Lab 4: Functions and Quantile Plots

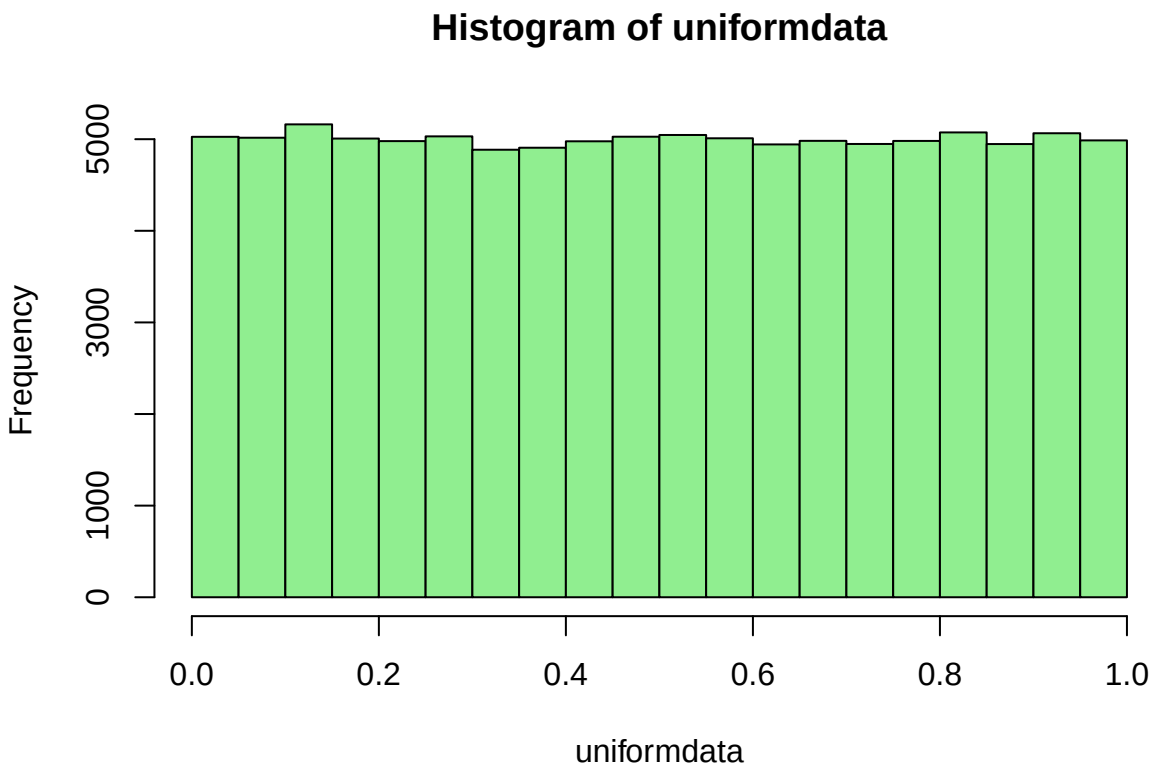
Complete all of the following questions, adding your inputs as code chunks (enclose within triple accent marks) within Rmarkdown.

The exercises are not marked and will not be factored into your course grade, but it is important to complete them to make sure you have the skills to answer assessment questions. You may consult any resource, including other students and the instructor. Please Knit this document to a PDF and upload your work via Canvas at the end of the session. Solutions will be posted for you to check your own answers.

Functions

1. Further test the skewness function from the instructor demo (or your own version if you prefer) by calculating the skewness of a large number of measurements randomly chosen from two additional symmetric distributions: the uniform distribution, and a t-distribution with 5 degrees of freedom. (Sample a uniform distribution with `runif()` and a t-distribution with `rt()`.)

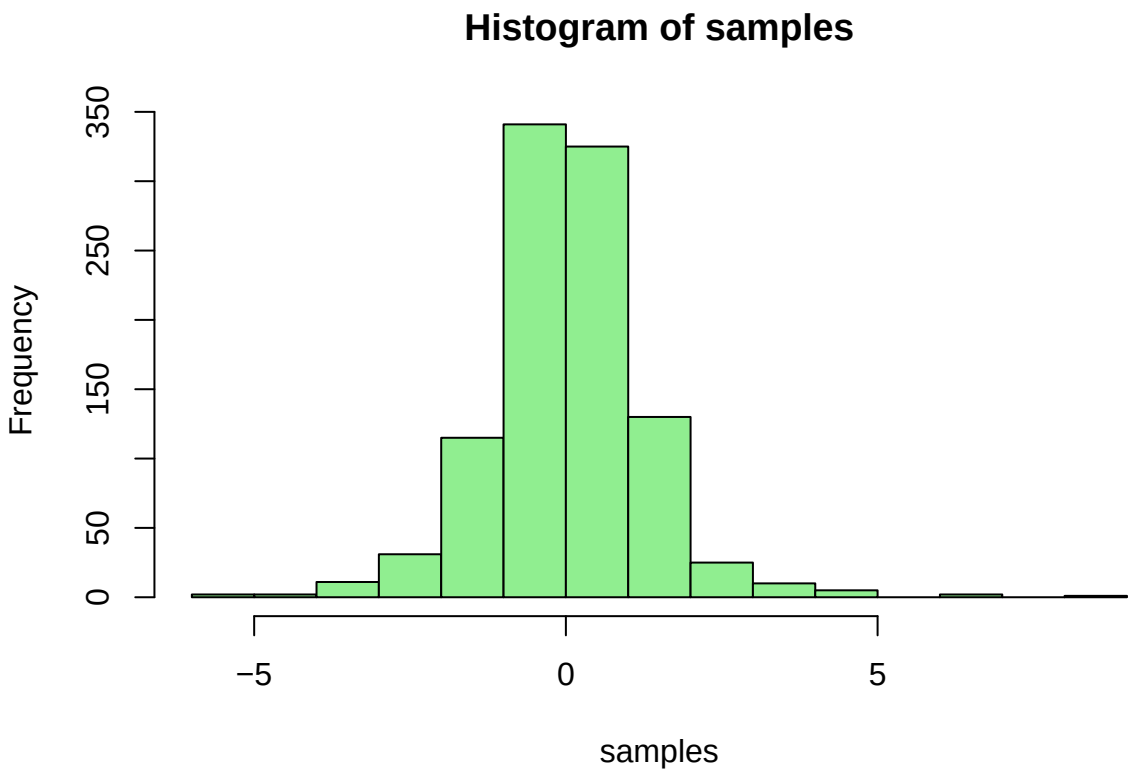
```
skewness = function(x) { sum((x - mean(x))^3)/(length(x)*sd(x)^3) }

uniformdata = runif(100000)
hist(uniformdata,col='lightgreen')
```




```
skewness(uniformdata)
```

```
## [1] 0.001269824
```

```
skewness = function(x) { sum((x - mean(x))^3)/(length(x)*sd(x)^3) }

samples <- rt(1000, 5)
hist(samples,col='lightgreen')
```




```
skewness(samples)
```

```
## [1] 0.4058092
```

2. Write a custom R function to calculate excess kurtosis of a sample. Confirm (using a simulation) that the excess kurtosis of a normal distribution is zero. (Hint: pay special attention to the positioning of brackets, and don't forget to subtract three.)

```
x <- rnorm(10000)

kurtosis= function(x){(sum((x-mean(x))^4)/length(x)*sd(x)^4)-3}

kurtosis(x)
```

```
## [1] 0.1029576
```

```
kurtosis(rnorm(100000,mean=100))
```

```
## [1] -0.009042183
```

3. Estimate (numerically, by drawing random variates) the excess kurtosis for a uniform distribution, and demonstrate that it is -1.2 (as claimed in lecture).

```
x <- rnorm(100000)

kurtosis= function(x){(sum((x-mean(x))^4)/length(x)*sd(x)^4)-3}

kurtosis(runif(x))
```

```
## [1] -2.999914
```

4. Estimate the excess kurtosis for a t distribution with 5 degrees of freedom. Is this a leptokurtic, mesokurtic, or platykurtic distribution?

```
 k<-kurtosis(rt(100000, df=5))
k
```

```
## [1] 59.80953
```

k poitive value shows eptokurtic distribution.

5. Show with a few examples (or by conducting a simulation) that the standard error on the skewness for normal data is approximately $\sqrt{6/n}$.

```
skewness = function(x) { sum((x - mean(x))^3)/(length(x)*sd(x)^3) }
values=rep(0,1000)
for (i in 1:1000) values[i]=skewness(rnorm(100))
sd(values); sqrt(6/100)
```

```
## [1] 0.231374
```

```
## [1] 0.244949
```

6. Show with a few examples (or by conducting a simulation) that the standard error on the kurtosis for normal data is approximately $\sqrt{24/n}$.

```
values=rep(0,1000)
for (i in 1:1000) values[i]=kurtosis(rnorm(100))
sd(values); sqrt(24/100)
```

```
## [1] 1.98395
```

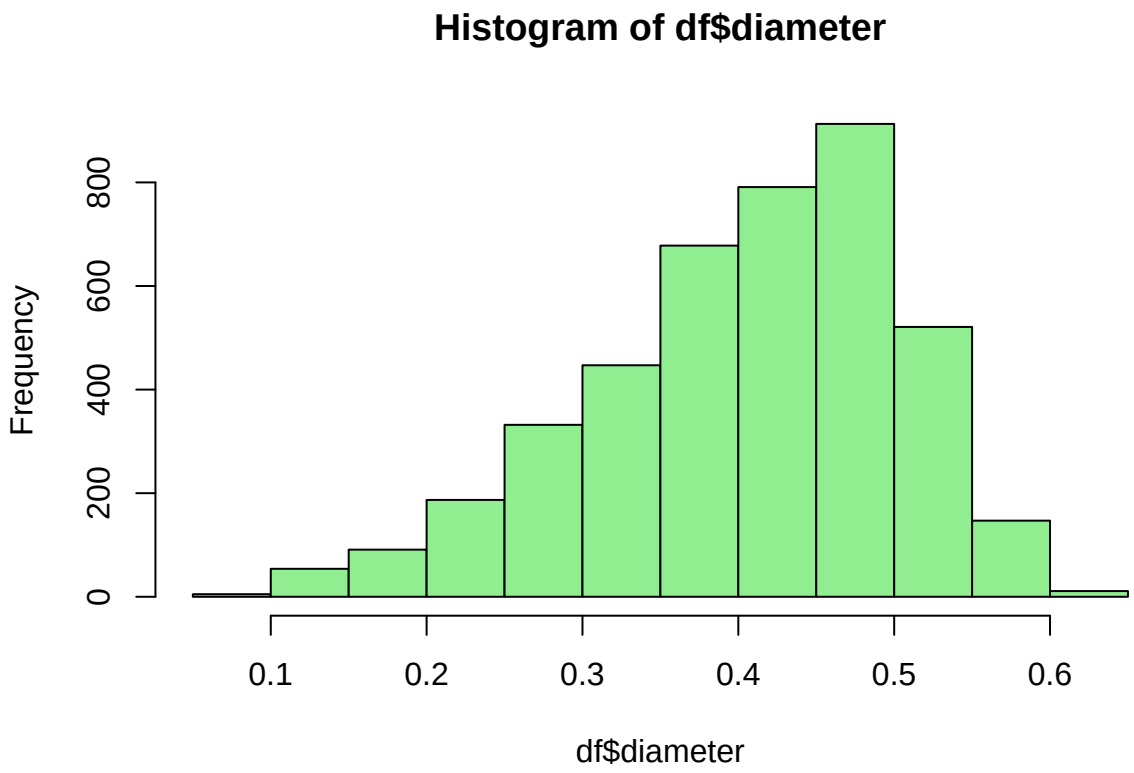
```
## [1] 0.4898979
```

ECDFs, EQFs, and QQ plots

Load in abalone.csv (from Lab 2) to R as a data frame. You may remember that the diameters of the whole sample were definitely not normally-distributed, but the diameters of the adults (>=13 rings) were roughly normally-distributed.

7. Produce a histogram of the diameters for the entire sample similar to the one you made before. Based on this histogram, what type of skewness does this sample exhibit: positive or negative?

```
df=read.csv('abalone.csv')

hist(df$diameter,col='lightgreen')
```



8. Verify your visual estimate above with your skewness function.

```
sk<-skewness(df$diameter)
sk
```

```
## [1] -0.6087607
```

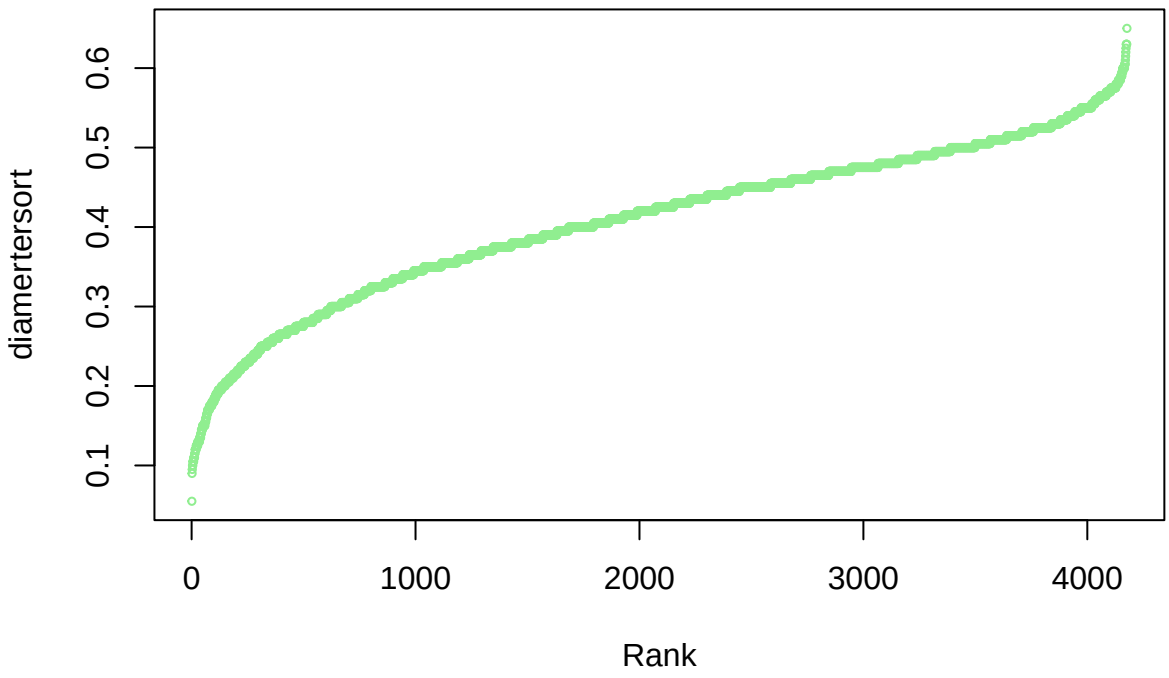
9. Create a vector that contains all the diameters, sorted from smallest to largest.

```
h<-sort(df$diameter)
h[1:10]
```

```
##  [1] 0.055 0.090 0.095 0.100 0.100 0.105 0.105 0.105 0.105 0.110
```

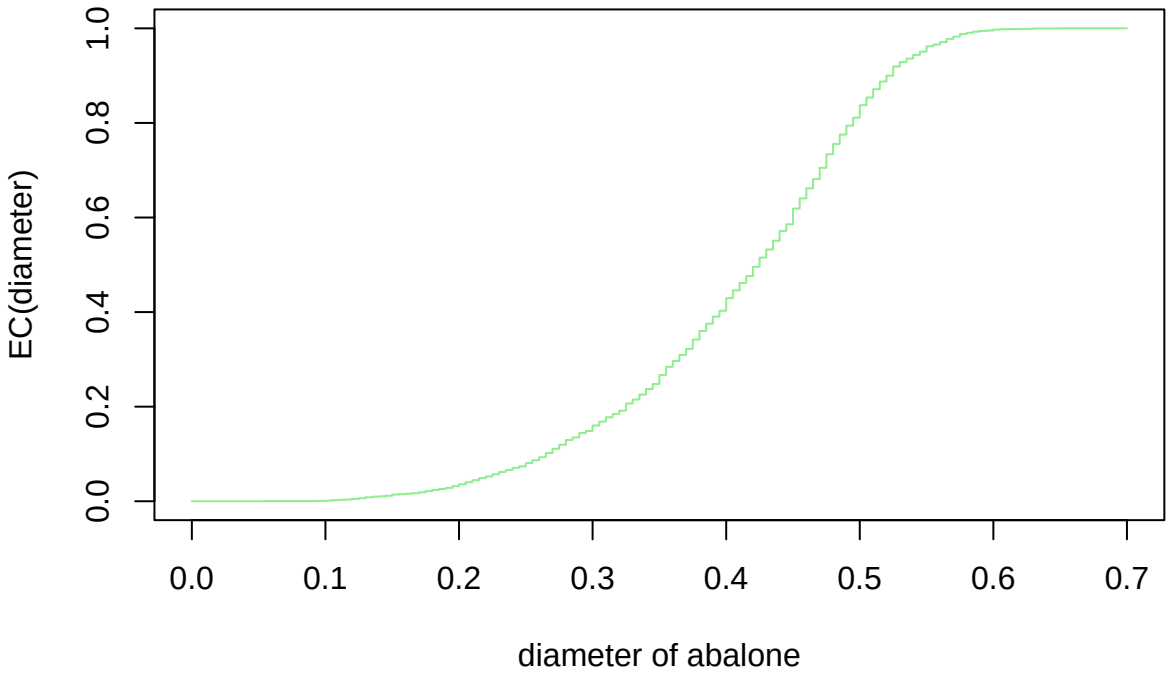
10. Make a simple plot of this sorted vector: index(/rank) versus value.

```
plot(h,cex=0.50, xlab='Rank', ylab='diamertersort', col='lightgreen')
```



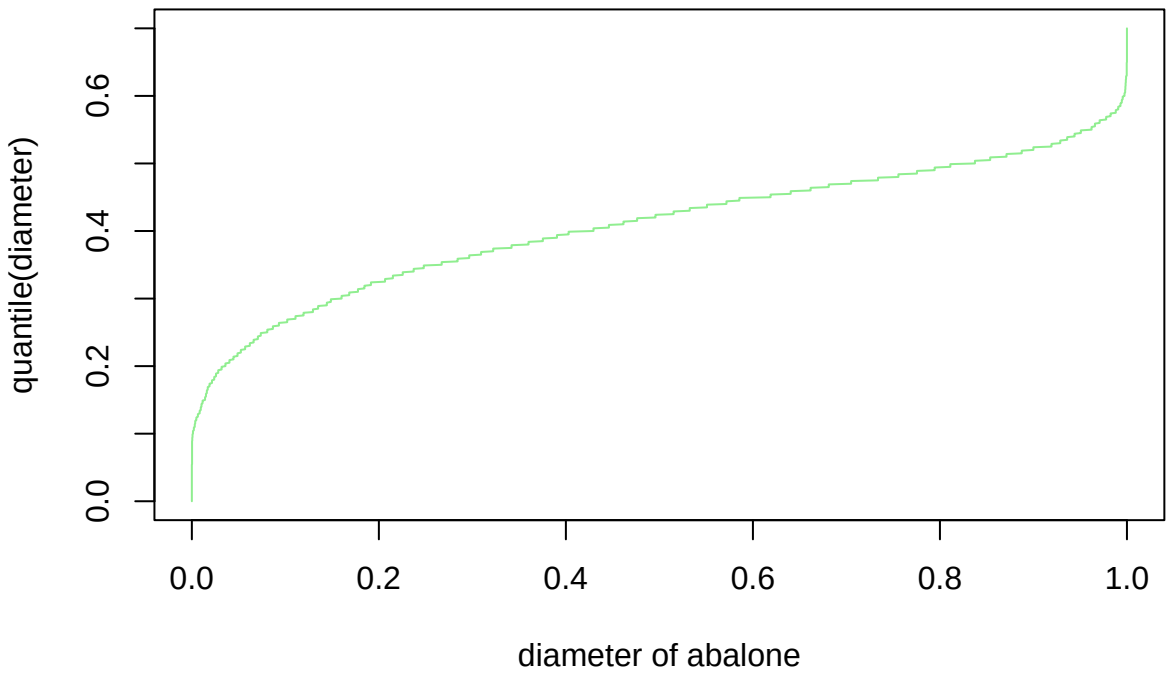
11. Make a step plot of the ECDF for the abalone diameters. Label the axes descriptively using words to explain what this means.

```
e = ecdf(df$diameter)
d=seq(0,0.7,0.001)
plot(d, e(d), xlab= 'diameter of abalone',
     ylab= 'EC(diameter)', col='lightgreen', type='s')
```



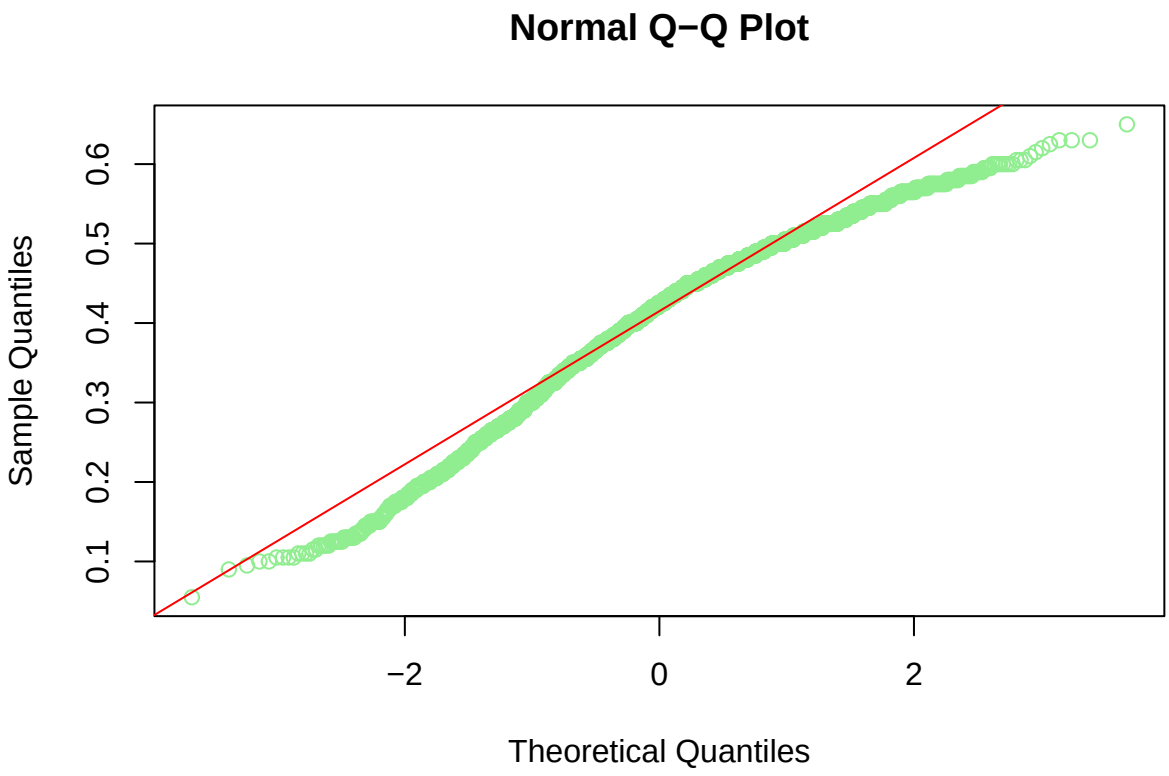
12. Make a plot of the EQF for the abalone diameters, either by reversing the axes for #11 or by using `quantile()`. Label the axes descriptively using words to explain what this means.

```
e = ecdf(df$diameter)
d=seq(0,0.7,0.001)
plot(e(d),d, xlab= 'diameter of abalone',
     ylab= 'quantile(diameter)', col='lightgreen', type='l')
```

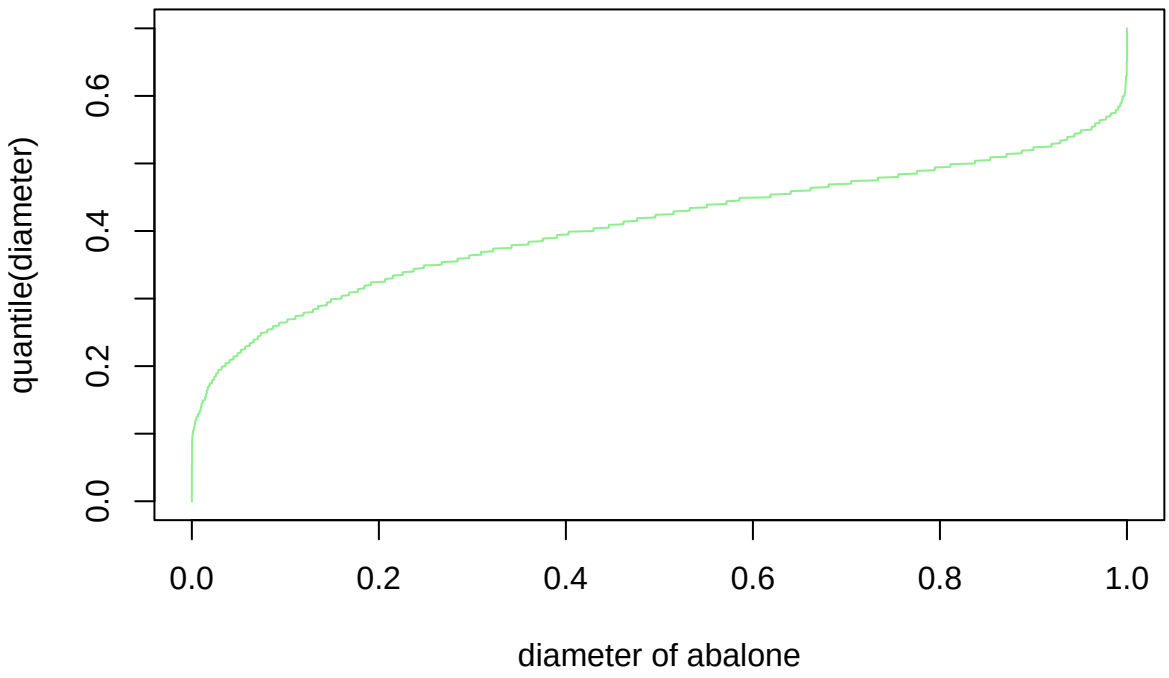


13. Make a normal QQ plot for the abalone diameters. Overplot a theoretical line for normal data. Does this match the type of curve you expect for a skewed distribution?

```
qqnorm(h, col='lightgreen')
qqline(h, col='red')
```




```
plot(e(d),d, xlab= 'diameter of abalone',
     ylab= 'quantile(diameter)', col='lightgreen', type='l')
```



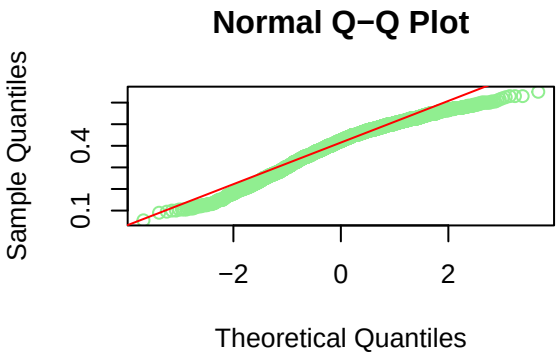
14. Perform a Shapiro-Wilk normality test and quote (and explain) the resulting p value.

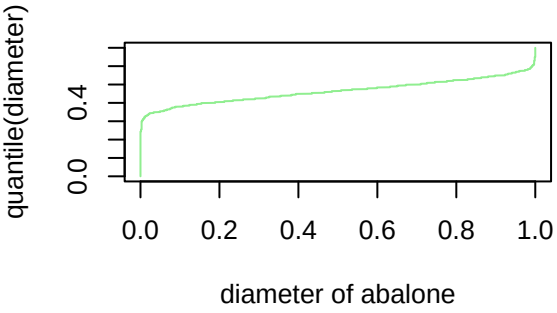
```
shapiro.test(h)
```

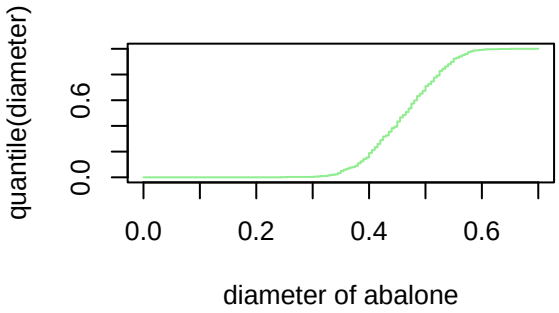
```
##
##  Shapiro-Wilk normality test
##
## data:  h
## W = 0.97048, p-value < 2.2e-16
```

15. Repeat steps 9-14 for the diameters of adult (rings >= 13) abalone.

```
adultdia=sort(df$diameter[df$rings>=13])

par(mfrow=c(2,2))
qqnorm(h, col='lightgreen')
qqline(h, col='red')
plot(ecdf(adultdia)(d),d, xlab= 'diameter of abalone',
     ylab= 'quantile(diameter)', col='lightgreen', type='l')
plot(d,ecdf(adultdia)(d), xlab= 'diameter of abalone',
     ylab= 'quantile(diameter)', col='lightgreen', type='s')
```








```
shapiro.test(adultdia)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  adultdia
## W = 0.99403, p-value = 0.007739
```

```
par(mfrow=c(1,1))
```

Array drops

16. Suppose you have six measurements: 101, 100, 98, 103, 0, and 80. Store these values in a vector (named u), initially in this order. Then sort these values from smallest to largest and store the result in a new vector (named sortu).

```
u=c(101, 100, 98, 103, 0, 80)
sortu=sort(u)
sortu
```

```
## [1]   0  80  98 100 101 103
```

17. Use negative subscripts on sortu to create a new vector with the highest and lowest values dropped. (Be sure to use negative subscripts, not another method). Print out this vector and its mean.

```
drop=sortu[c(-1,-6)]
drop; mean(drop)
```

```
## [1]  80  98 100 101
```

```
## [1] 94.75
```

18. Write a function `modifiedmean(v)` that calculates the so-called *modified mean* of a vector. The modified mean is a trimmed mean in which the single highest and single lowest values are excluded. The function should accept one argument ("v") and do the following:

    a. Sort the vector given to the function ("v") and store the sorted vector in a new variable.
    b. Remove the first and last elements of the new vector using negative subscripts, and store the trimmed vector in a new variable.
    c. Calculate the mean of the trimmed vector, and return this value to the user.
    (Note: You can also do all this on one line without defining new variables, if you prefer.)
    Then, test it out by calculating the modified mean of u.

```
modifiedmean= function(v){
  x=sort(v)

  y=x[c(-1,-length(x))]
  mean(y)
  return(y)
}
```

Jackknife and bootstrap

19. Write a function `jackknifemeans(v)` to calculate the jackknife means of a vector (suppose the vector is called "v"). Do this as follows:

    a. Calculate the length of the vector and store it as the variable `n`.
    b. Create a variable jmeans, which will store the jackknife means. It should have length n and be all zeros to start with.
    c. Then loop n times. For each loop iteration, set jmeans[i] to be equal to the mean of all the elements of v except for the i-th element, which is dropped.
    d. Return the final vector.

```
jackknifemeans= function(v){
  n=length(v)
  jmean= rep(0, n)
  for (i in 1:n)  jmean[i]=mean(v[-i])
return(v)
}
```

20. Calculate `jackknifemeans(u)` and store it in a new vector. Print out the vector, and calculate its mean and standard deviation. ("u" is the data variable from #7.)

```
jackknifemeans(u)
```

```
## [1] 101 100  98 103   0  80
```

21. Write a function `bootstrapmeans(v, Nboot)` to calculate Nboot bootstrap means of a vector (suppose the vector is called "v"). Do this as follows:
    a. Calculate the length of the vector and store it as the variable "n".
    b. Create a variable bmeans, which will store the bootstrap means. It should have length bootstrap-means and be all zeros to start with.
    c. Then loop Nboot times. For each loop iteration, set bmeans[i] to be equal to the mean of n values resampled (with replacement) from v.
    d. Return (and print) the final vector.

```
bootstrapmeans= function(v,Nboot){
  n=length(v)
  bmeans= rep(0, Nboot)
  for (i in 1:Nboot)  bmeans[i]=mean(sample(v, n, replace=TRUE))
return(bmeans)

}
```

22. Calculate `bootstrapmeans(u, Nboot=1000)` and store it in a new vector. Calculate its mean and standard deviation.

```
z=bootstrapmeans(u, Nboot=1000)
mean(z)
```

```
## [1] 79.34983
```

```
sd(z)
```

```
## [1] 15.93286
```

23. Calculate the mean, standard deviation, and (normal-statistics) standard error on the mean of u. Compare the standard error to the SD of jackknife means and the SD of bootstrap means.

```
mean(u); sd(u)
```

```
## [1] 80.33333
```

```
## [1] 40.23266
```

```
sr= sd(u)/sqrt(length(u))
sr
```

```
## [1] 16.42491
```

Random sampler for a custom function

Suppose a distribution has a non-normal PDF described as follows: it is zero for x<0 and x>A, and it is 2x/A^2 for 0<x<A. A is an unknown parameter. We'll call this the "tilt" distribution. The CDF and QF are:

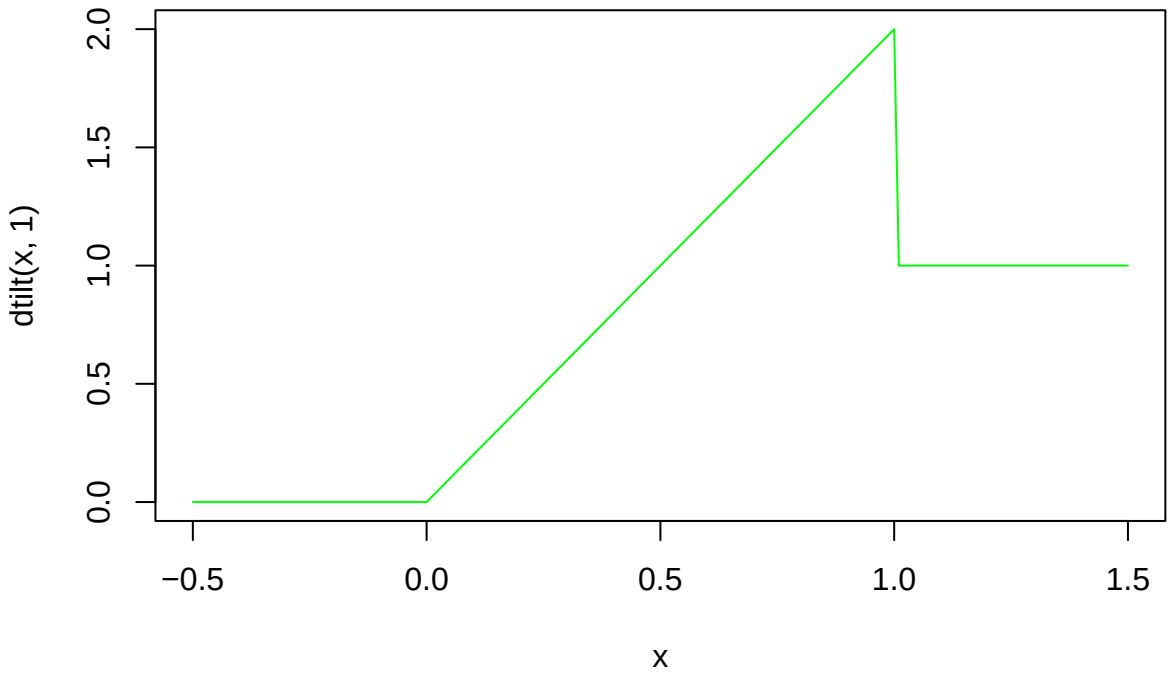
$CDF_{tilt}(x) = x^2/A^2$ for $0 < x < A$. (At $x < 0$ it is 0 and at $x > A$ it is 1.)

$QF_{tilt}(f) = A\sqrt{f}$ (Defined only for f between 0 and 1.)

24. Write down (as an equation - not as code) a mathematical expression for the median of the distribution in terms of A. $A/\sqrt{2}$

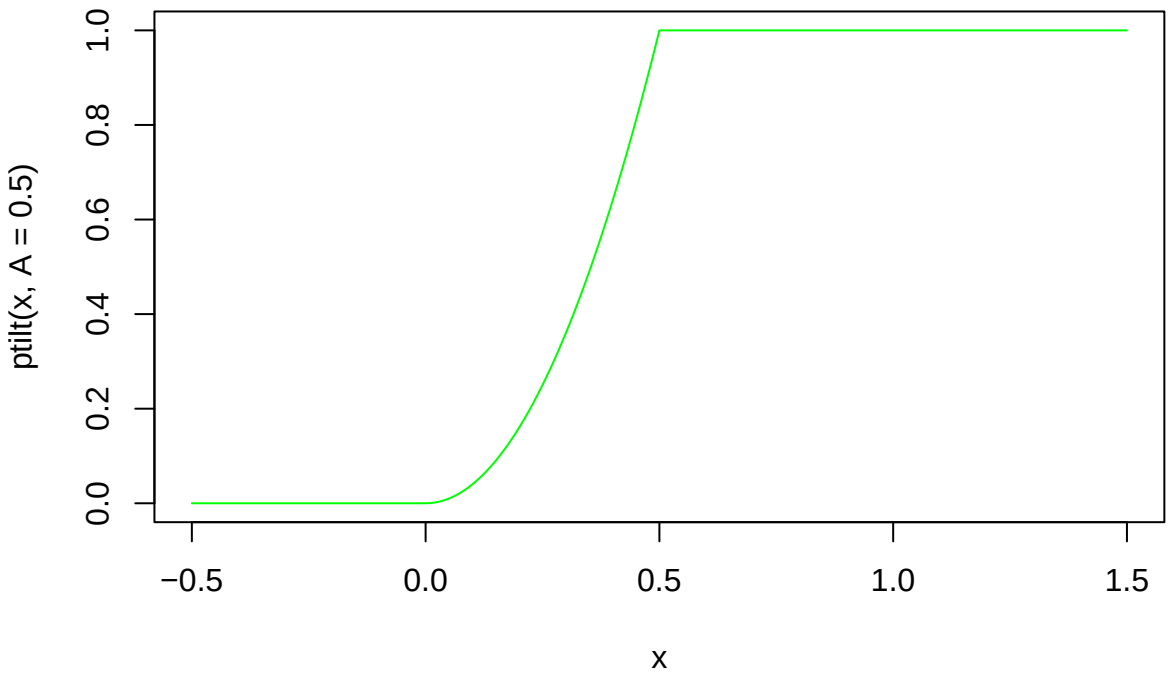
25. Write an R function that evaluates the probability density of the distribution at x. The function should support vector input for x. Call it `dtilt(x,A)`. Plot a test lineplot (for A=1 and x ranging between -0.5 and 1.5).

```
dtilt=function(x,A){
  y=2*x/A^2
  y[x<0]=0
  y[x>A]=1
  return(y)
}

x = seq(-0.5,1.5,0.01)
plot(x,dtilt(x,1),typ='l', col='green')
```



26. Write an R function that evaluates the cumulative distribution function of x. Call it `ptilt(x,A)`. Produce a test line plot of this CDF (for A=+0.5 and x ranging between -0.5 and 1.5).

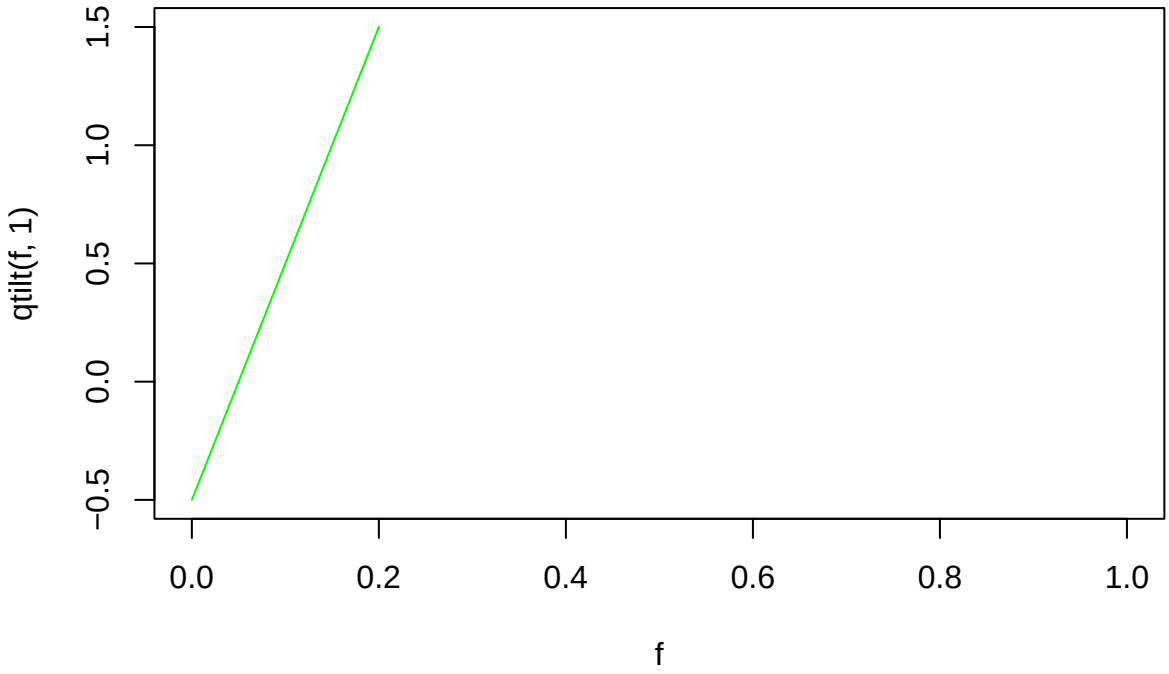
```
ptilt=function(x,A){

  y=x^2/A^2
  y[x<0]=0
  y[x>A]=1
  return(y)
}

x = seq(-0.5,1.5,0.01)
plot(x,ptilt(x,A=0.5),typ='l', col='green')
```



27. Write an R function that evaluates the quantile function to return a quantile value of x. Call it `qtilt(f,A)`. Produce a test line plot of this QF (for A=1 and x ranging between -0.5 and 1.5) below.

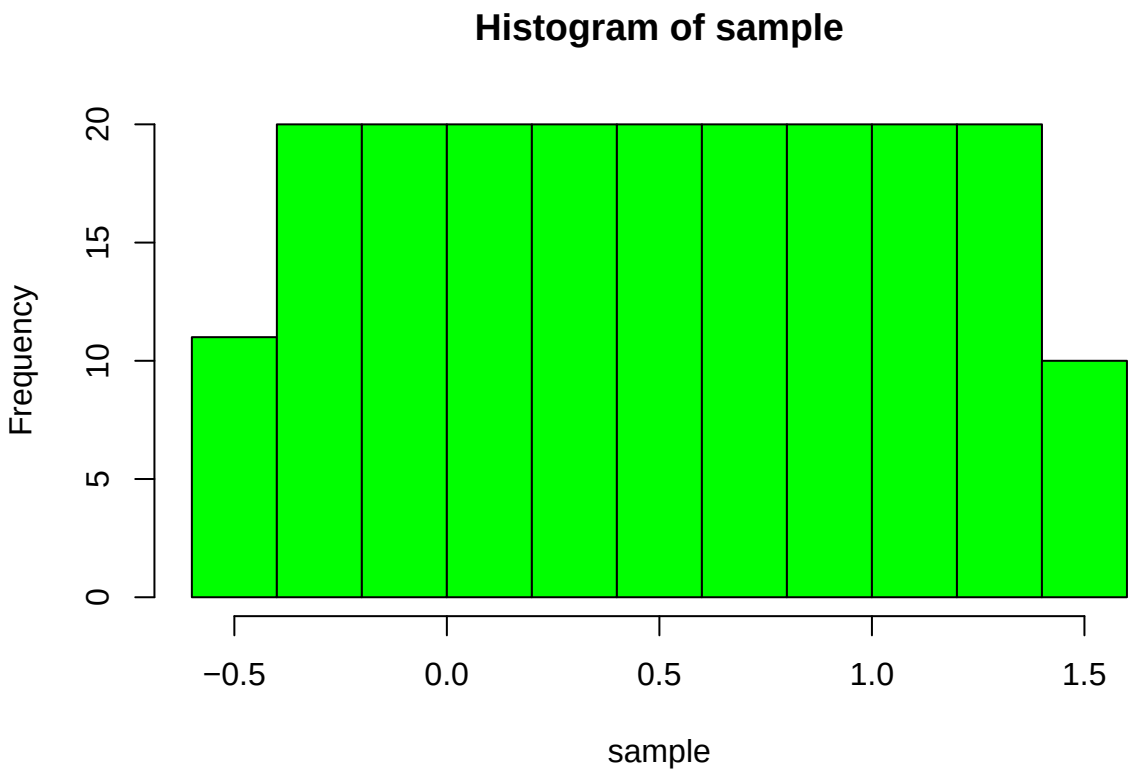
```
qtilt=function(f,A){
    y=A * sqrt(f)
  x[f<0]=NaN
  x[f>1]=NaN
  return(x)
}
```

```
f = seq(0,1,0.001)
plot(f,qtilt(f,1),typ='l', col='green')
```



28. Write an R function that generates n random variates sampled according to the PDF described above—first by sampling the standard uniform distribution, and then passing that random sample through the quantile function you just wrote. Call it `rtilt(n,A)`. Test the function by generating n=10000 random variates for A=1, and plot a histogram.

```
rtilt = function(n,A) {
c = runif(n)
q = qtilt(c,A)
return(q)
}
sample = rtilt(10000,1)
hist(sample, col='green')
```



29. Suppose that the following 5 data points were sampled from a tilt distribution of unknown A: 15.6, 4.5, 17.2, 11.1, 18.7. Calculate their median and use it estimate the value of A by inverting the equation you wrote down in #24.

```
datapoints = c(15.6, 4.5, 17.2, 11.1, 18.7)
md=median(datapoints)

A=md*sqrt(2)
A
```

```
## [1] 22.06173
```

30. Estimate the standard error on the value of A above by repeatedly re-calculating A using five measurements sampled from the `rtilt()` function (give it your above estimate of A each time), storing these values in a vector, and taking the SD of that vector. (Refer back to questions #5 and #6.)

```
recal = numeric(1000)
for (i in 1:1000) recal[i] = median(rtilt(5,A))*sqrt(2)
sd(recal)
```

```
## [1] 0
```