Statistical Methods for Data Science

Mini Project 1

Members:

SHIVA RANGA CHAWALA	sxc167630
KRISHNA SINDHU KOTA	kxk171030

Contribution:

Shiva Ranga Chawala – Q1

Krishna Sindhu Kota – Q2

1. **Consider a discrete random variable X that takes 4 values -- 1, 2, 3 and 4 with respective probabilities 1/2, 1/8, 1/8, and 1/4.**

a. **Compute E(X), Var(X) and P(X <=2) analytically, i.e., using their formulas.**

$E(X) = \sum_x x.f(x)$

$= (1)(1/2) + (2)(1/8) + (3)(1/8) + (4)(1/4)$
$= (1/2) + (1/4) + (3/8) + (1)$
$= 0.5 + 0.25 + 0.375 + 1$
$= 2.125$

$Var(X) = \sum_x (x-\mu)^2.f(x),\ \mu = E(X)$

$= (1-2.125)^2 (1/2) + (2-2.125)^2(1/8) + (3-2.125)^2(1/8) + (4-2.125)^2(1/4)$
$= (1.265) (0.5) + (0.015) (0.125) + (0.765) (0.125) + (3.515) (0.25)$
$= 0.6325 + 0.0018 + 0.0956 + 0.8787$
$= 1.6086$

$P(X<=2) = P(X=1) + P(X=2)$

$= ½ + 1/8$
$= 0.5 + 0.125$
$= 0.625$

b. **Explain how you would simulate a draw from the distribution of X.**

We will be using inbuild R function to simulate draws. i.e sample(x, size, replace = FALSE/TRUE, prob)

Where x → vector that contains discrete random variables
size →number of numbers to choose
replace → used to specify if we want draws with/without replacement
prob → vector that contains probability values for given variables

In our case this will be sample(x,1000,replace=TRUE,prob)
Where 'x' will contain c(1,2,3,4) and 'prob' will contain c(0.5,0.125,0.125,0.25), so this will make draws of 1 with probability 0.5, 2 with probability 0.125, 3 with probability 0.125 and 4 with probability 0.25.

We can do the same process without using inbuilt function. i.e runif() → will simulate a draw with value between 1 and 0.
runif(1000) will simulate 1000 draws with values between 1 and 0.

Since the probabilities of 1,2,3,4 are ½,1/8,1/8,1/4, all the values that are in interval 0 and 0.5 will be mapped to 1, values in interval 0.5 and 0.625 will be mapped to 2, values in interval 0.625 and 0.75 will be mapped to 3 and values in interval 0.75 to 1 will be mapped to 4.

c. **Approximate E(X), var(X) and P(X _ 2) using Monte Carlo simulation with 1,000 draws 5 times. Summarize the results in a table.**

R code for making 1000 draws 5 times:

```
#defining vectors x, prob to store values of x and their probabilities
x = c(1,2,3,4) #Storing variables in x
prob = c(0.5,0.125,0.125,0.25) #Storing their probabilities in prob
times=5
a=b=c=integer(times)
#for loop to simulate 5 times
for(i in 1:5)
{
  #simulating 1000 draws and storing them in set1k
  set1k = sample(x, size=1000, replace = TRUE, prob)
  #Dividing size of draws whose value is less than or equal to 2 by total size of
#draws we get P(X<=2)
  prob1k = length(which(set1k<=2))/length(set1k)
  #Printing E(X), Var(X) and P(X<=2)
  print(paste("Values for draw i:",i))
  print(paste("E(X) for size 1000 =",mean(set1k)))
  a[i] = mean(set1k) #Storing the mean everytime in vector "a"
  print(paste("Var(X) for size 1000 =",var(set1k)))
  b[i] = var(set1k) #Storing the variance everytime in vector "b"
  print(paste("P(X<=2) for size 1000 =",prob1k))
  c[i] = prob1k #Storing the P(X<=2) everytime in vector "c"
}
E = mean(a) #Calculating mean of all 5 values of mean
print(paste("Mean of values of E(X) after 5 times:",E))
V = mean(b) #Calculating mean of all 5 values of variance
print(paste("Mean of values of Var(X) after 5 times:",V))
P = mean(c) #Calculating mean of all 5 values of P(X<=2)
print(paste("Mean of values of P(X<=) after 5 times:",P))
```

Below are the results of the code:

	1	2	3	4	5	Mean
E(X)	2.078	2.126	2.154	2.167	2.143	2.1336
Var(X)	1.537	1.625	1.661	1.612	1.620	1.611
P(X<=2)	0.646	0.627	0.615	0.611	0.611	0.622

d. Repeat (c) with 5,000 and 10,000 draws.

i. R code for making 5000 draws 5 times:

```
#defining vectors x, prob to store values of x and their probabilities
x = c(1,2,3,4) #Storing variables in x
prob = c(0.5,0.125,0.125,0.25) #Storing their probabilities in prob
times=5
a=b=c=integer(times)
#for loop to simulate 5 times
for(i in 1:5)
{
  #simulating 5000 draws and storing them in set1k
  set5k = sample(x, size=5000, replace = TRUE, prob)
  #Dividing size of draws whose value is less than or equal to 2 by total size of
#draws we get P(X<=2)
  prob5k = length(which(set5k<=2))/length(set5k)
  #Printing E(X), Var(X) and P(X<=2)
  print(paste("Values for draw i:",i))
  print(paste("E(X) for size 5000 =",mean(set5k)))
  a[i] = mean(set5k)
  print(paste("Var(X) for size 5000 =",var(set5k)))
  b[i] = var(set5k)
  print(paste("P(X<=2) for size 5000 =",prob5k))
  c[i] = prob5k
}
E = mean(a) #Calculating mean of all 5 values of mean
print(paste("Mean of values of E(X) after 5 times:",E))
V = mean(b) #Calculating mean of all 5 values of Variance
print(paste("Mean of values of Var(X) after 5 times:",V))
P = mean(c) #Calculating mean of all 5 values of P(X<=2)
print(paste("Mean of values of P(X<=) after 5 times:",P))
```

Below are the results of the code:

	1	2	3	4	5	Mean
E(X)	2.113	2.101	2.103	2.129	2.12	2.1132
Var(X)	1.617	1.598	1.581	1.611	1.603	1.602
P(X<=2)	0.632	0.634	0.631	0.626	0.625	0.6296

ii. R code for making 10000 draws 5 times:

```
#defining vectors x, prob to store values of x and their probabilities
x = c(1,2,3,4) #Storing variables in x
prob = c(0.5,0.125,0.125,0.25) #Storing their probabilities in prob
#for loop to simulate 5 times
for(i in 1:5)
{
  #simulating 10000 draws and storing them in set1k
  set10k = sample(x, size=10000, replace = TRUE, prob)
  #Dividing size of draws whose value is less than or equal to 2 by total size of
#draws we get P(X<=2)
  prob10k = length(which(set10k<=2))/length(set10k)
  #Printing E(X), Var(X) and P(X<=2)
  print(paste("Values for draw i:",i))
  print(paste("E(X) for size 10000 =",mean(set10k)))
  a[i] = mean(set10k)
  print(paste("Var(X) for size 10000 =",var(set10k)))
  b[i] = var(set10k)
  print(paste("P(X<=2) for size 10000 =",prob10k))
  c[i] = prob10k
}
E = mean(a) #Calculating mean of all 5 values of mean
print(paste("Mean of values of E(X) after 5 times:",E))
V = mean(b) #Calculating mean of all 5 values of Variance
print(paste("Mean of values of Var(X) after 5 times:",V))
P = mean(c) #Calculating mean of all 5 values of P(X<=2)
print(paste("Mean of values of P(X<=) after 5 times:",P))
```

Below are the results of the code:

	1	2	3	4	5	Mean
E(X)	2.113	2.128	2.118	2.126	2.124	2.1218
Var(X)	1.611	1.605	1.598	1.621	1.619	1.6108
P(X<=2)	0.627	0.624	0.627	0.622	0.627	0.6254

e. **Compare you results in (a), (c) and (d). Explain, with justi_cation, what you observe.**

Calculated values of E(X), Var(X) and P(X<=2):

E(X) = 2.125
Var(X) = 1.6086
P(X<=2) = 0.625

Values of means of E(X), Var(X) and P(X<=2) for 1000 draws replicated 5 times:

	Mean	Calculated values
E(X)	2.1336	2.125
Var(X)	1.611	1.6086
P(X<=2)	0.622	0.625

Values of E(X), Var(X) and P(X<=2) for 5000 draws replicated 5 times:

	Mean	Calculated values
E(X)	2.1132	2.125
Var(X)	1.602	1.6086
P(X<=2)	0.6296	0.625

Values of E(X), Var(X) and P(X<=2) for 10000 draws replicated 5 times:

	Mean	Calculated values
E(X)	2.1218	2.125
Var(X)	1.6108	1.6086
P(X<=2)	0.6254	0.625

If we observe the values that are generated for 1000 draws, 5000 draws and 10000 draws the values and coming closer to the values that we calculated in (a). This proves that as we perform large no. of trials, the mean, variance and the probability values will come close to the actual values. This is what Law of Large Numbers tells us.

2. **Suppose X1,X2, ….., Xn denotes a random sample from a Bernoulli (p) population, represented by the random variable X, and let X denote the sample mean. This sample mean also represents the proportion of 1s in the sample, say, ^p. We know from Central Limit Theorem that ^p approximately follows a normal distribution when n is large. The goal of this exercise is investigate how large n should be for the**

approximation to be good. For this investigation, we will focus on p = 0.10,0.25, 0.50, 0.75, 0.90, and n = 10, 30, 50, 100.

(a) **What is the approximate distribution of ^p when n is large?**
When n is large, it has a normal distribution with mean as p and with variance as p*(1-p) which can be observed from the results of below code.

```
#2
#p = 0.10, 0.25, 0.50, 0.75, 0.90 and n = 10, 30, 50, 100.
p=c (0.10,0.25,0.50,0.75,0.90)
n=c (10,30,50,100)

#2a Distribution of sample mean of X
dis_mean = c (dis_mean, 1:(length(n)*length(p))) #vector to store the mean
dis_var = c (dis_var, 1:(length(n)*length(p))) #vector to store the variance
for (i in n) {
  for (j in p) {
    dis = rbinom (i, 1, j) #Bernoulli distribution
    dis_mean =mean(dis) #Mean of the distribution
    dis_var= var(dis) #Variance of the distribution
    print(paste("p",j))
    print (paste ("dis_mean ", dis_mean))
    print(paste("p*(1-p)" ,  j*(1-j)))
    print (paste ("dis_var ", dis_var))
  }
}
#  Q-Q plot for larger values of n
Repli = replicate (500 , mean(rbinom(10000,1,0.50))) #Simulating 500 values of sample
#mean
#Generating Q-Q plot of 500 values
qqnorm(repli)
qqline(repli)
```

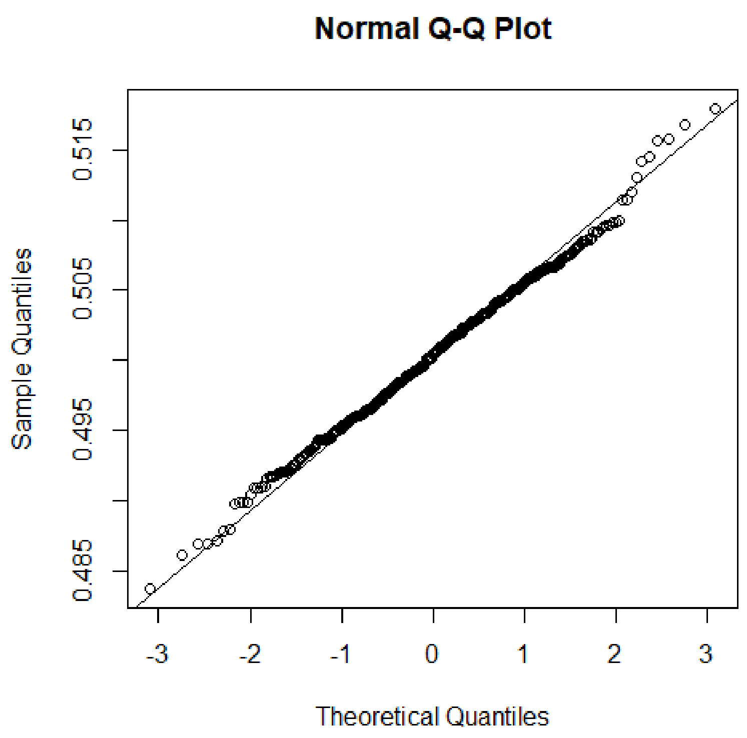
Result:

```
[1] "p 0.1"

[1] "dis_mean  0.1"

[1] "p*(1-p) 0.09"
```

```
[1] "dis_var  0.1"
[1] "p 0.25"
[1] "dis_mean  0.1"
[1] "p*(1-p) 0.1875"
[1] "dis_var  0.1"
[1] "p 0.5"
[1] "dis_mean  0.5"
[1] "p*(1-p) 0.25"
[1] "dis_var  0.277777777777778"
[1] "p 0.75"
[1] "dis_mean  0.8"
[1] "p*(1-p) 0.1875"
[1] "dis_var  0.177777777777778"
[1] "p 0.9"
[1] "dis_mean  1"
[1] "p*(1-p) 0.09"
[1] "dis_var  0"
[1] "p 0.1"
[1] "dis_mean  0.0333333333333333"
[1] "p*(1-p) 0.09"
[1] "dis_var  0.0333333333333333"
[1] "p 0.25"
[1] "dis_mean  0.266666666666667"
[1] "p*(1-p) 0.1875"
[1] "dis_var  0.202298850574713"
[1] "p 0.5"
[1] "dis_mean  0.433333333333333"
[1] "p*(1-p) 0.25"
[1] "dis_var  0.254022988505747"
[1] "p 0.75"
```

```
[1] "dis_mean  0.8"
[1] "p*(1-p) 0.1875"
[1] "dis_var  0.16551724137931"
[1] "p 0.9"
[1] "dis_mean  0.966666666666667"
[1] "p*(1-p) 0.09"
[1] "dis_var  0.0333333333333333"
[1] "p 0.1"
[1] "dis_mean  0.06"
[1] "p*(1-p) 0.09"
[1] "dis_var  0.0575510204081633"
[1] "p 0.25"
[1] "dis_mean  0.32"
[1] "p*(1-p) 0.1875"
[1] "dis_var  0.222040816326531"
[1] "p 0.5"
[1] "dis_mean  0.5"
[1] "p*(1-p) 0.25"
[1] "dis_var  0.255102040816327"
[1] "p 0.75"
[1] "dis_mean  0.78"
[1] "p*(1-p) 0.1875"
[1] "dis_var  0.175102040816327"
[1] "p 0.9"
[1] "dis_mean  0.96"
[1] "p*(1-p) 0.09"
[1] "dis_var  0.0391836734693878"
[1] "p 0.1"
[1] "dis_mean  0.1"
[1] "p*(1-p) 0.09"
```

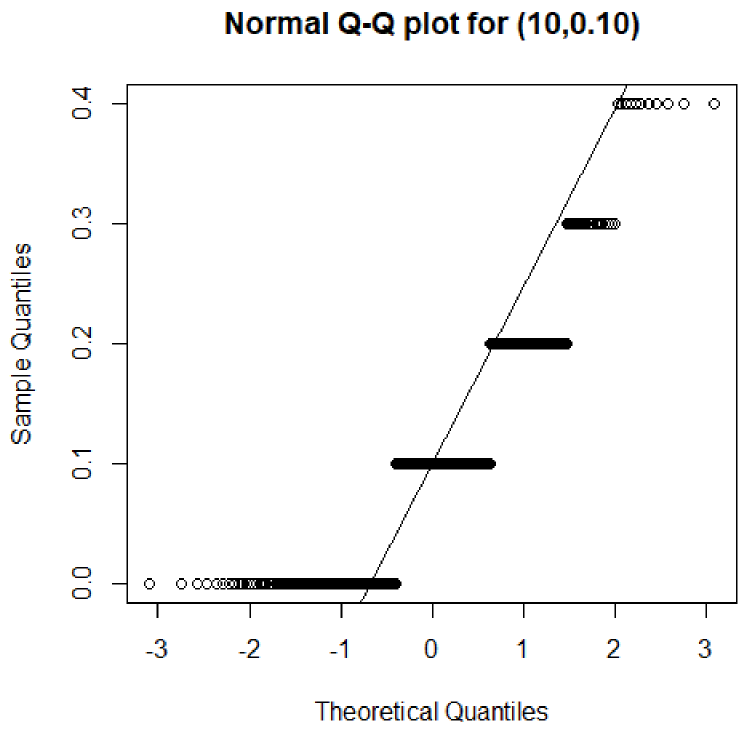
```
[1] "dis_var  0.0909090909090909"
[1] "p 0.25"
[1] "dis_mean  0.27"
[1] "p*(1-p) 0.1875"
[1] "dis_var  0.199090909090909"
[1] "p 0.5"
[1] "dis_mean  0.55"
[1] "p*(1-p) 0.25"
[1] "dis_var  0.25"
[1] "p 0.75"
[1] "dis_mean  0.76"
[1] "p*(1-p) 0.1875"
[1] "dis_var  0.184242424242424"
[1] "p 0.9"
[1] "dis_mean  0.93"
[1] "p*(1-p) 0.09"
[1] "dis_var  0.0657575757575758"
```



(b) For a given (n, p) combination, simulate 500 values of ^p, and make a normal Q - Q plot of the values. Does the distribution look approximately normal?

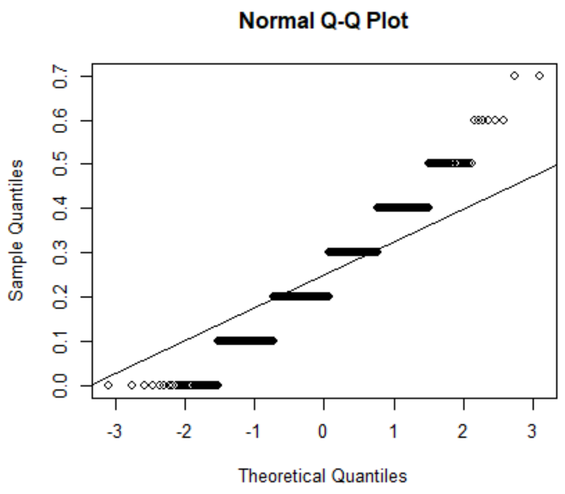
For a combination of (10,0.10) for (n,p), 500 values of ^p are simulated and Q-Q plot is made whose distribution does not look normal.

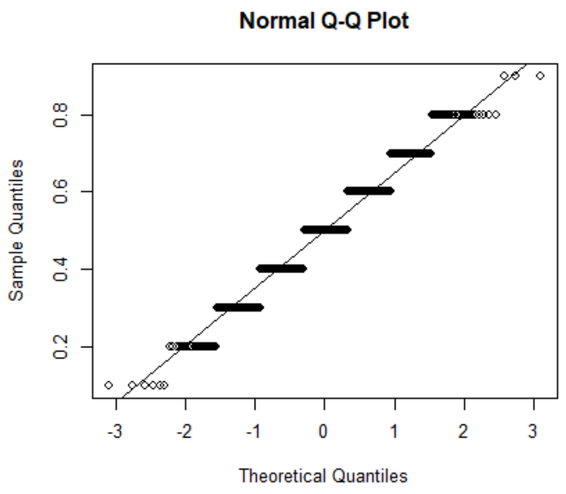
```
#2b 500 values of sample mean of X for (10,0.10) combination
re=replicate (500 , mean(rbinom(10,1,0.10)))
#Q-Q plot of the values
qqnorm (re,main = "Normal Q-Q plot for (10,0.10)")
qqline(re)
```

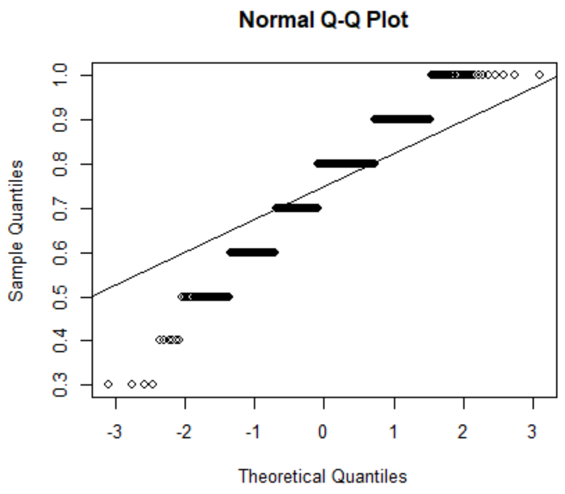


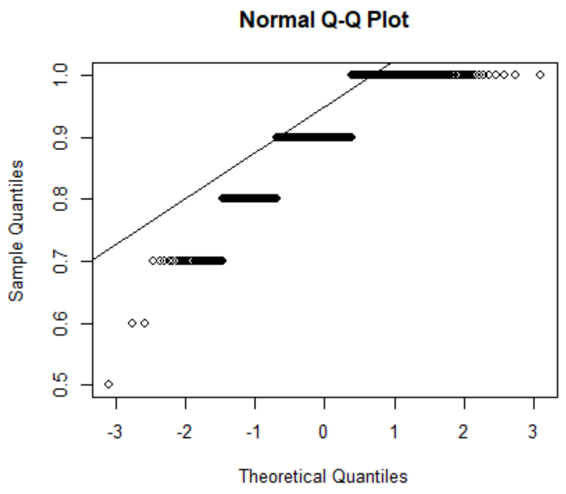
(c) Repeat (b) for the remaining combinations of (n, p) values.

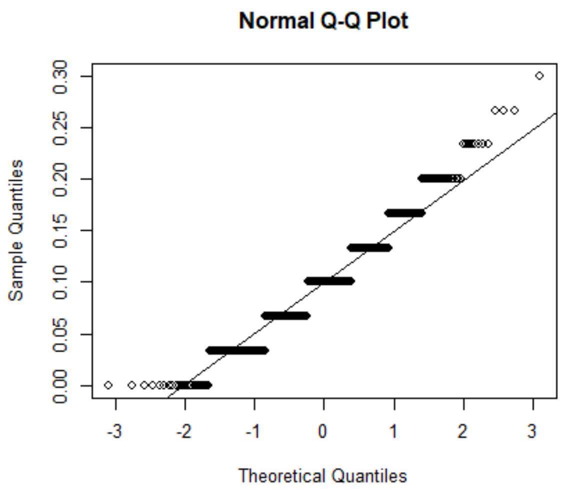
```
#2c 500 values of sample mean of X for remaining 19 combinations of (n,p)
par(mfrow=c(2,2)) #Creates 2x2 multi-paneled plotting window
for(i in n){
 for(j in p){
  if(i!=10 | j!=0.10)
  {
  #Simulating 500 values of other 19 combinations of ( n,p )
  r=replicate(500,mean(rbinom(i,1,j)))
  #Q-Q plot for 19 combinations of (n,p)
  qqnorm(r)
  qqline(r)
  } } }
```

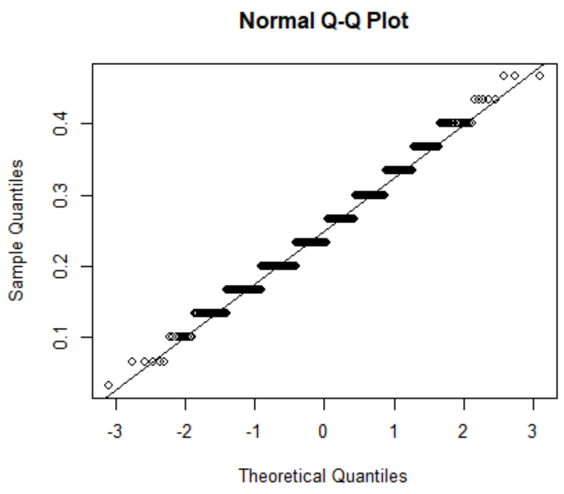

Normal Q-Q Plot
Sample Quantiles
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
-3
-2
-1
0
1
2
3
Theoretical Quantiles

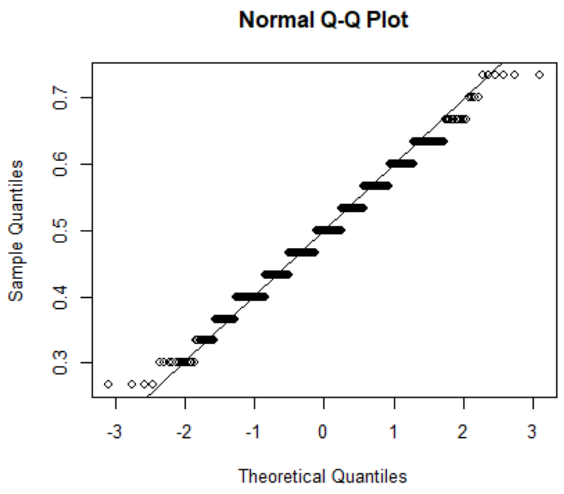

Normal Q-Q Plot
Sample Quantiles
0.2
0.4
0.6
0.8
-3
-2
-1
0
1
2
3
Theoretical Quantiles

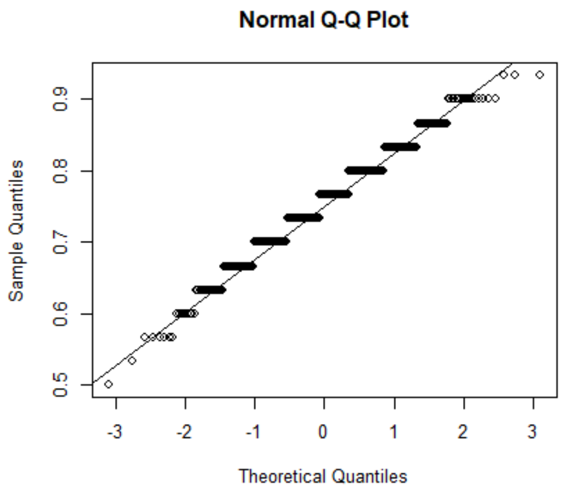

Normal Q-Q Plot
Sample Quantiles
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
-3
-2
-1
0
1
2
3
Theoretical Quantiles

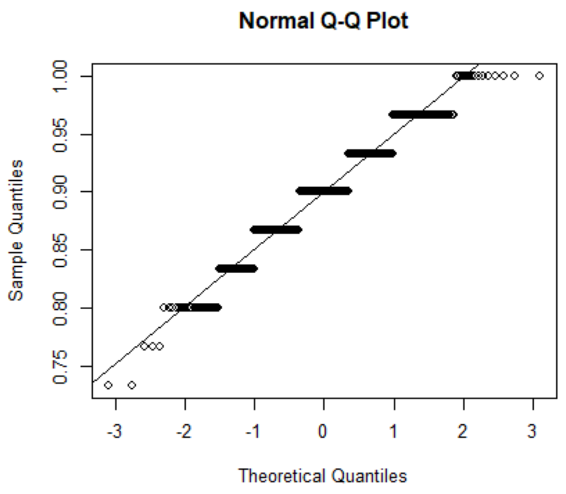

Normal Q-Q Plot
Sample Quantiles
0.5
0.6
0.7
0.8
0.9
1.0
-3
-2
-1
0
1
2
3
Theoretical Quantiles

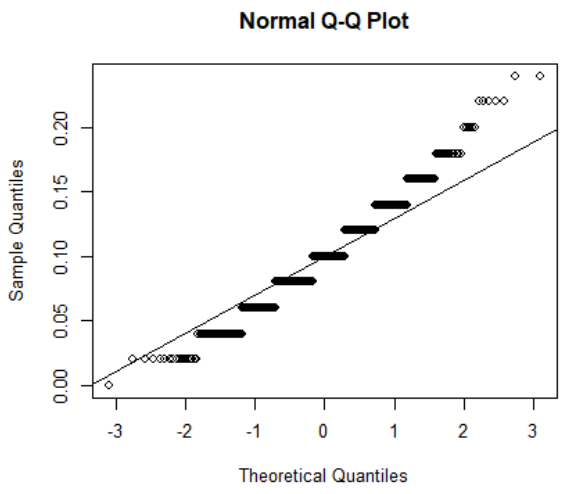

Normal Q-Q Plot
Sample Quantiles
0.00
0.05
0.10
0.15
0.20
0.25
0.30
-3
-2
-1
0
1
2
3
Theoretical Quantiles

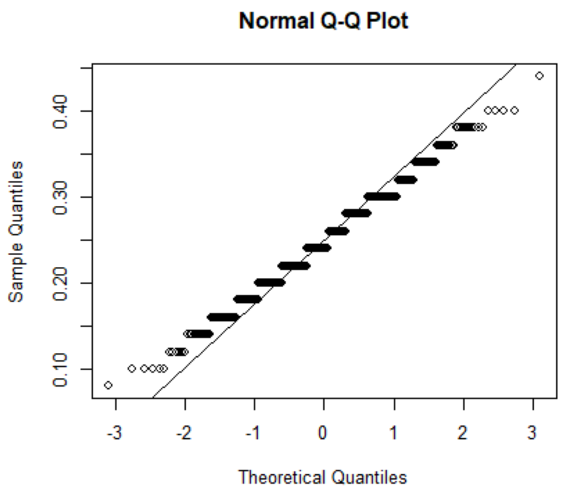

Normal Q-Q Plot
Sample Quantiles
0.1
0.2
0.3
0.4
-3
-2
-1
0
1
2
3
Theoretical Quantiles

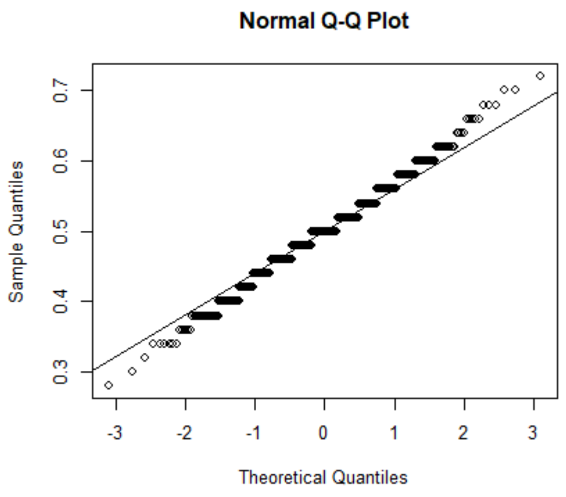

Normal Q-Q Plot
Sample Quantiles
0.3
0.4
0.5
0.6
0.7
-3
-2
-1
0
1
2
3
Theoretical Quantiles

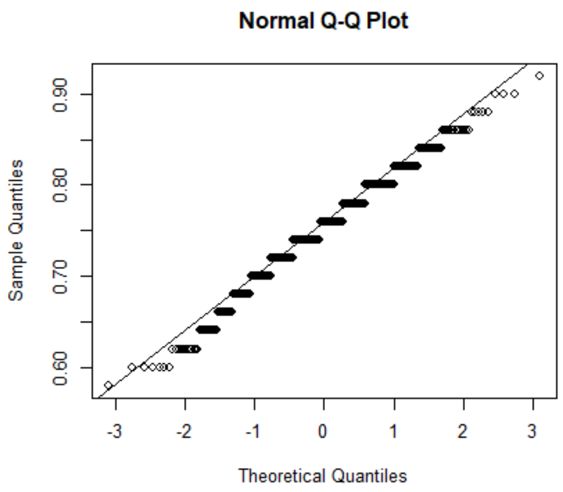

Normal Q-Q Plot
Sample Quantiles
0.5
0.6
0.7
0.8
0.9
-3
-2
-1
0
1
2
3
Theoretical Quantiles

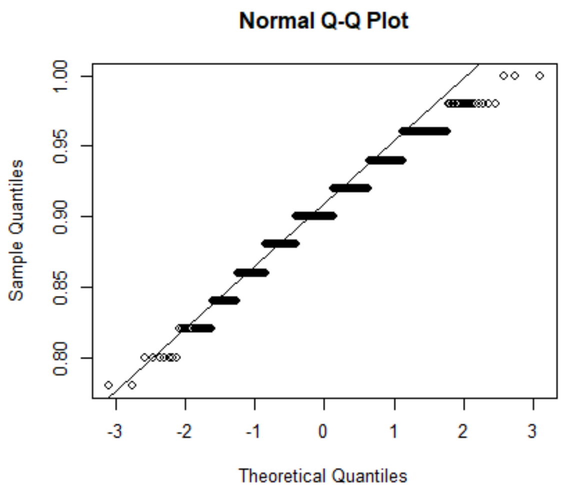

Normal Q-Q Plot
Sample Quantiles
Theoretical Quantiles
0.75
0.80
0.85
0.90
0.95
1.00
-3
-2
-1
0
1
2
3

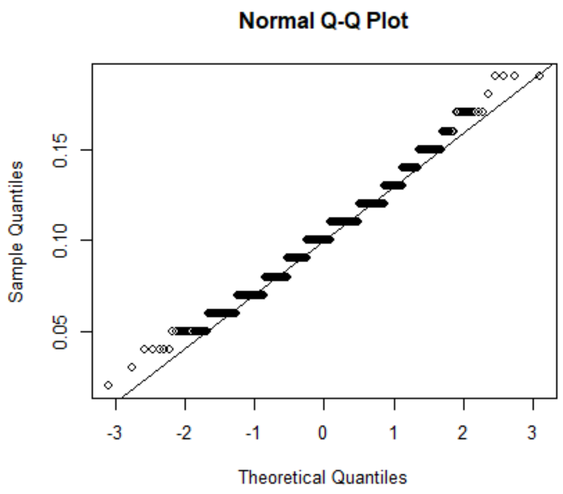

Normal Q-Q Plot
Sample Quantiles
Theoretical Quantiles
0.00
0.05
0.10
0.15
0.20
-3
-2
-1
0
1
2
3

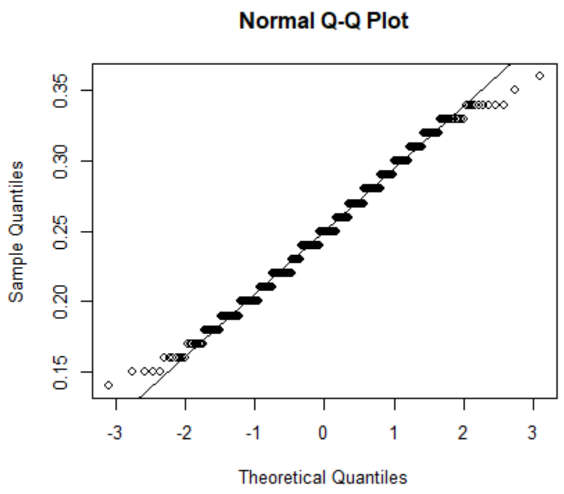

Normal Q-Q Plot
Sample Quantiles
Theoretical Quantiles
0.10
0.20
0.30
0.40
-3
-2
-1
0
1
2
3

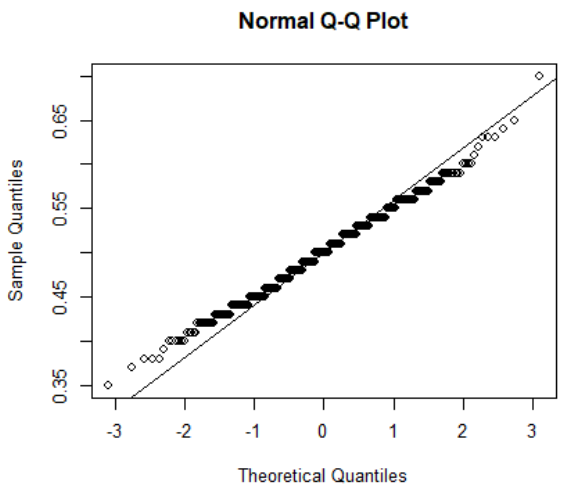

Normal Q-Q Plot
Sample Quantiles
Theoretical Quantiles
0.3
0.4
0.5
0.6
0.7
-3
-2
-1
0
1
2
3

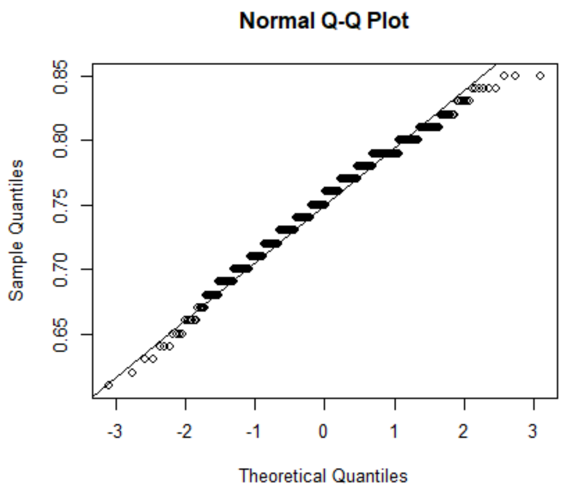

Normal Q-Q Plot
Sample Quantiles
0.60
0.70
0.80
0.90
Theoretical Quantiles
-3
-2
-1
0
1
2
3

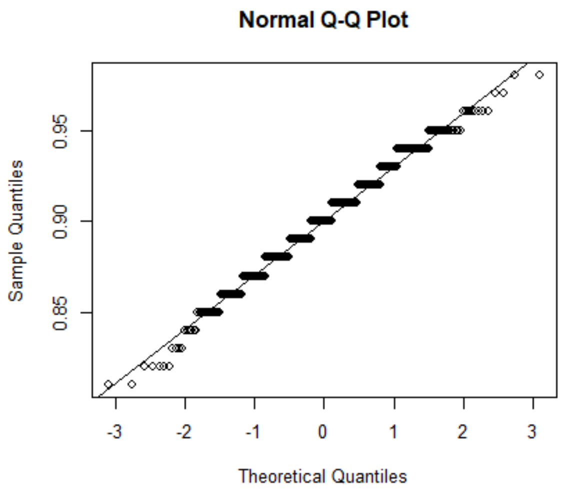

Normal Q-Q Plot
Sample Quantiles
0.80
0.85
0.90
0.95
1.00
Theoretical Quantiles
-3
-2
-1
0
1
2
3


Normal Q-Q Plot
Sample Quantiles
0.05
0.10
0.15
Theoretical Quantiles
-3
-2
-1
0
1
2
3


Normal Q-Q Plot
Sample Quantiles
0.15
0.20
0.25
0.30
0.35
Theoretical Quantiles
-3
-2
-1
0
1
2
3



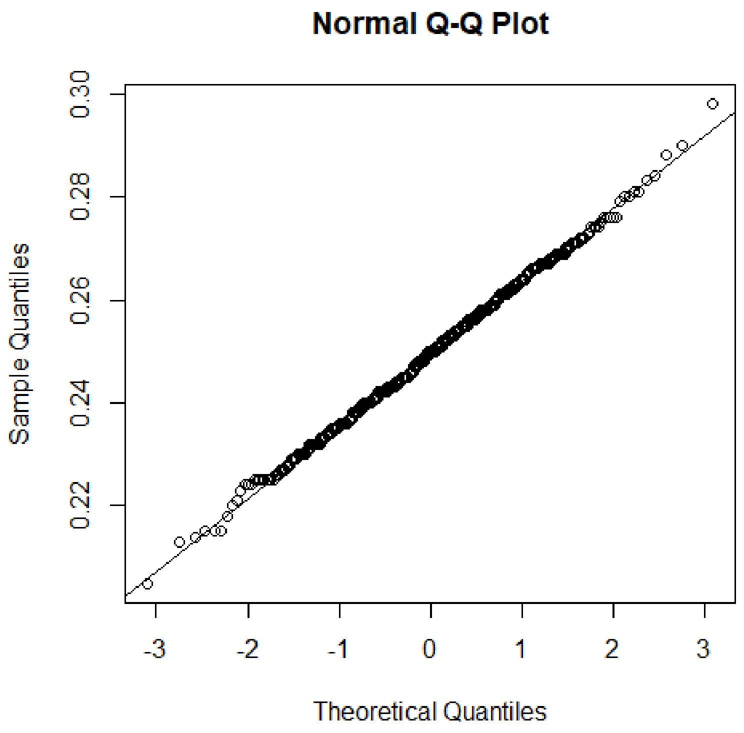




(d) What would you say about how large n should be for the approximation to be good? Does this answer depend on p? Justify your conclusions.

For the given distribution,n=1000 seems to be good and it also depends on values of p. If p has extreme values then we require higher n values because it is not symmetric.

```
repli=replicate (500, mean(rbinom(1000,1,0.25)))
qqnorm(repli)
qqline(repli)
```


Normal Q-Q Plot
Sample Quantiles
0.22
0.24
0.26
0.28
0.30
-3
-2
-1
0
1
2
3
Theoretical Quantiles