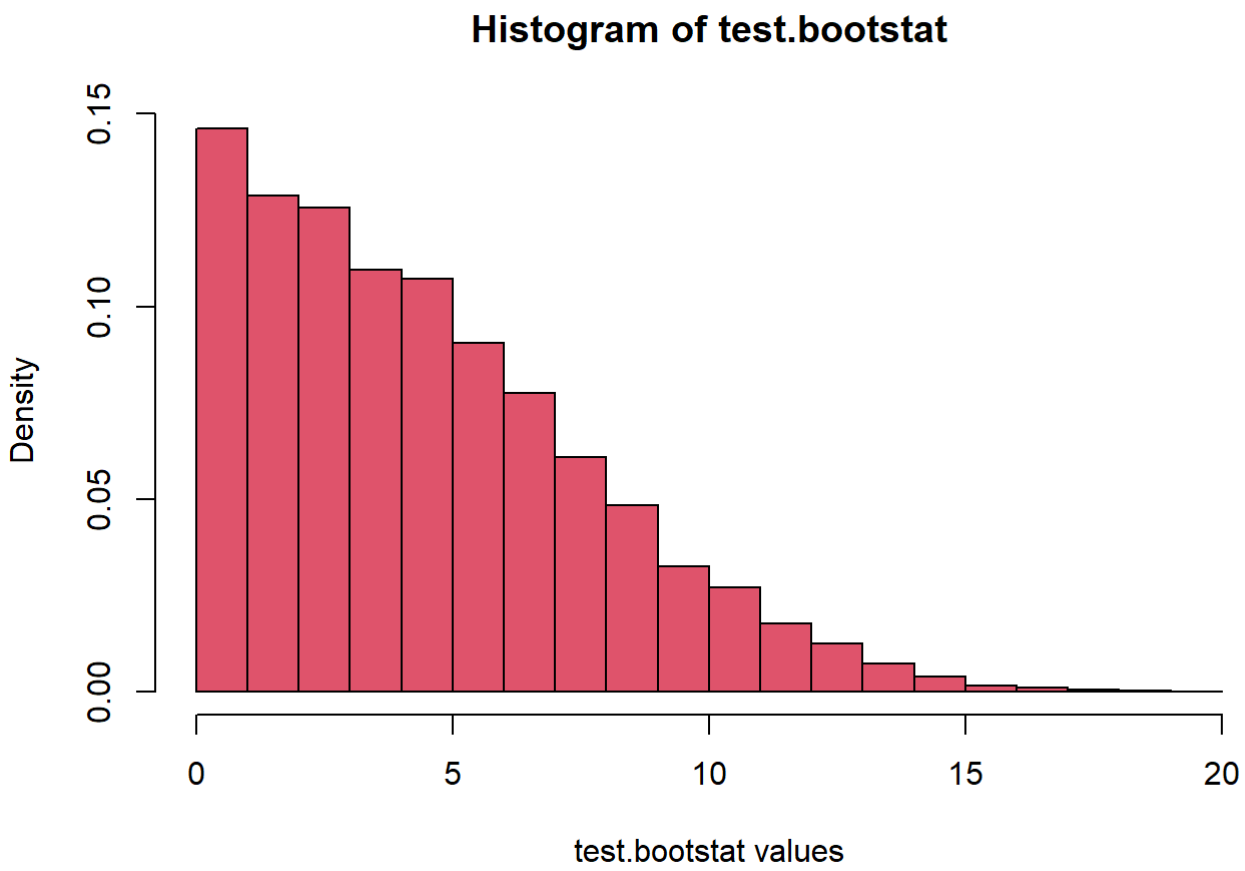
```
[1] 2.5
```

R code and Output of Bootstrapping:

```
boot.num=10000 #number of times we are generating bootstrap samples
test.bootstat=rep(0,boot.num) #observed values of test statistic
set.seed(11)
for(i in 1:boot.num){
  x=sample(dt1$method1,10,replace=TRUE) #sampling from group 1
  y=sample(dt1$method2,10,replace=TRUE) #sampling from group 2
  boot.sam=c(x,y)                       #bootstrap samples
  test.bootstat[i]=abs((mean(x)-mean(y)))
}

head(test.bootstat,100)
```

```
  [1]  0.5  0.5 12.3  1.9  0.5  2.4  4.2  4.5  5.8  2.1  0.7  3.7  9.6  6.1  6.1
 [16]  2.6  5.7  5.0  6.5  0.2  4.3  8.1  0.5  2.0  9.0  2.0  5.9  2.8  4.3  0.3
 [31]  1.9  6.3  1.1  0.5  0.2  4.3  1.4  4.0  0.4  7.5  5.1  3.3  6.9  5.4  1.7
 [46]  2.6  1.3  4.3  1.3  0.9  8.9  7.3  7.5  2.2  2.0  2.5  2.5  0.5  3.6  3.7
 [61]  3.9  4.5  5.0  6.4  3.6  3.3 10.9  2.8  0.5  3.6  1.1  8.4  0.5  5.6  6.3
 [76]  5.6  2.0  7.9 11.3  2.9  3.1  0.4  1.8  2.1  0.8  7.9  3.2  2.5 10.7  4.3
 [91]  2.5  5.5  8.9  3.9  4.4  9.9  4.9  4.8  1.1  2.1
```

```
hist(test.bootstat,xlab="test.bootstat values",probability=T,col=2) #histogram of the test st
atistic
```



Observing the histogram, we assume that the distribution of the test statistic is a **Folded Normal.**

Obtaing the P-value:

```
n=sum(test.bootstat>obs.diff)
p.value=(n/boot.num)
p.value
```

```
[1] 0.664
```

```
if(p.value>0.05){
  print("We accept H0")   #our conclusion
} else {
  print("We reject H0")
}
```

```
[1] "We accept H0"
```

Our Conclusion:

We accept H_0, i.e. the new teaching method is effective.

Monte Carlo Method

What is Monte Carlo?

Monte Carlo simulation is named after the popular gambling destination of Monaco. This technique was developed by Stanislaw Ulan who worked on Manhattan Project.This method is a burning example of how we can actually bypass the complications in Mathematical calculations and step forward. Monte Carlo simulation is used for modelling the probability of different outcomes in a process that can not be predicted due to the presence of random variables which are not easy to deal with.

Monte Carlo method is very flexible in handling complicated integrals which benefits us. We can still calculate expectation and variance (which are basically integrals) even when the tractable form of the probability function is unavailable.

So why Monte Carlo when we have Numerical Methods?

Indeed we have numerical methods to solve complex integrals. But those methods generally cannot be extended to higher dimension. Even if we could extend them, they need very intense algebraic calculations. Here is where Monte Carlo surpasses them.

If it is not possible to obtain the exact analytic solution often Monte Carlo method can be used to provide a very good approximate solution to,

- Calculation the area below a curve.
- Calculation multidimensional integration.
- Optimization.
- Analyzing any complicated stochastic system.

Monte Carlo Integration

Methodology

Let $X \sim f$ be a random variable (could be multidimensional!) and consider a real valued function g and the corresponding random variable $g(X)$. The expectation of $g(X)$ is given by, $E[g(x)] = \int_{\chi} g(x)f(x)dx$

where χ is the support of X.Two common examples of g are $g(x) = x$ and $g(z) = (z - E[X])$. In these cases $E[g(X)$ is nothing but mean and variance of X.

Let $\theta = E[g(X)]$,the Monte Carlo algorithm to compute an approximation to θ is denoted by $\hat{\theta_{MC}}$ is as follows:

* Generate n samples from $f(x) : X_1, X_2, \ldots X_n$, then the Monte Carlo approximation is given by, $\hat{\theta_{MC}} = \frac{1}{n}\sum_{i=1}^{n} g(X_i)$

An Illustration:

Suppose we have to compute $I = \int_{2}^{10} log(x)^{log(x)} dx$

We could write, $I = \int_{2}^{10} log(x)^{log(x)} dx$

or, $I = 8 \int_{2}^{10} \frac{1}{10-2} log(x)^{log(x)} dx$

so we have $g(x) = log(x)^{log(x)}$ and $f(x) = \frac{1}{10-2}$

So we generate n samples from $U(2, 10)$ and use the Monte Carlo approximation.

```
# the evaluated value of the integral
f=function(x){
  (log(x)^log(x))
}
integrate(f,2,10)
```

```
25.26641 with absolute error < 1.7e-08
```

```
#The Monte Carlo approximation
set.seed(11)
u=runif(1000,2,10)
x=(log(u)^log(u))
8*mean(x) #pretty close!
```

```
[1] 24.96239
```

```
#lets increase the sample size,
set.seed(11)
u=runif(5000,2,10)
x=(log(u)^log(u))
8*mean(x) #Hallelujah! 25.261!
```

```
[1] 25.26181
```

Hence we have successfully obtained our value.

But then every coin has two sides.
If we come across a rare event and are to sample that then we have to hit a very large size of samples. Which could be time consuming or might not serve our purpose. Then we opt for Importance Sampling.