Statistical Inference 18225012

Colm Kenny

15/12/2020

Q1 - Estimation bias and efficiency

i

```
meanpois <- function(n, lambda)
{mean_p <- mean(rpois(n, lambda))
return(mean_p)}
```

This code is used to generate a random sample of poisson data and calculate the mean of the data.The function meanpois() needs 2 arguments, a number n(size of sample), and lambda/rate.

ii

```
set.seed(18225012)
meanpois(10, 1)
```

```
## [1] 0.7
```

This code uses function meanpois with arguments n=10 and lambda=1 to create a sample mean for poisson data. set.seed is used so results are equal when we run it numerous times, as it is a random generator.

iii

```
reps <- 1000
set.seed(18225012)
vec_pois <- replicate(reps, meanpois(10,1))
mean(vec_pois)
```

```
## [1] 0.9763
```

```
var(vec_pois, y=NULL)
```

```
## [1] 0.09962794
```

The code above produces a vector of 1000 sample means (a collection of random meanpois results using the same arguments). We are able to calculate the mean and variance of these means using commands in R. The MSE is done by hand where we have to find the bias of the estimator $\bar{x}$.

$MSE(\bar{x}) = Var(\bar{x}) + [bias(\bar{x})]^2$

$bias(\bar{x}) = E(\bar{x}) - \mu$

$E(\bar{x}) = E(\frac{x_1+x_2+...x_n}{n})$

As the samples mean are identically distributed, they share the same mean.

$E(\bar{x}) = \frac{1}{n}[\mu + \mu + ...\mu]$

$= \frac{1}{n}[n\mu] = \mu$

$\therefore bias = \mu - \mu = 0$

$MSE(\bar{x}) = Var(\bar{x}) = 0.09962794$

iv

```
#n=20
set.seed(18225012)
vec_pois20 <- replicate(reps, meanpois(20,1))
mean(vec_pois20)
```

```
## [1] 0.97405
```

```
var(vec_pois20, y=NULL)
```

```
## [1] 0.05154064
```

```
#find MSE
#need bias
#n=50
set.seed(18225012)
vec_pois50 <- replicate(reps, meanpois(50,1))
mean(vec_pois50)
```

```
## [1] 0.98838
```

```
var(vec_pois50, y=NULL)
```

```
## [1] 0.02086144
```

```
#n=200
set.seed(18225012)
vec_pois200 <- replicate(reps, meanpois(200,1))
mean(vec_pois200)
```

```
## [1] 0.99696
```

```r
var(vec_pois200, y=NULL)
```

```
## [1] 0.005316975
```

```r
#n=400
set.seed(18225012)
vec_pois400 <- replicate(reps, meanpois(400,1))
mean(vec_pois400)
```

```
## [1] 0.9985225
```

```r
var(vec_pois400, y=NULL)
```

```
## [1] 0.002777501
```

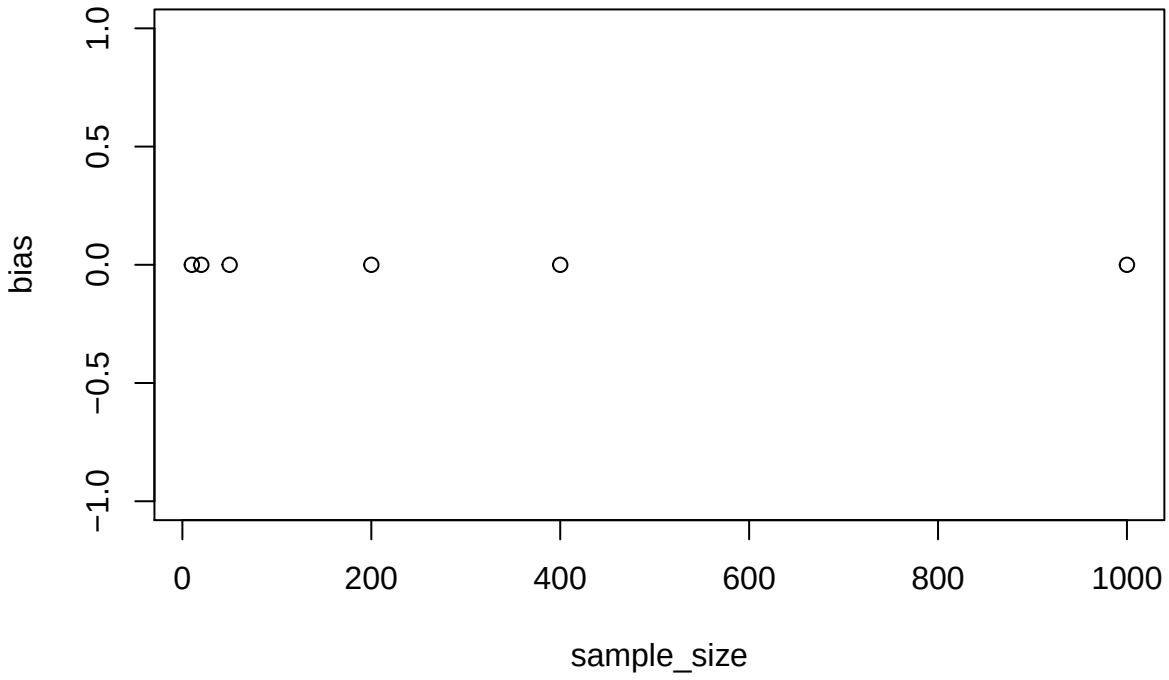
```r
#n=1000
set.seed(18225012)
vec_pois1000 <- replicate(reps, meanpois(1000,1))
mean(vec_pois1000)
```

```
## [1] 0.999393
```

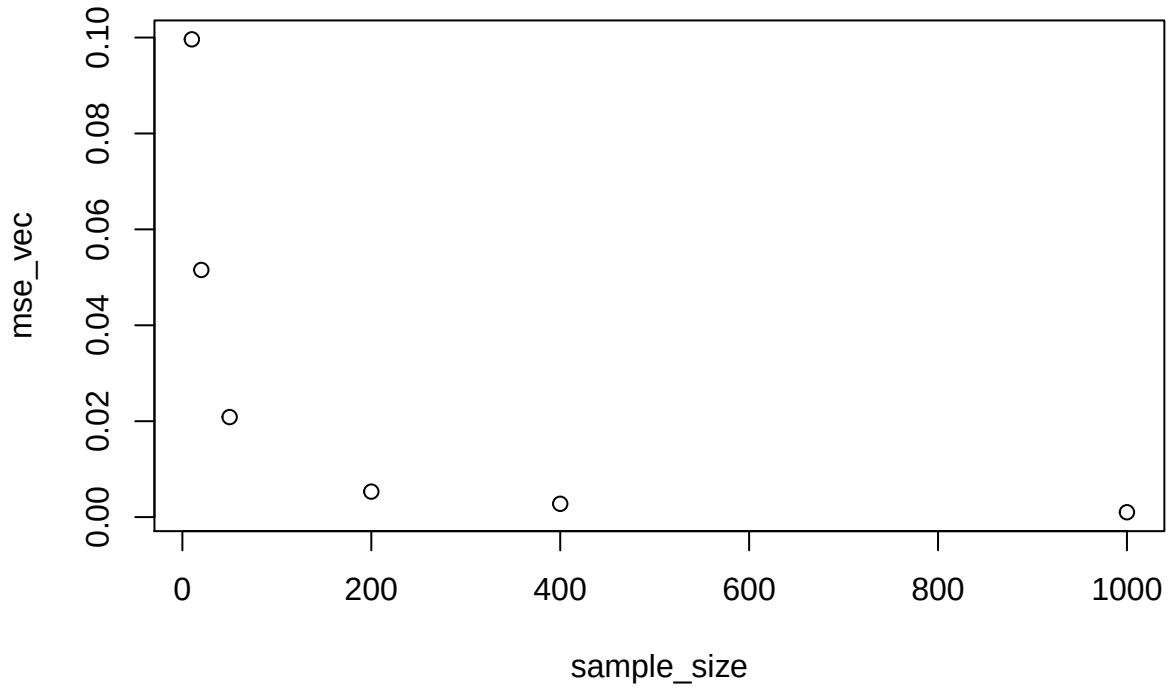
```r
var(vec_pois1000, y=NULL)
```

```
## [1] 0.001014231
```

```r
#bias vs sample size, size=x bias=y
sample_size <- c(10,20,50,200,400,1000)
bias <- c(0,0,0,0,0,0)
plot(sample_size, bias)
```




```
#mse v sample size = variance v sample size
mse_vec <- c(var(vec_pois), var(vec_pois20), var(vec_pois50), var(vec_pois200), var(vec_pois400), var(v
plot(sample_size, mse_vec)
```

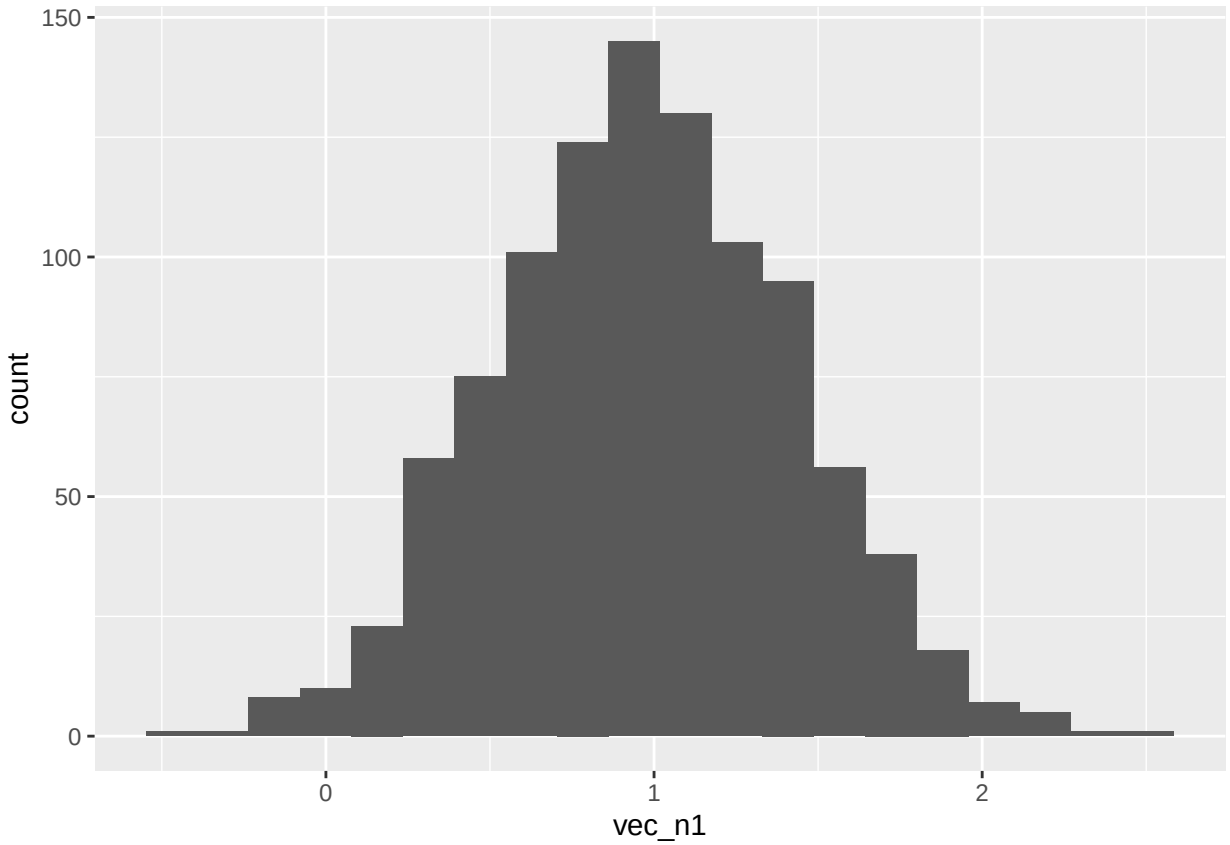


As the bias = 0, bias does not change with varying sample sizes. However, this is not the case with the MSE, as this decreases as sample size decreases. This makes sense, as the sample size increases the data becomes increasingly similar to the population data. therefore less error.

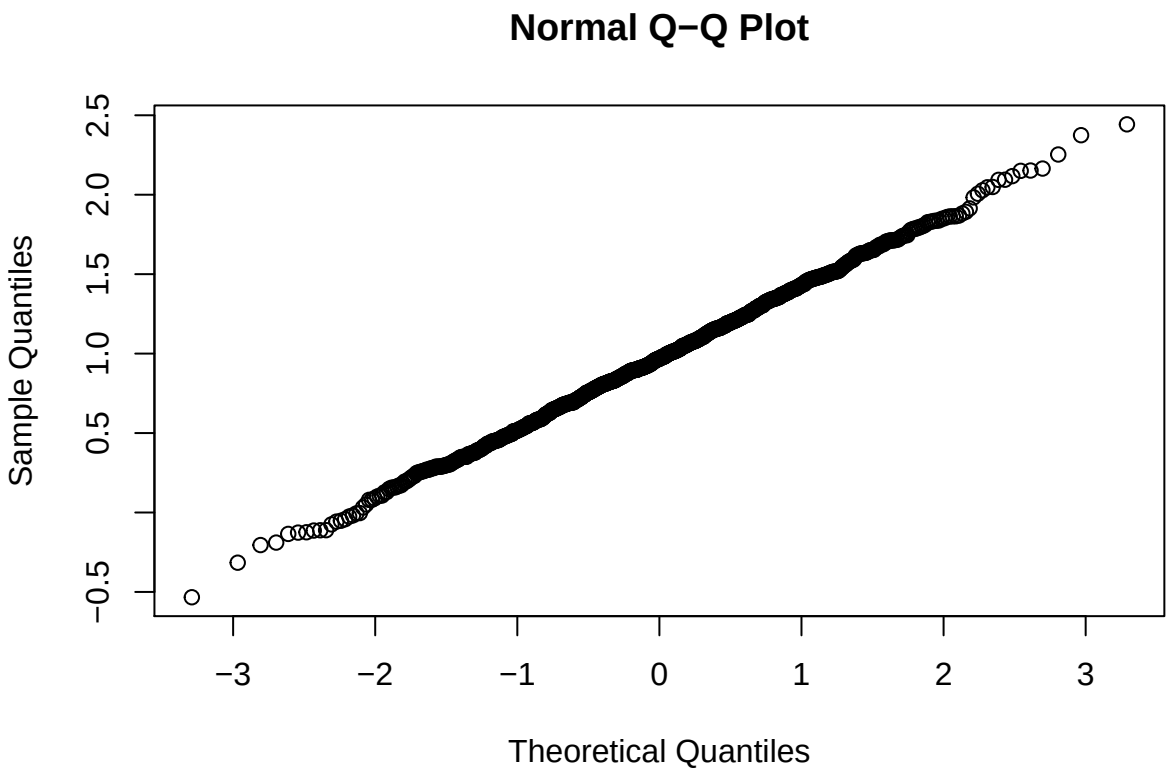
Q2 - Central Limit Theorem

i

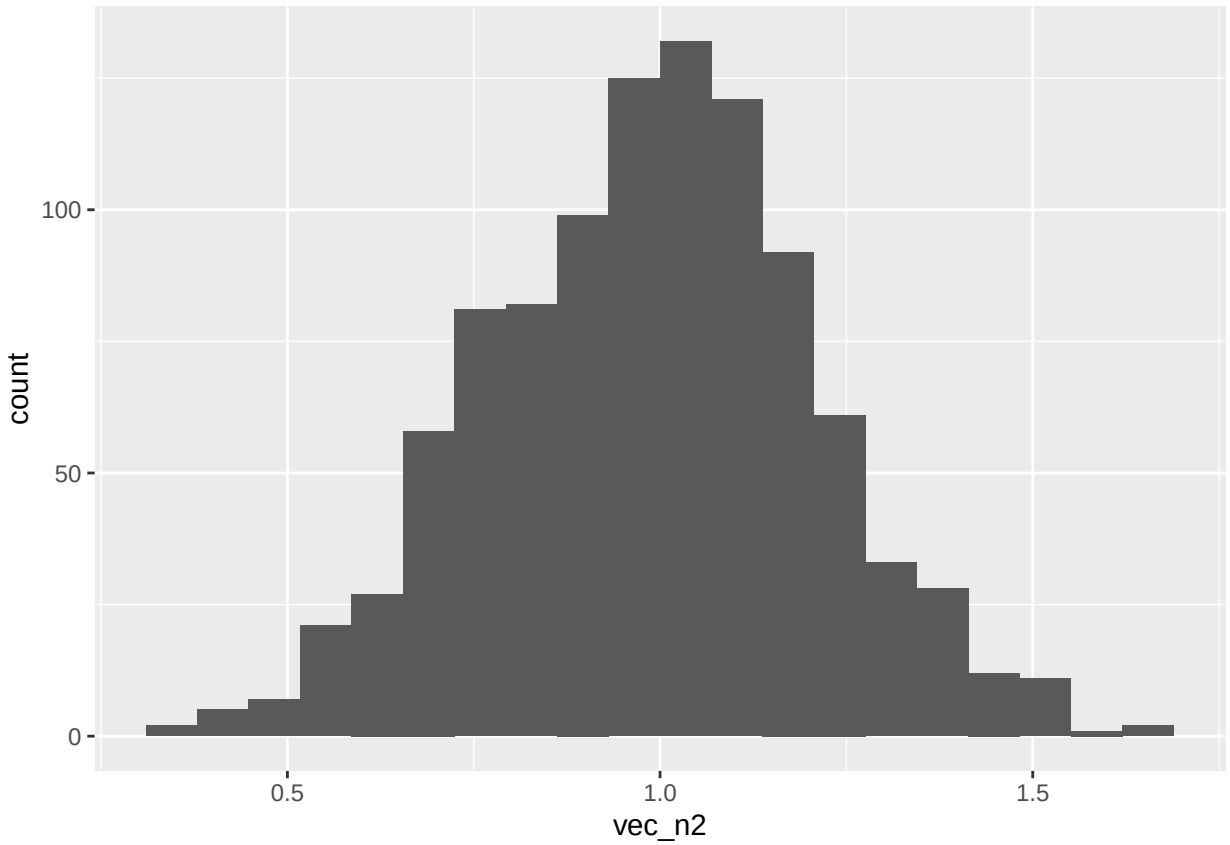
```
# n =5
mean_norm <- function(n, u, s)
{mean_norm <- mean(rnorm(n, u, s))
return(mean_norm)}
set.seed(18225012)
n1 <- mean_norm(5, 1, 1)
vec_n1 <- replicate(reps, mean_norm(5, 1, 1))
ggplot(data = data.frame(vec_n1), aes(x = vec_n1)) +
  geom_histogram(bins=20)
```



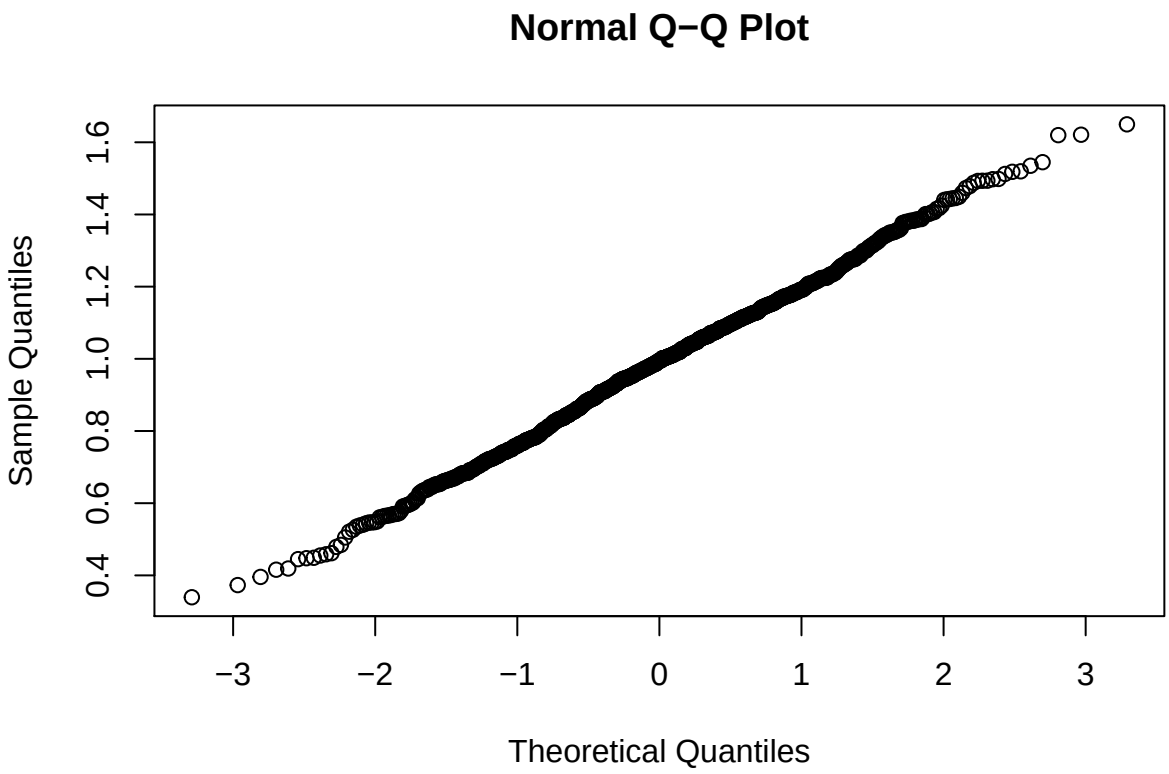

```
qqnorm(vec_n1) #plots suggest normally distributed
```



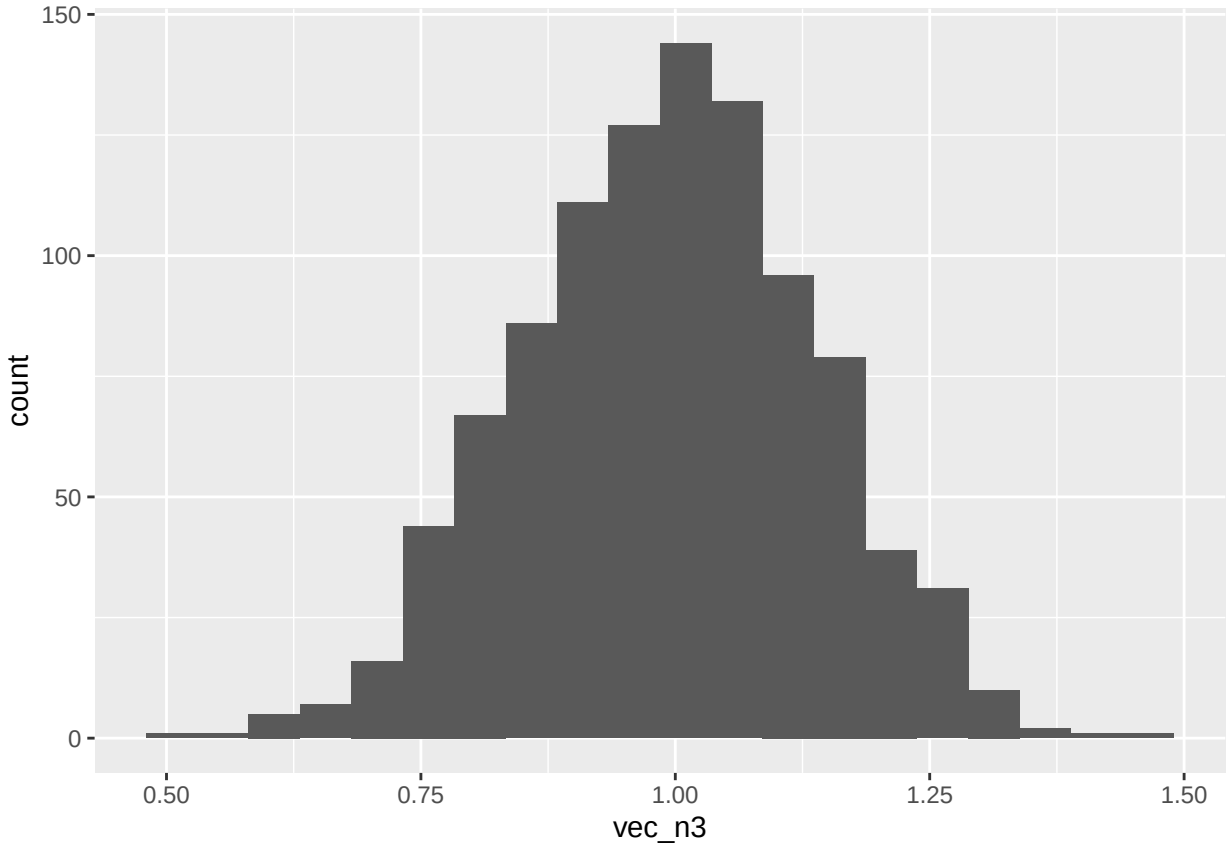

```
# n = 20
set.seed(18225012)
n2 <- mean_norm(20, 1, 1)
vec_n2 <- replicate(reps, mean_norm(20, 1, 1))
ggplot(data = data.frame(vec_n2), aes(x = vec_n2)) +
  geom_histogram(bins = 20)
```



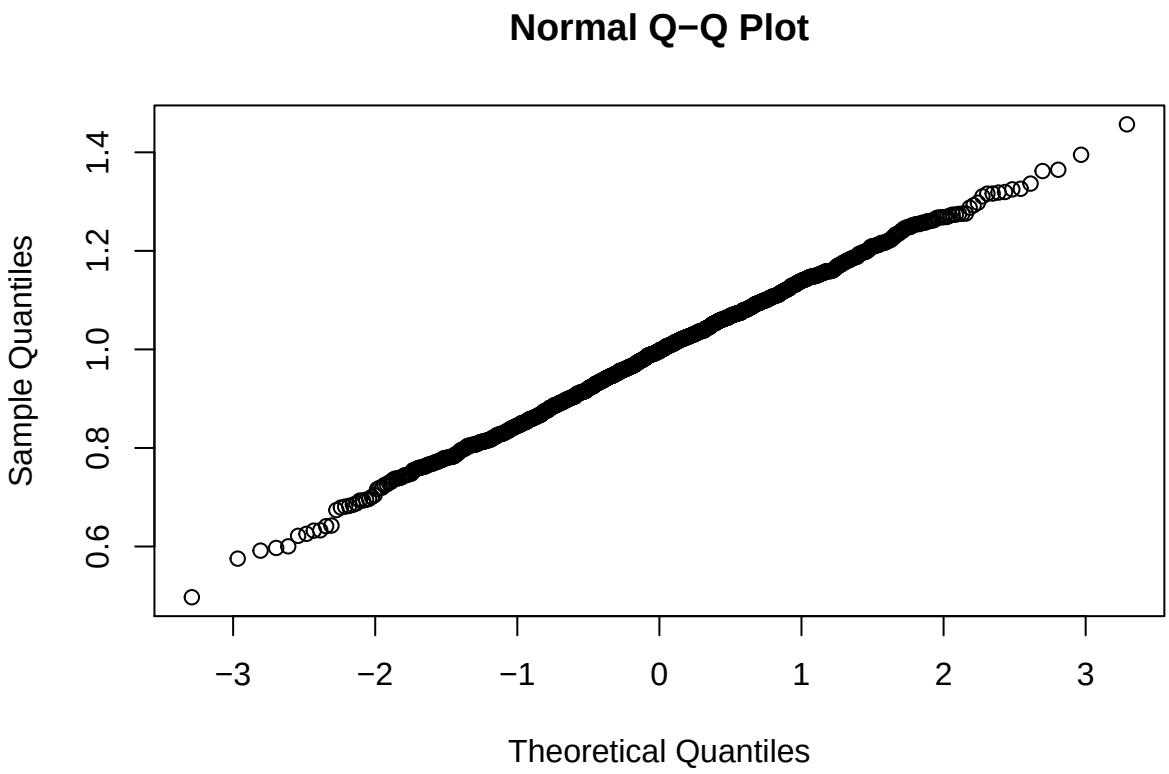

```
qqnorm(vec_n2)
```




```r
# n = 50
set.seed(18225012)
n3 <- mean_norm(50, 1, 1)
vec_n3 <- replicate(reps, mean_norm(50, 1, 1))
ggplot(data = data.frame(vec_n3), aes(x = vec_n3)) +
  geom_histogram(bins = 20)
```




```
qqnorm(vec_n3)
```



These are all normally distributed, as they should be, they are derived from that distribution.

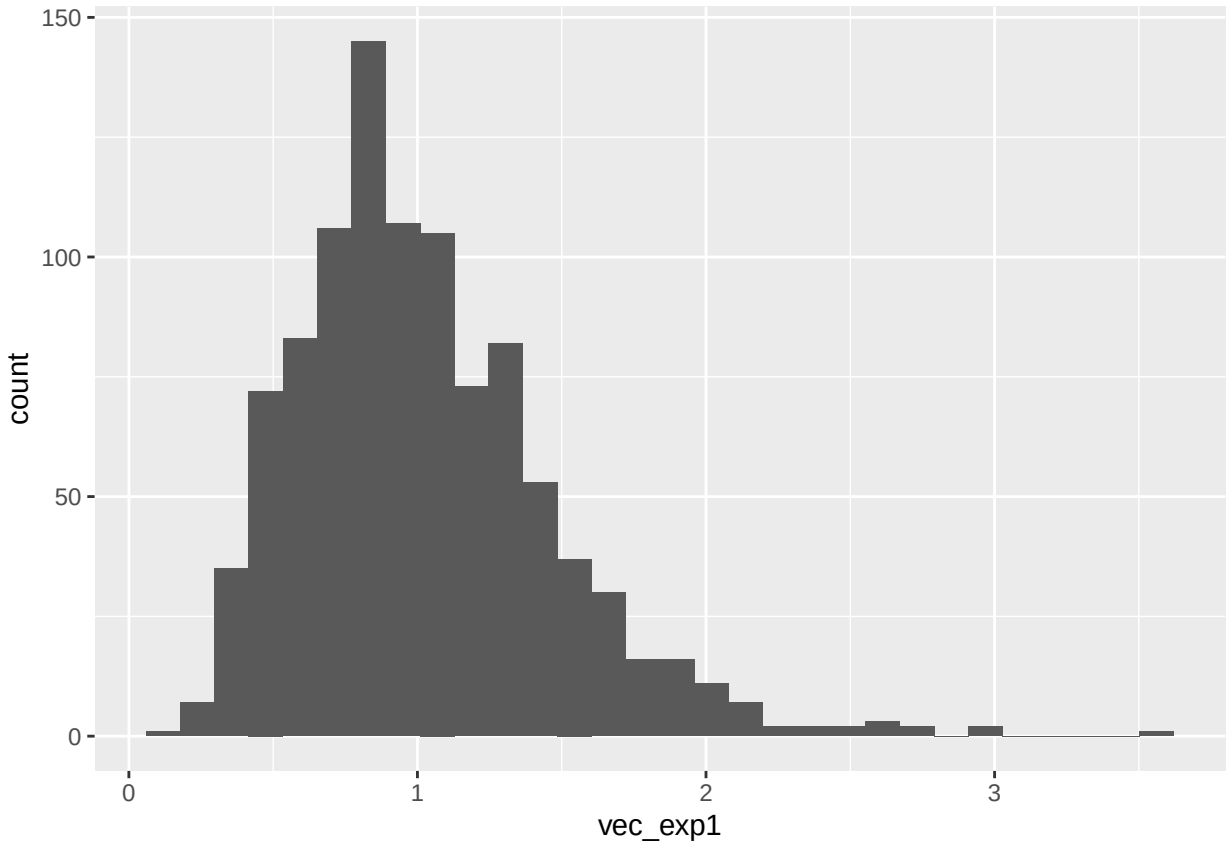
ii

```
ii ~exp(lambda=1)
```

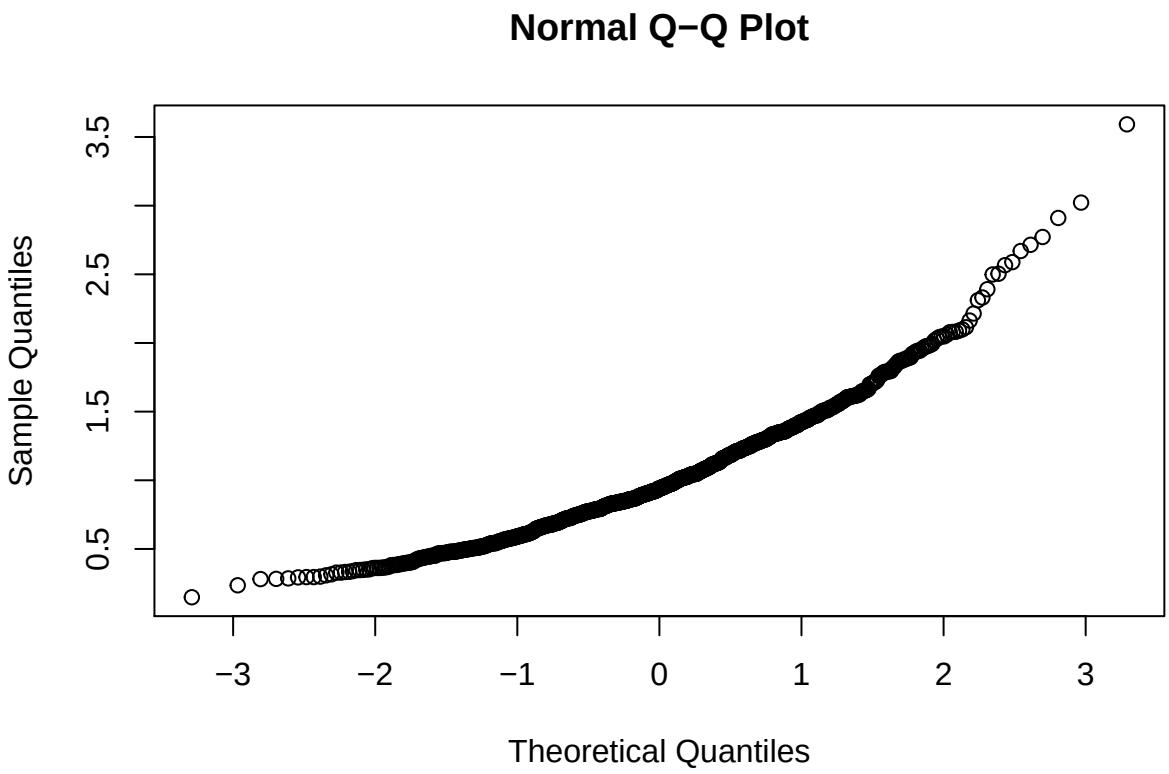
```
## ii ~ exp(lambda = 1)
```

```
set.seed(18225012)
meanexp <- function(n, lambda)
{mean_exp <- mean(rexp(n, lambda))
return(mean_exp)}
vec_exp1 <- replicate(reps, meanexp(5,1))
ggplot(data = data.frame(vec_exp1), aes(x = vec_exp1)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

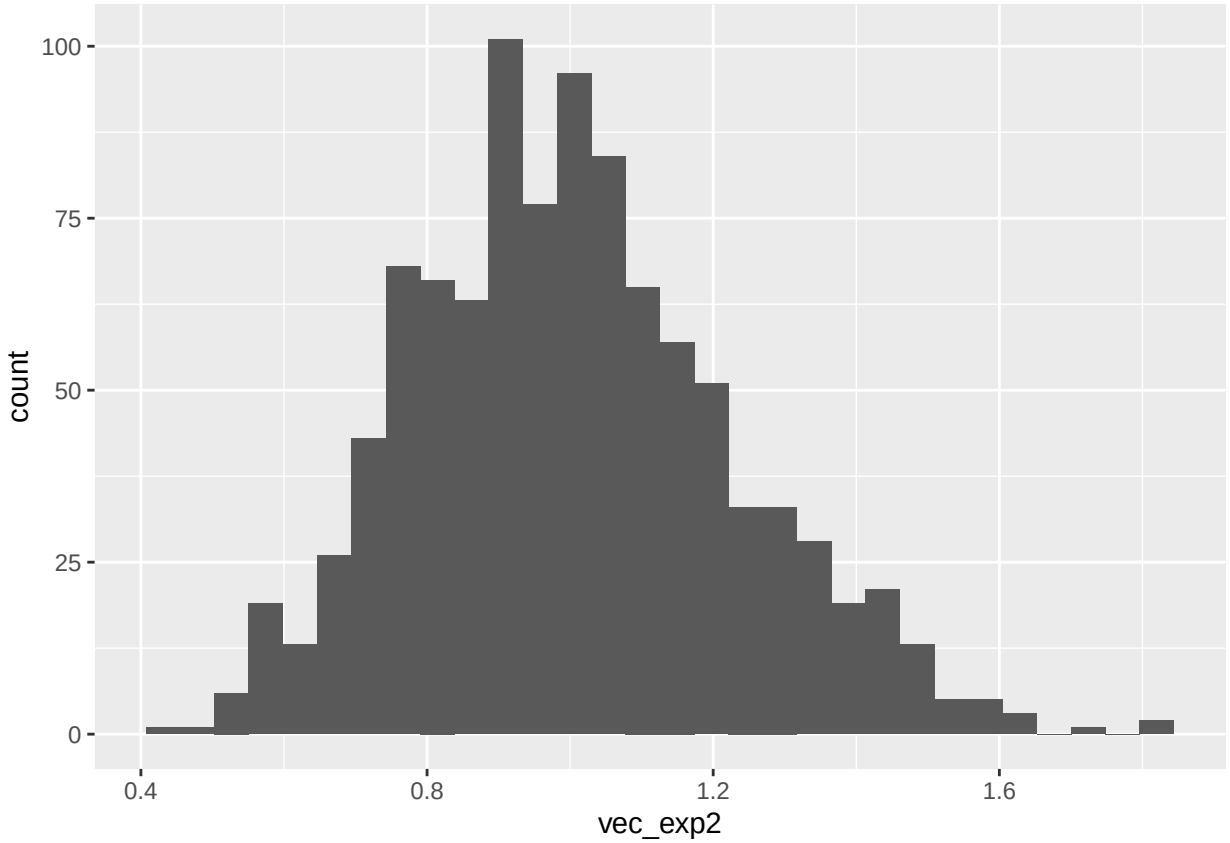



```
qqnorm(vec_exp1)
```

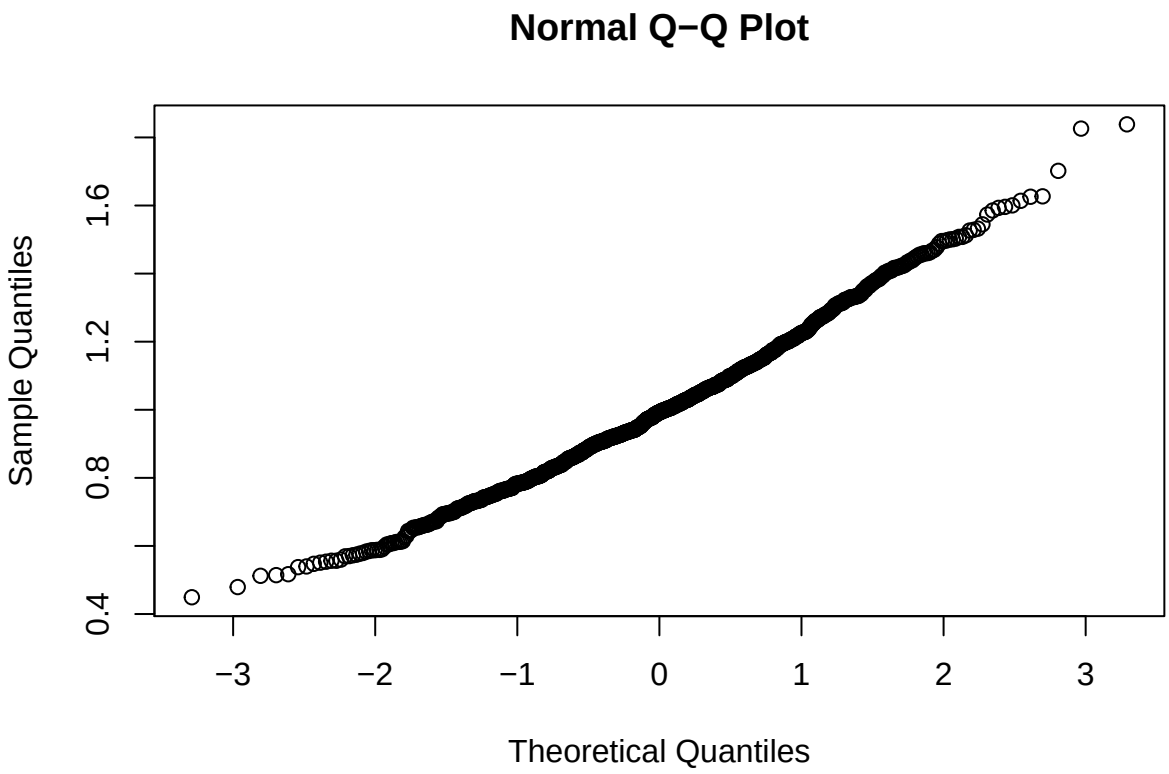



```
#not normally distributed

#n=20
set.seed(18225012)
vec_exp2 <- replicate(reps, meanexp(20,1))
ggplot(data = data.frame(vec_exp2), aes(x = vec_exp2)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

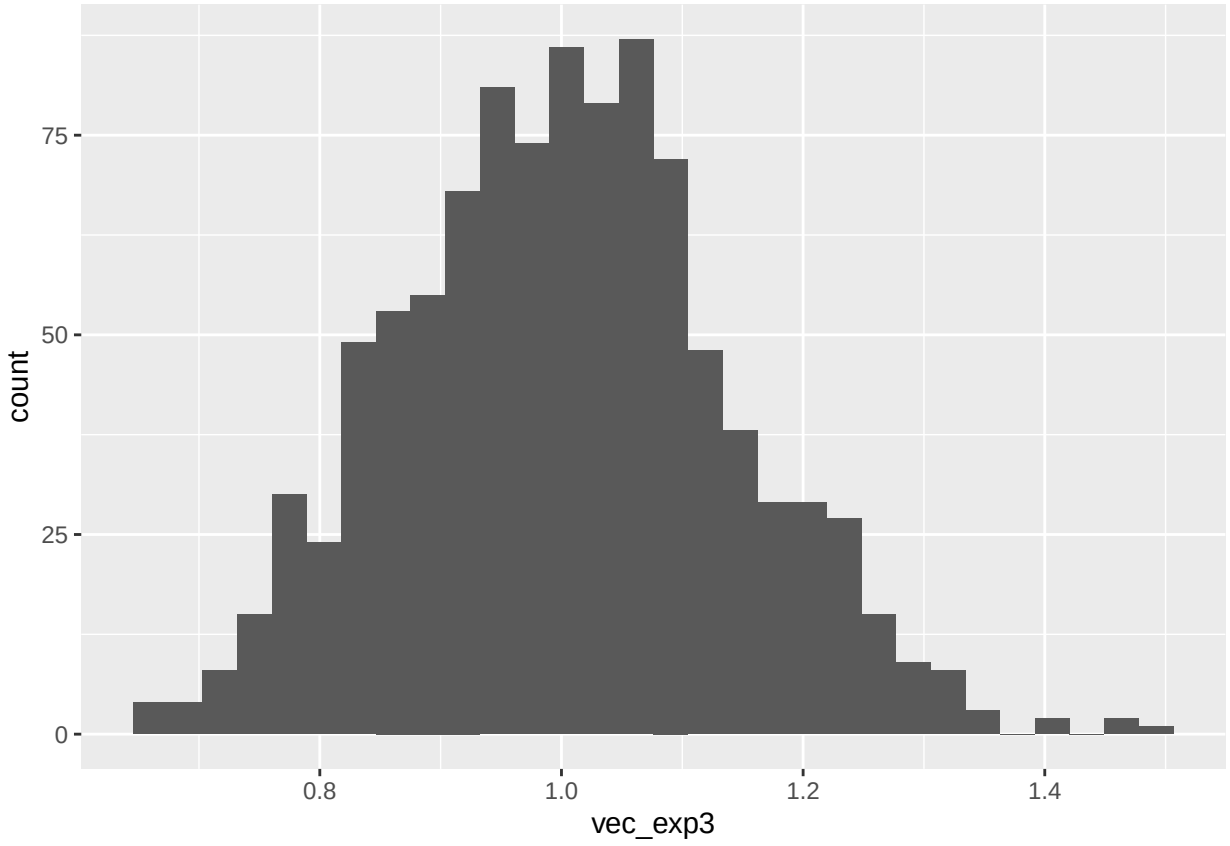



```
qqnorm(vec_exp2)#improvement from n=5 but still not normally distributed
```

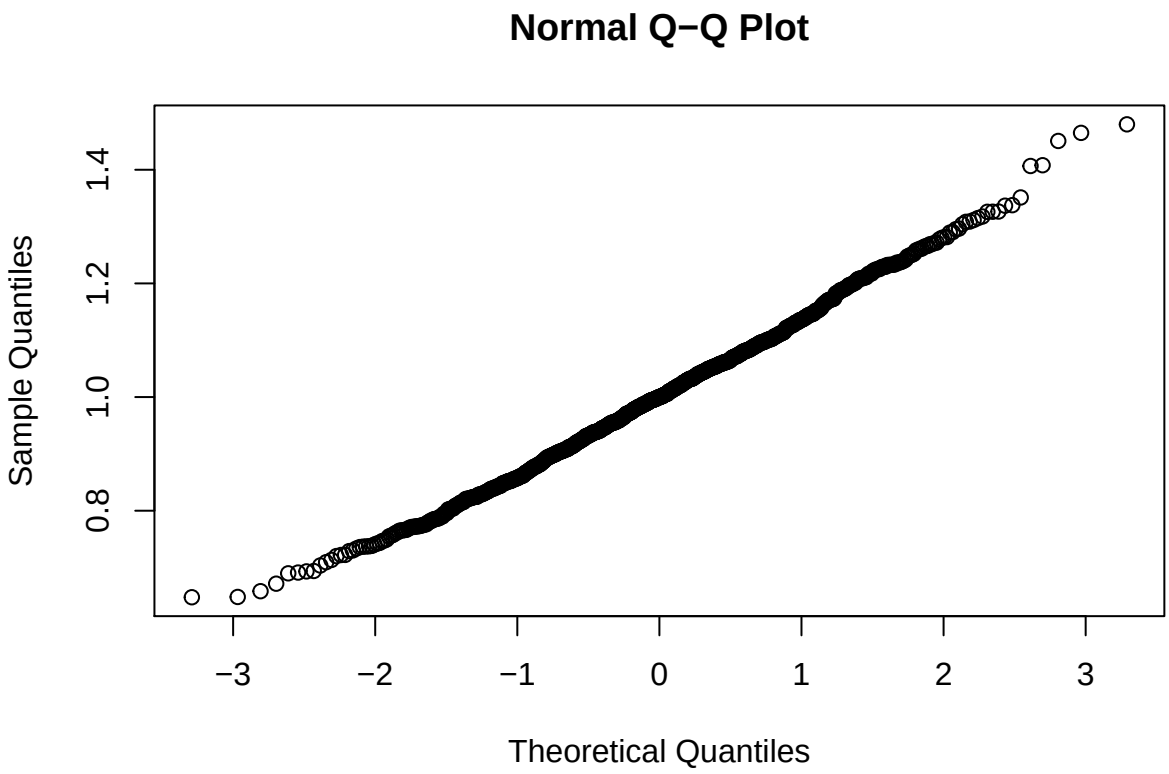



```r
#n=50
set.seed(18225012)
vec_exp3 <- replicate(reps, meanexp(50,1))
ggplot(data = data.frame(vec_exp3), aes(x = vec_exp3)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



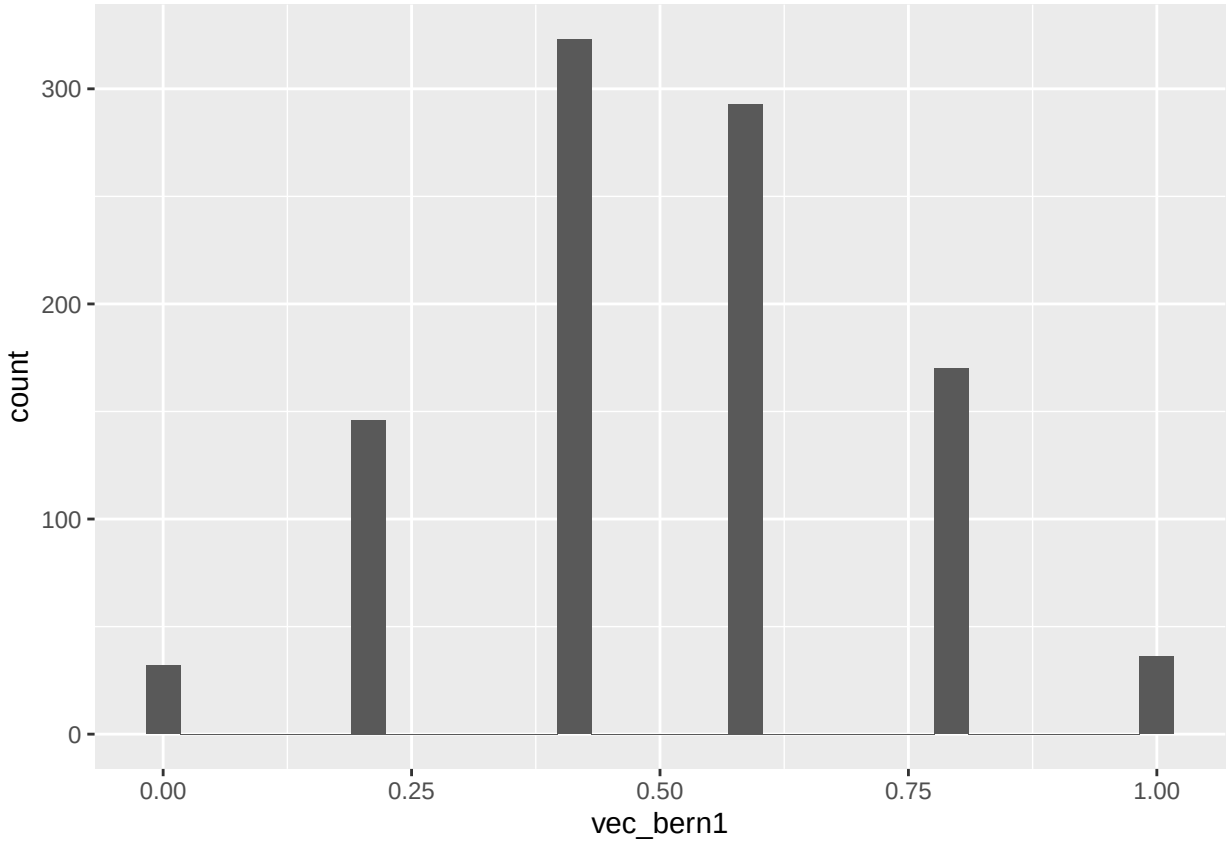

```
qqnorm(vec_exp3)#normally distributed
```



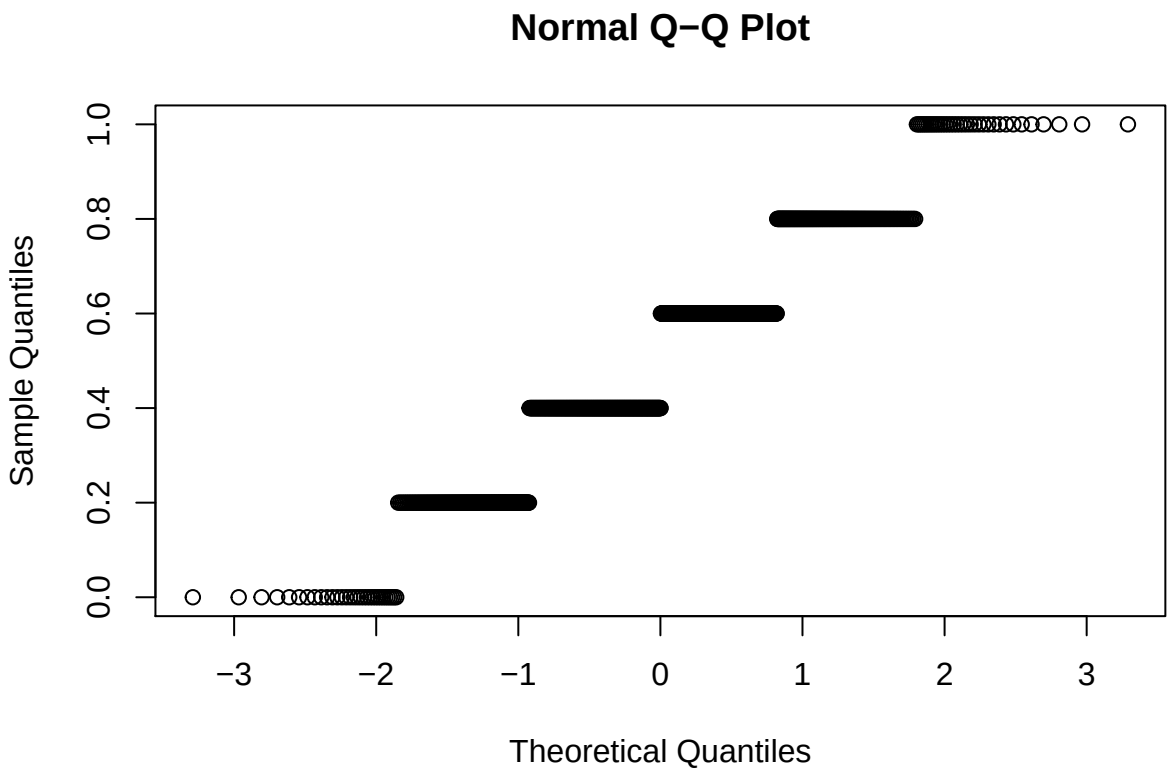
As n increases, the graphs become closer to a normal distribution, this supports the Central Limit Theorem

```
#bernoulli p=0.5
#50/50 chance n=5
set.seed(18225012)
vec_bern1 <- replicate(reps, mean(sample(c(0, 1), size = 5, replace = TRUE)))
ggplot(data = data.frame(vec_bern1), aes(x = vec_bern1)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

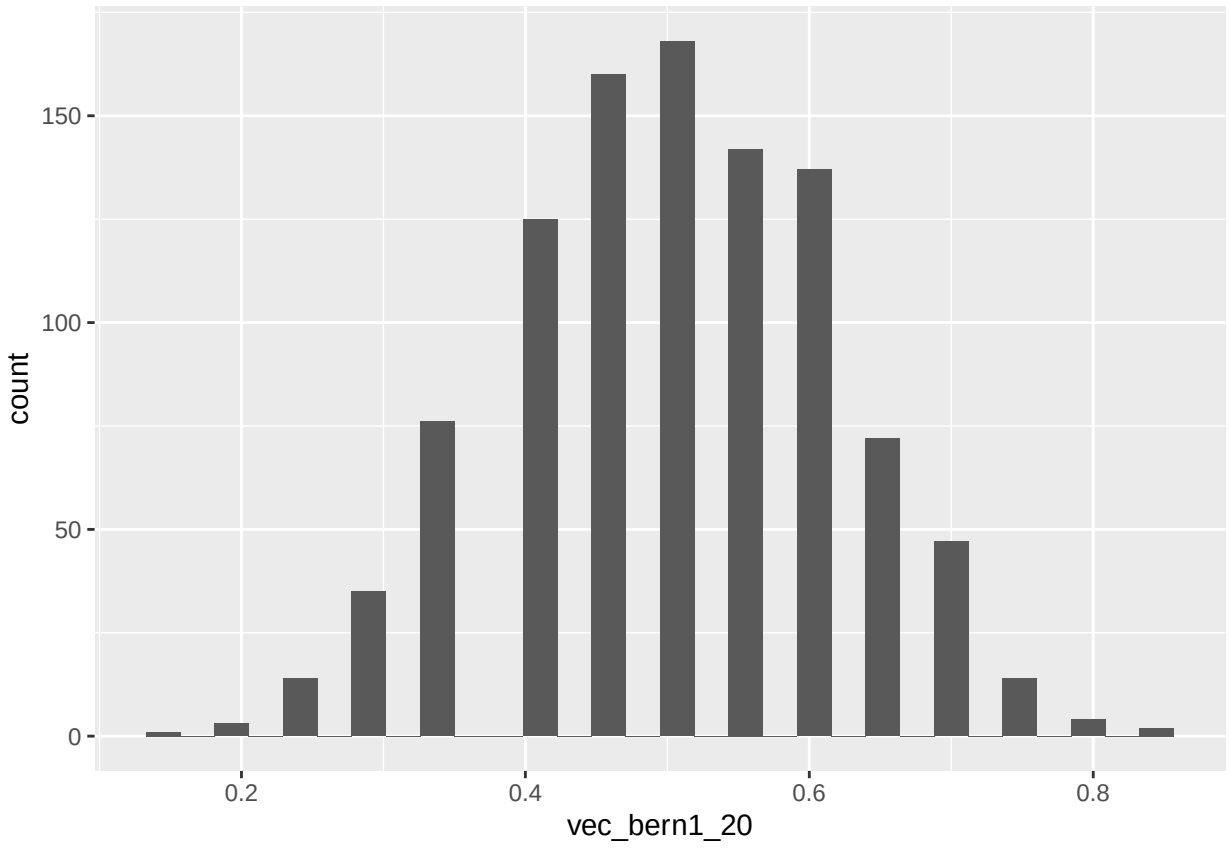



```
qqnorm(vec_bern1)
```

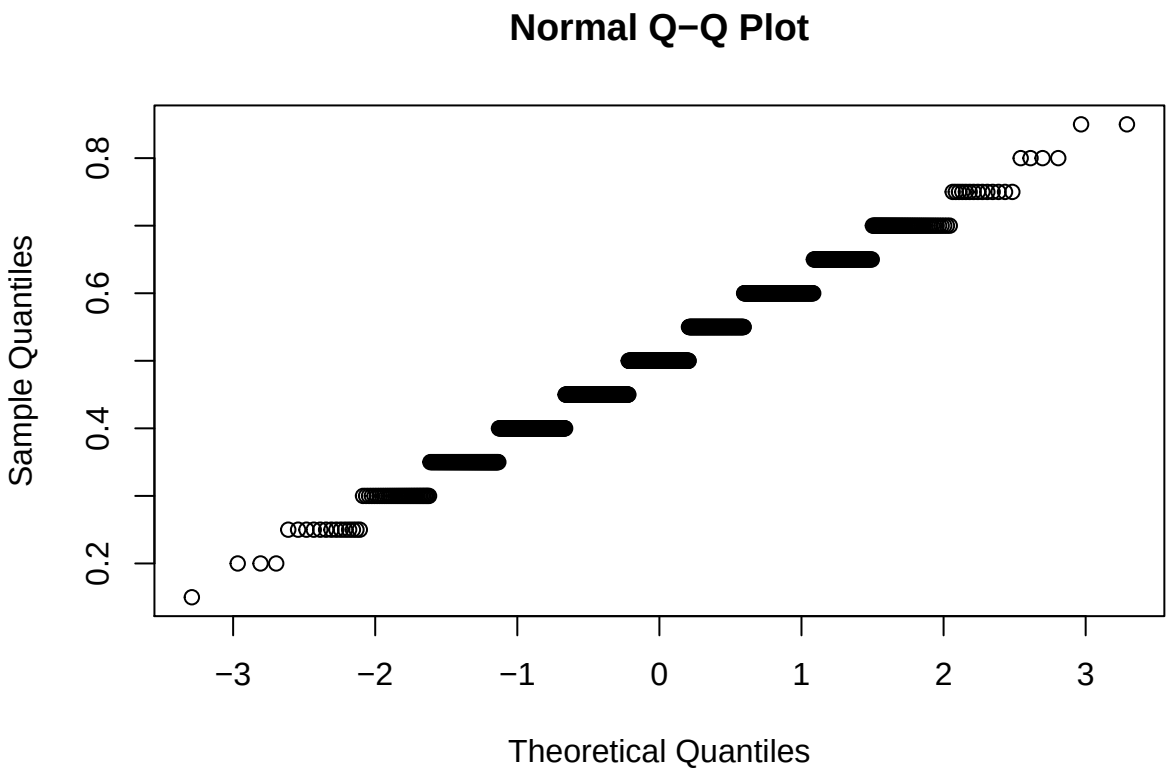



```
#n=20
set.seed(18225012)
vec_bern1_20 <- replicate(reps, mean(sample(c(0, 1), size = 20, replace = TRUE)))
ggplot(data = data.frame(vec_bern1_20), aes(x = vec_bern1_20)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

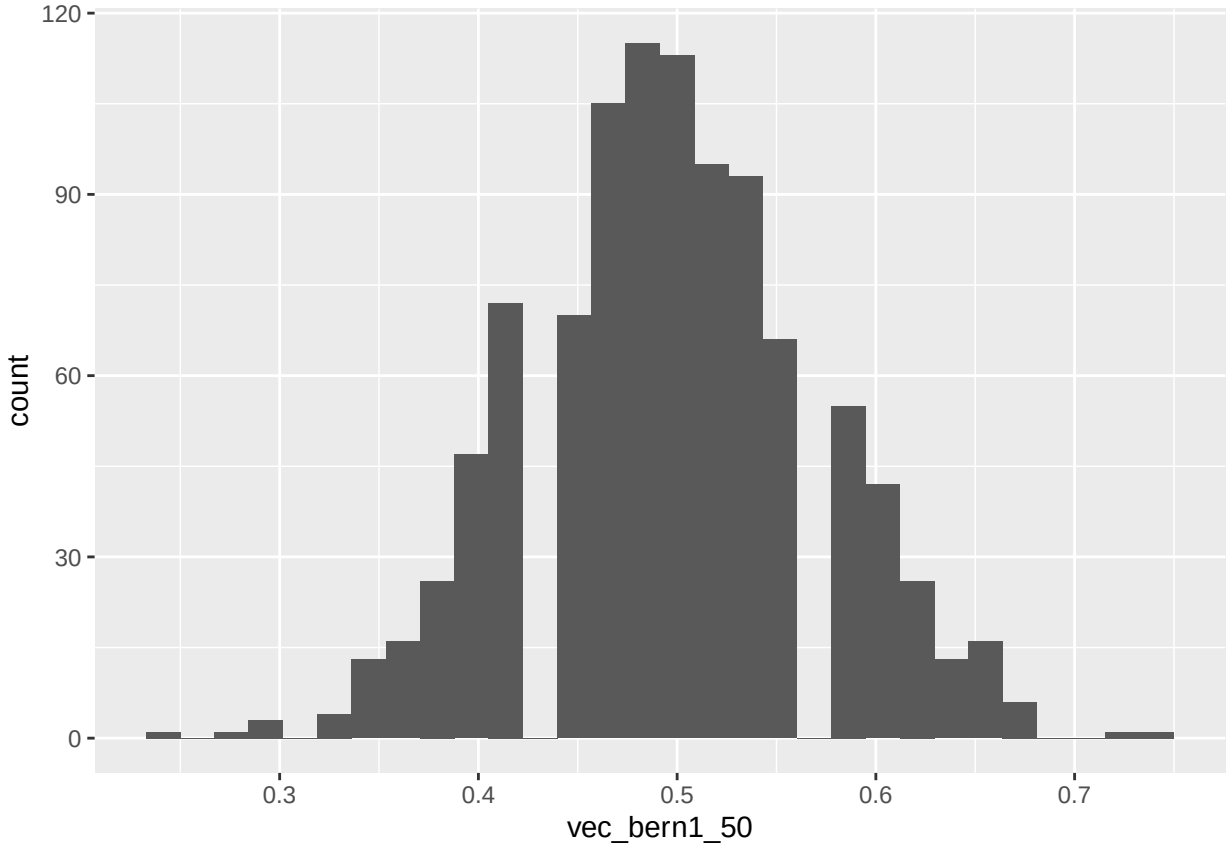



```
qqnorm(vec_bern1_20)
```

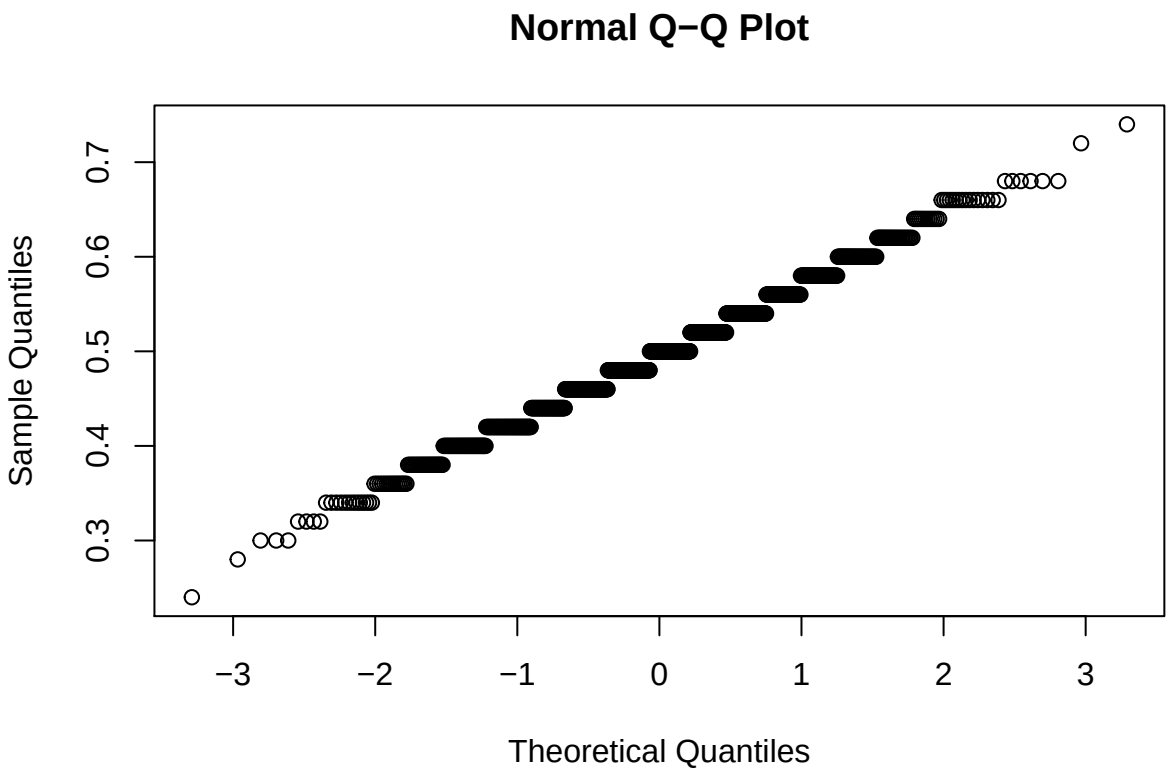



```
#n=50
set.seed(18225012)
vec_bern1_50 <- replicate(reps, mean(sample(c(0, 1), size = 50, replace = TRUE)))
ggplot(data = data.frame(vec_bern1_50), aes(x = vec_bern1_50)) +
  geom_histogram()
```

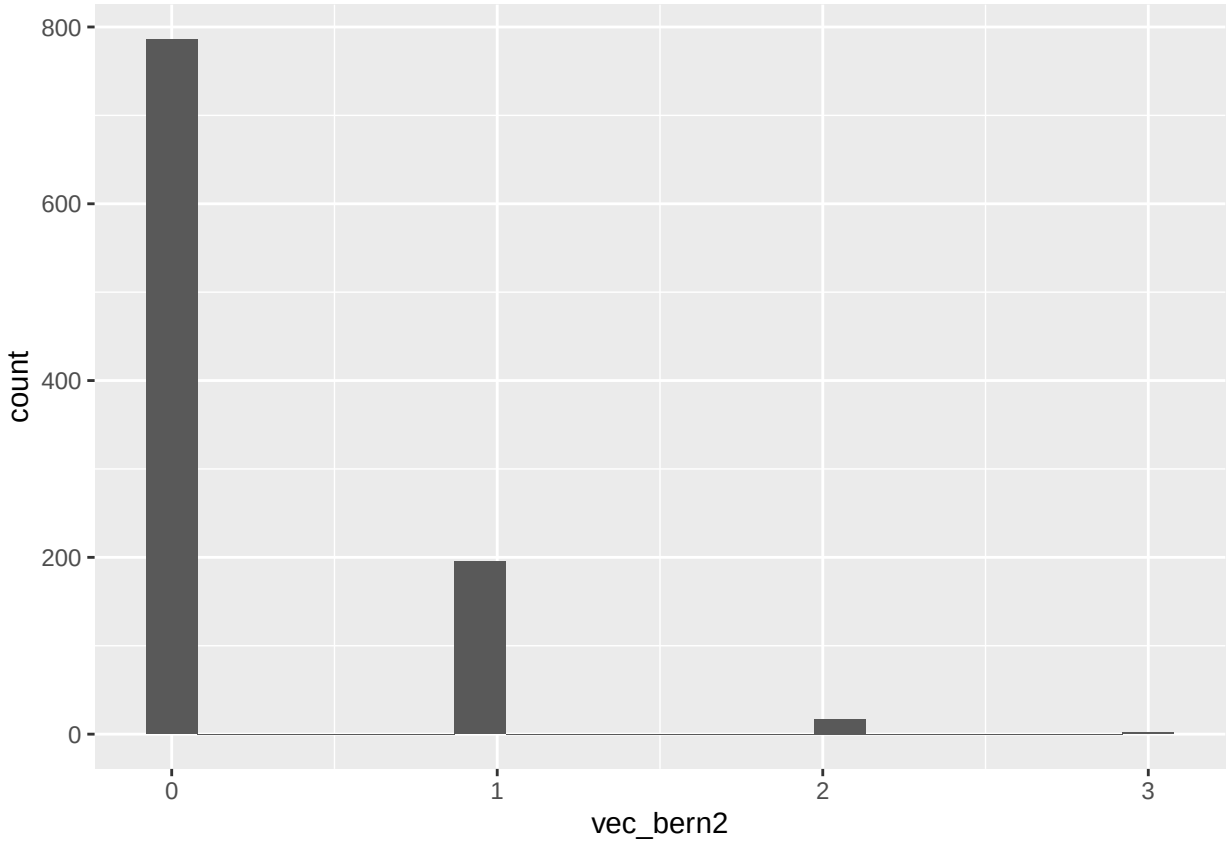
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



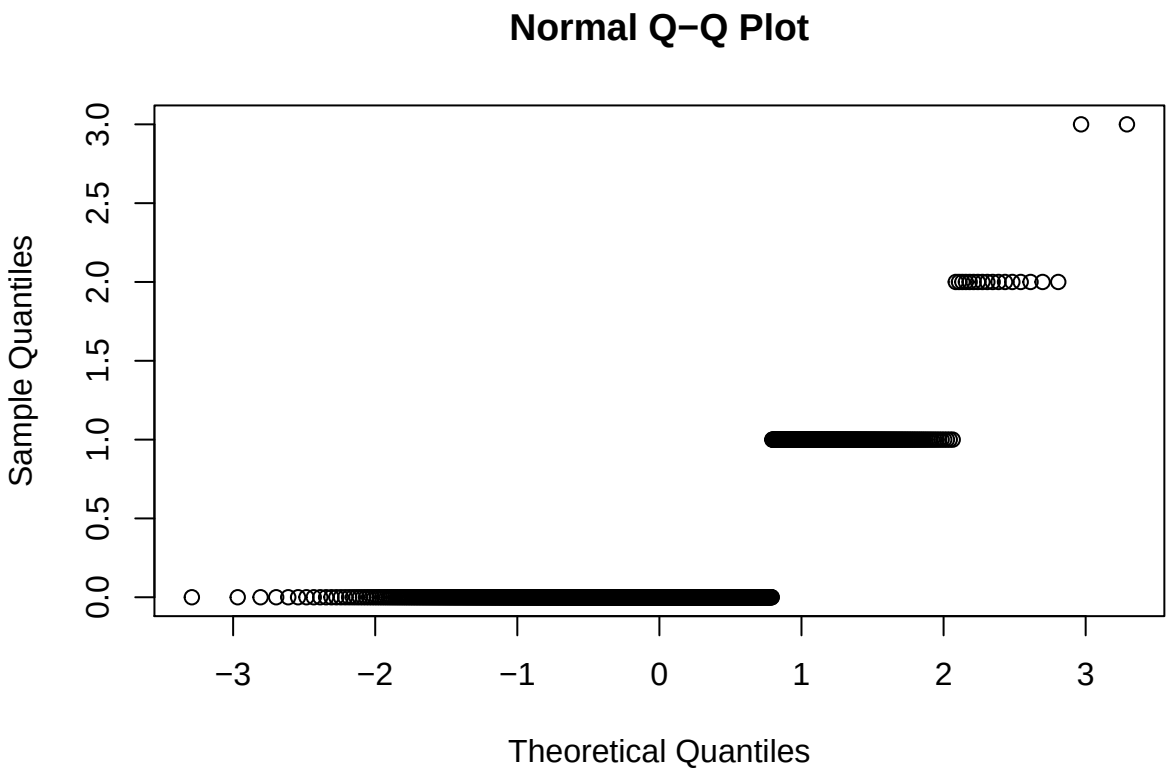

```
qqnorm(vec_bern1_50)
```



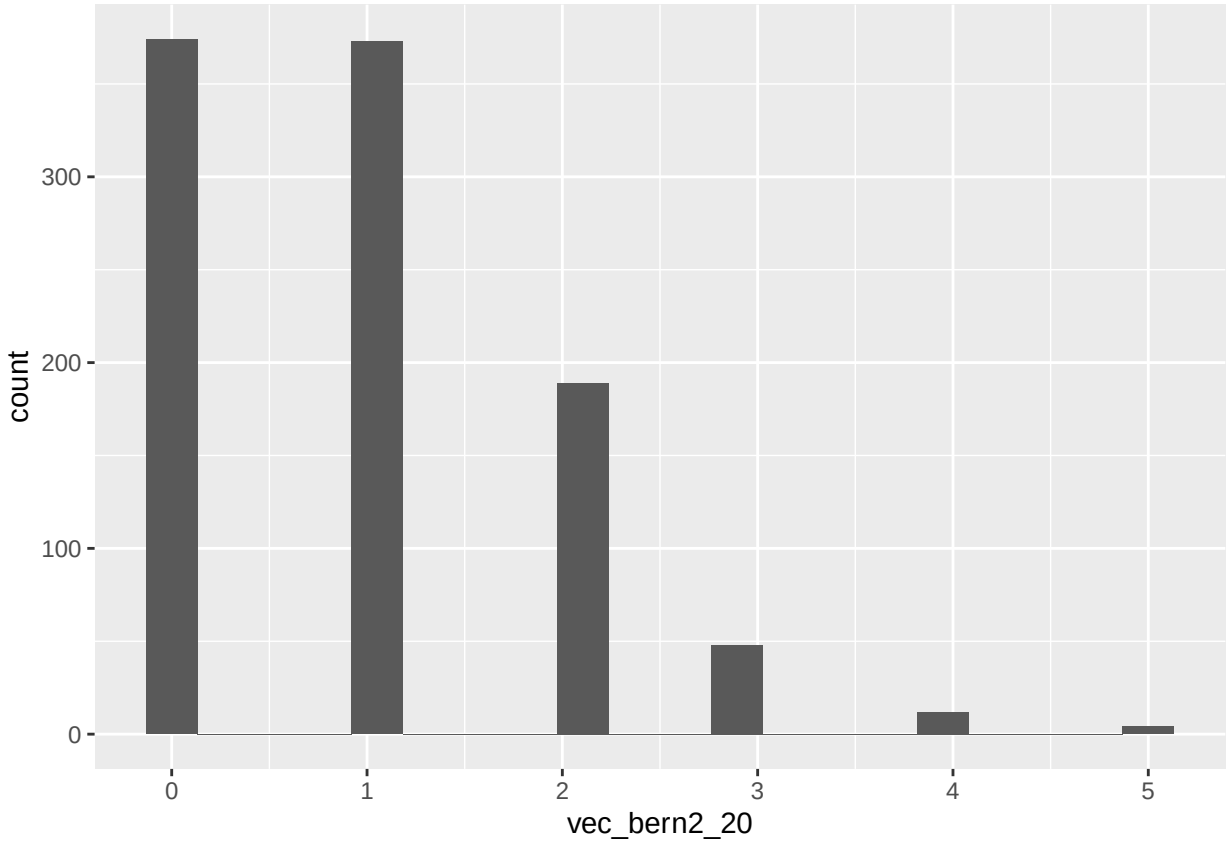

```
#0.05
set.seed(18225012)
vec_bern2 <- replicate(1000, mean(rbinom(1, 5, 0.05)))
ggplot(data = data.frame(vec_bern2), aes(x=vec_bern2)) +
  geom_histogram(bins = 20)
```



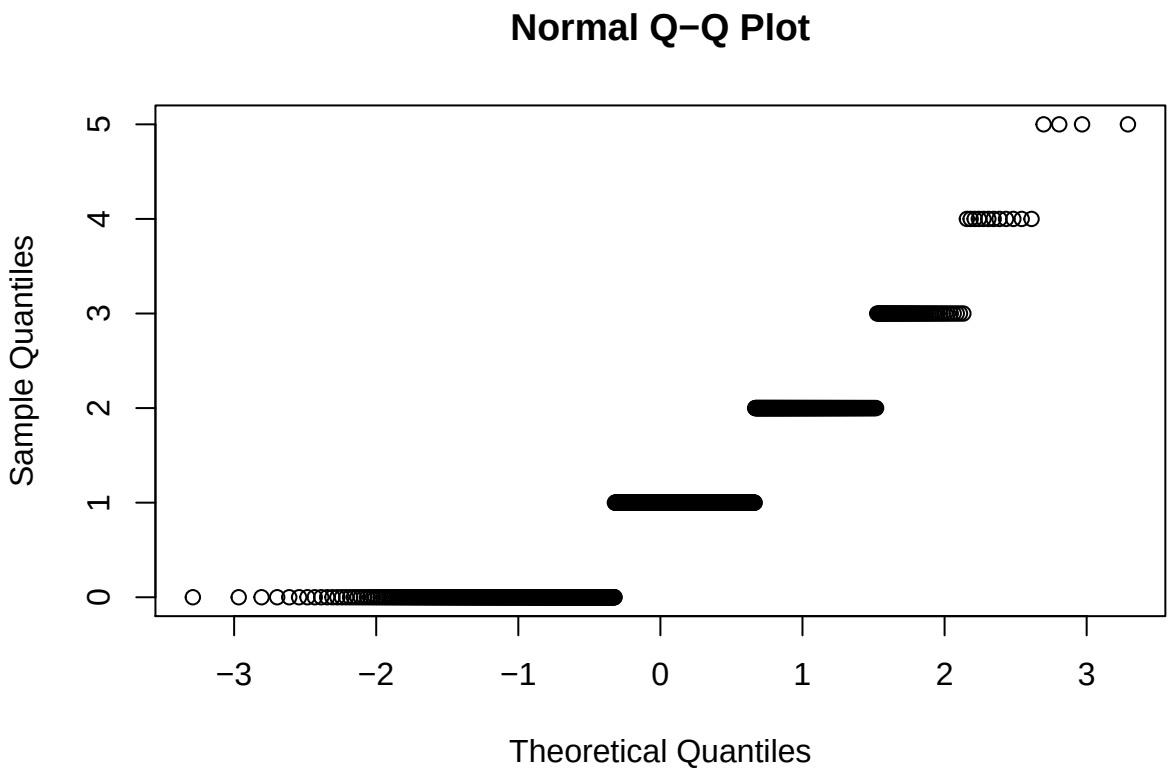

```
qqnorm(vec_bern2)
```



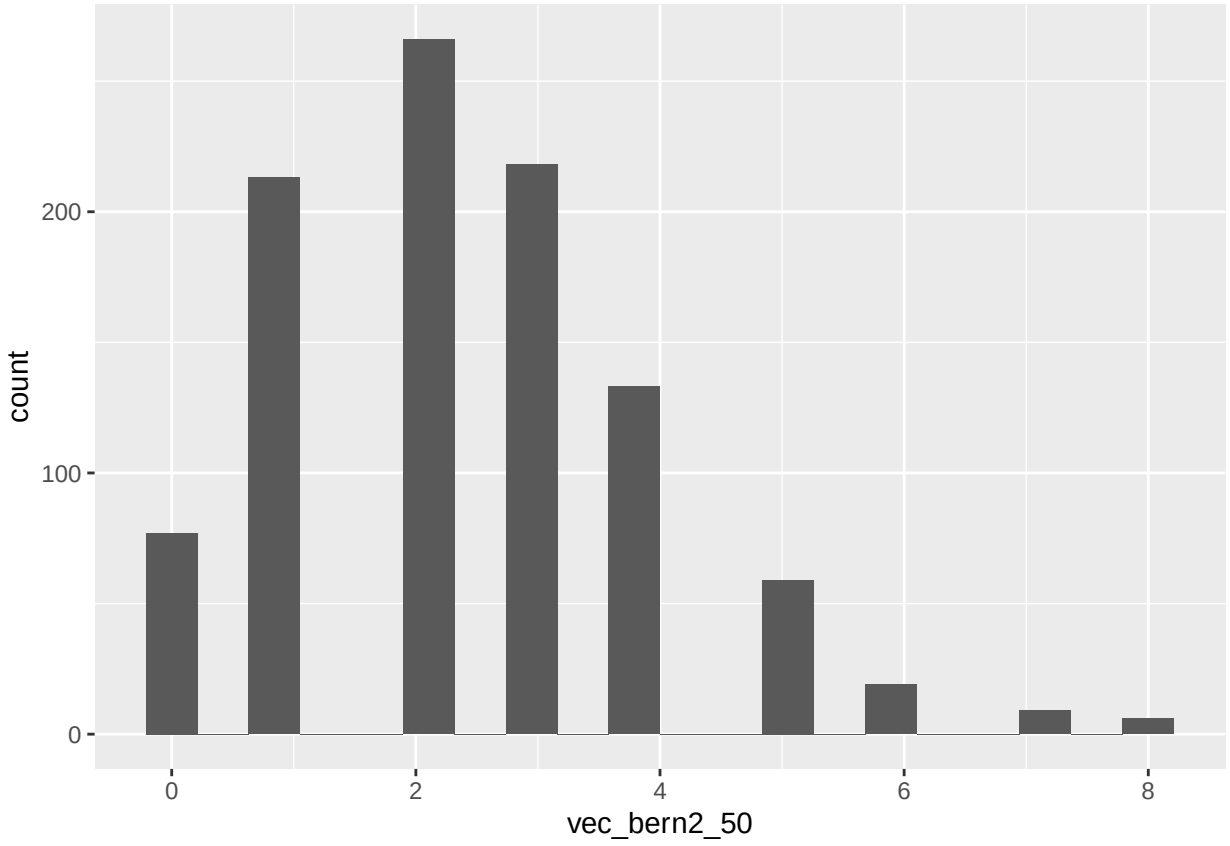

```
#n=20
set.seed(18225012)
vec_bern2_20 <- replicate(1000, mean(rbinom(1, 20, 0.05)))
ggplot(data = data.frame(vec_bern2_20), aes(x=vec_bern2_20)) +
  geom_histogram(bins = 20)
```



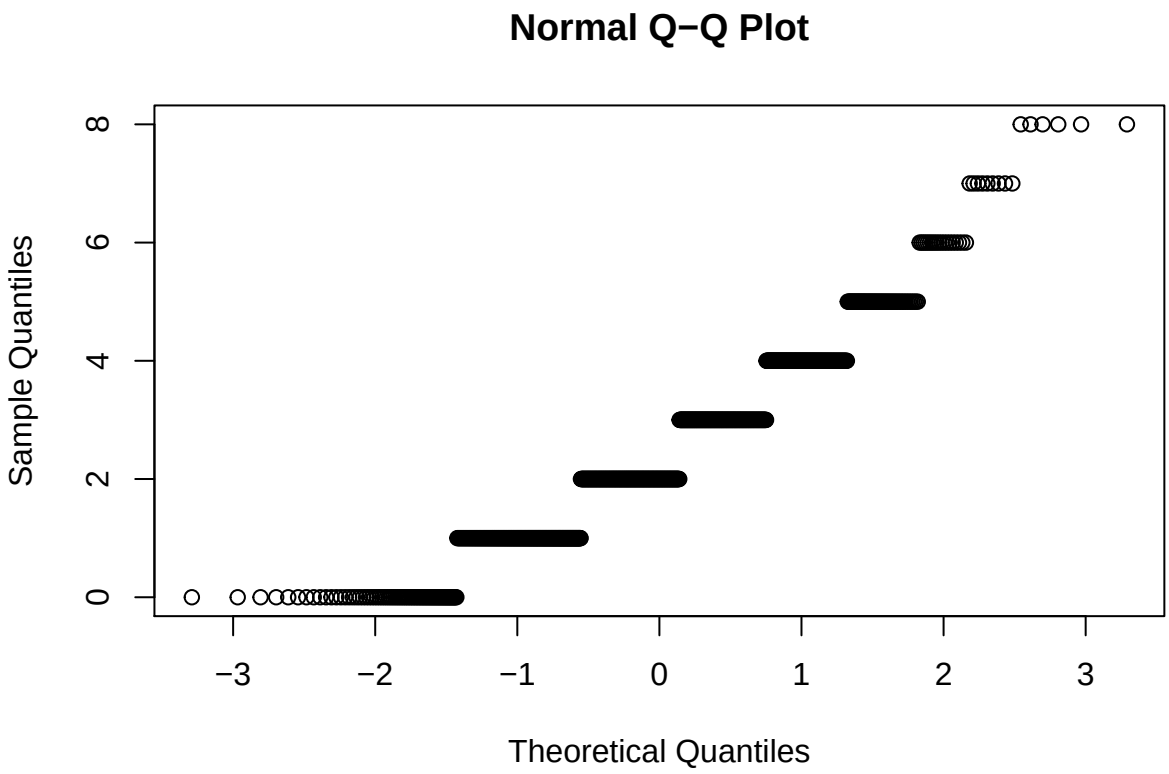

```
qqnorm(vec_bern2_20)
```




```r
#n=50
set.seed(18225012)
vec_bern2_50 <- replicate(1000, mean(rbinom(1, 50, 0.05)))
ggplot(data = data.frame(vec_bern2_50), aes(x=vec_bern2_50)) +
  geom_histogram(bins = 20)
```




```
qqnorm(vec_bern2_50)
```



As can also be seen with both cases of bernoulli data, the larger the sample size, the closer the sample mean gets to a normal distribution. n=5 for all 3 cases appears to not be normally distributed.

Q3 - Gamma distribution maximum likelihood estimation

i

```
#i sample of n=100, gamma data lambda=1, alpha=3
set.seed(18225012)
sample_gam <- rgamma(100, 3, 1)
```

ii

$l(\alpha, \lambda) = \sum log \frac{\lambda^\alpha x^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)}$

$= n\alpha log\lambda - \lambda \sum x_i + (\alpha - 1) \sum log x_i - n log\Gamma(\alpha)$

$\frac{\partial}{\partial \lambda} = \frac{n\alpha}{\lambda} - \sum x_i = 0$

$\frac{n\alpha}{\lambda} = \sum x_i$

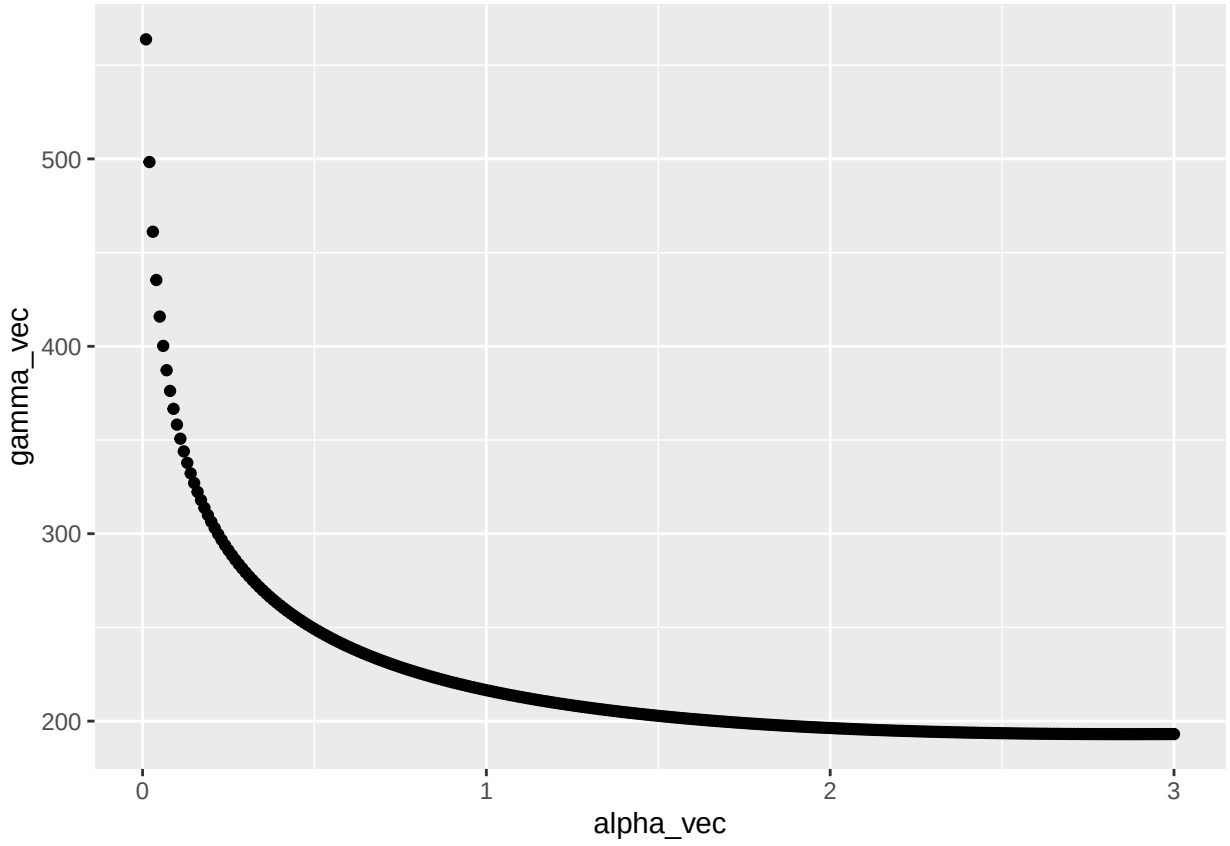
$\lambda = \frac{n\alpha}{\sum x_i}$

$\hat{\lambda} = \frac{n\hat{\alpha}}{\sum x_i} = \frac{\hat{\alpha}}{\bar{x}}$

The initial formula used here is derived from the joint pdf for the gamma distribution.

iii

```
gamma_likelihood_alph <- function(gamma_vec, alpha) {

  xbar <- mean(gamma_vec)
  n <- length(gamma_vec)
  log_xi <- log(gamma_vec)

  est <- (alpha*n*log(alpha/xbar)) - n*log(gamma(alpha)) + (alpha - 1)*(sum(log_xi)) -
    (alpha/xbar)*sum(gamma_vec)
  return(-est)
}



alpha_vec <- vector()
gamma_vec <- vector()



for (alpha in seq(0.01,3,0.01)) {
  obvs <- gamma_likelihood_alph(sample_gam, alpha)
  alpha_vec <- c(alpha_vec, alpha)
  gamma_vec <- c(gamma_vec, obvs)

}
df.3 <- data.frame(alpha_vec, gamma_vec)
ggplot(data = df.3, aes(x = alpha_vec, y = gamma_vec)) + geom_point()
```




```
i <- min(gamma_vec)
for (count in seq(1,300))
  if (df.3[count,2]==i)
    index <- count
alpha_hat <- df.3[index,1]
lamda_hat <- alpha_hat/mean(sample_gam)
print(alpha_hat)
```

```
## [1] 2.87
```

```
print(lamda_hat)
```

```
## [1] 0.8952869
```

From looking at our graph, these are the most likely values for lambda and alpha, which uses the data from the sample to calculate this.

Q4 - Evaluating Confidence Intervals

i

```r
nconf_95 <- function(n, mu, s2) #mu=mean, #s2=variance #1000 random samples?
{data_norm <- rnorm(n, mu, sqrt(s2))
sample_mean <- mean(data_norm)
mean_95_interval <- vector(mode = "numeric", length = 2)
mean_95_interval[1] <-  sample_mean - 1.96*sqrt(s2)*n^-0.5
mean_95_interval[2] <-  sample_mean + 1.96*sqrt(s2)*n^-0.5
return(mean_95_interval)
}
#test with n=1000
nconf_95(1000,0,1)
```

```
## [1] -0.08461525  0.03934603
```

ii

```r
t_nconf_95 <- function(n, mu, s2) #mu=mean, #s2=variance #1000 random samples?
{data_norm <- rnorm(n, mu, sqrt(s2))
sample_mean <- mean(data_norm)
t_sample <- qt(.975, n-1, lower.tail = TRUE, log.p = FALSE)
t_mean_95_interval <- vector(mode = "numeric", length = 2)
t_mean_95_interval[1] <-  sample_mean - t_sample*sqrt(s2)*n^-0.5
t_mean_95_interval[2] <-  sample_mean + t_sample*sqrt(s2)*n^-0.5
return(t_mean_95_interval)
}
```

iii

t distribution is better for smaller samples therefore function t_nconf_95 for smaller samples.

iv

```r
reps <- 1000
pop_mean <- mean(replicate(reps, rnorm(10,1,1)))
set.seed(18225012)
rep_nsample4 <- replicate(reps, nconf_95(10,1,1))#population
prop_count <- 0
for (i in 1:1000)
{
  if  ((pop_mean > rep_nsample4[1,i]) & (pop_mean < rep_nsample4[2,i]))
  {prop_count <- prop_count+1}
}
print(prop_count/1000)
```

```
## [1] 0.94
```

```r
set.seed(18225012)
rep_ntsample4 <- replicate(reps, t_nconf_95(10,1,1))
prop_tcount <- 0
for (i in 1:1000)
{
  if  ((pop_mean > rep_ntsample4[1,i]) & (pop_mean < rep_ntsample4[2,i]))
  {prop_tcount <- prop_tcount+1}
}
print(prop_tcount/1000)
```

```
## [1] 0.975
```

v

```r
pop_mean <- mean(replicate(reps, rnorm(500,1,1)))
set.seed(18225012)
rep_nsample4 <- replicate(reps, nconf_95(500,1,1))#population
prop_count <- 0
for (i in 1:1000)
{
  if  ((pop_mean > rep_nsample4[1,i]) & (pop_mean < rep_nsample4[2,i]))
  {prop_count <- prop_count+1}
}
print(prop_count/1000)
```

```
## [1] 0.961
```

```r
set.seed(18225012)
rep_ntsample4 <- replicate(reps, t_nconf_95(500,1,1))
prop_tcount <- 0
for (i in 1:1000)
{
  if  ((pop_mean > rep_ntsample4[1,i]) & (pop_mean < rep_ntsample4[2,i]))
  {prop_tcount <- prop_tcount+1}
}
print(prop_tcount/1000)
```

```
## [1] 0.963
```

Results here are more accurate, as a result of the sample size being larger.

Q5 - Bootstrapping

i

Bootstrapping is a method used in statistics which takes a single data set and resamples with replacement to create a larger simulated data set which allows you to measure many estimators and perform different tests. The process involves taking random samples from the data and using these as to calculate sample means, medians or confidence intervals.The larger the amount of samples produced, the more accurate this method becomes, as your data should approximate the true population data.

ii

```
set.seed(18225012)
sample_exp <- rexp(100,1)
set.seed(18225012)
bootstrap <- lapply(1:1000, function(i) sample(sample_exp, replace = T))
bootMedian <- sapply(bootstrap, median)
boot_CI <- quantile(bootMedian, c(0.025, 0.975))
print(boot_CI)
```

```
##      2.5%     97.5%
## 0.4652868 0.9186848
```