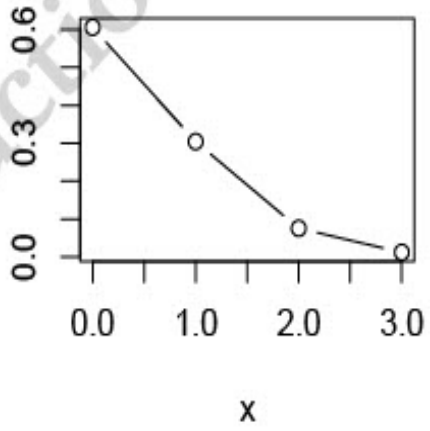
2 Distributions

2.1 Binomial and Poisson Distribution

```
# The format is dbinom(x, n, pi), where x = number of heads out of n tosses of a
# coin, and pi = prob of head. For example,
dbinom(0, 100, 0.005)  # returns the value of the distribution (pmf) itself.

[1] 0.6058

dbinom(0:3, 100, 0.005)  # running dbinom() for multiple values of x in one sweep.

[1] 0.60577 0.30441 0.07572 0.01243

sum(dbinom(0:3, 100, 0.005))  # summing up the above probabilities.

[1] 0.9983
```

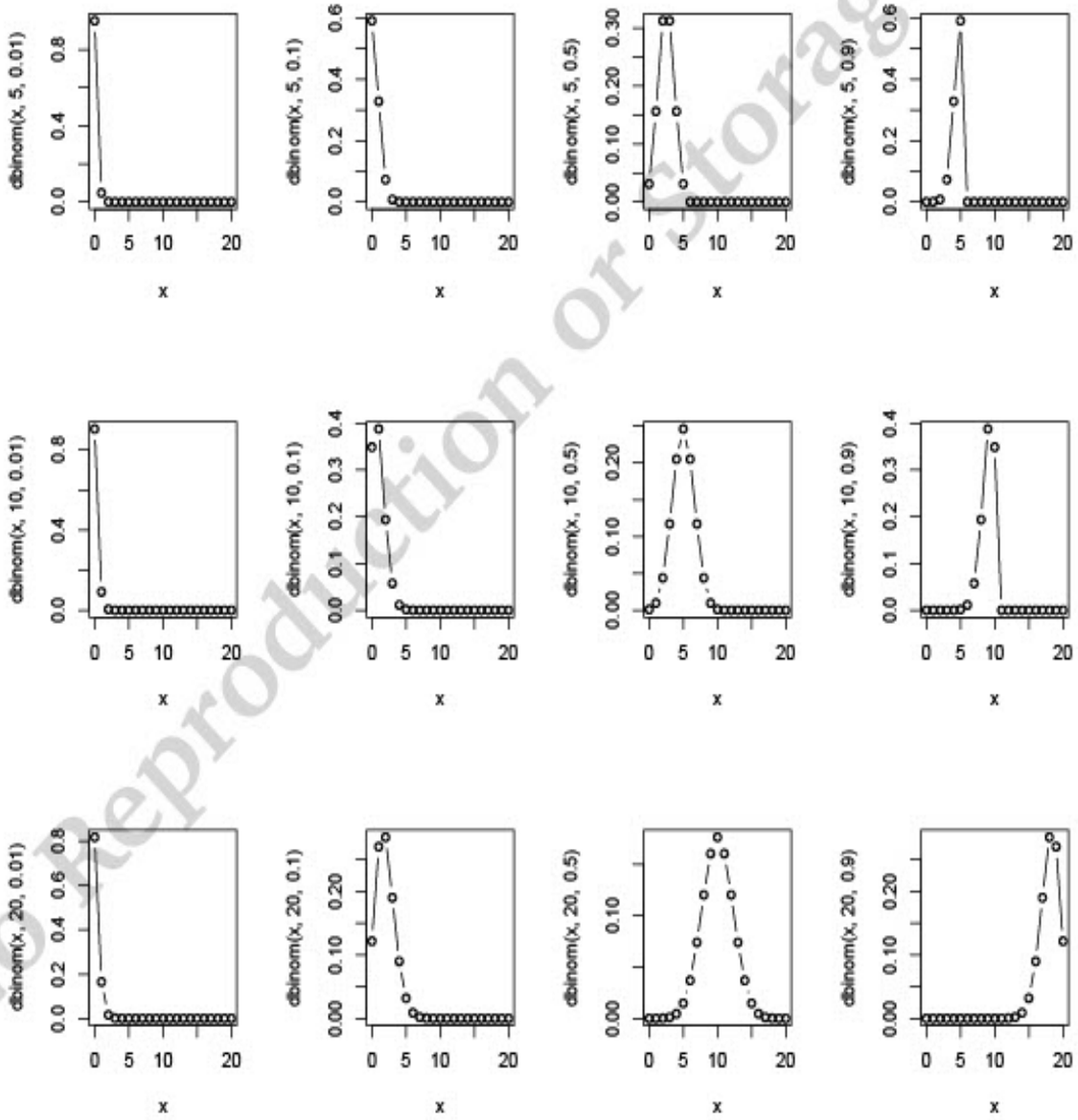
2.1.1 Plotting

```
x <- 0:3
y <- dbinom(0:3, 100, 0.005)
plot(x, y, type = "b")  # "b" (for "both") connects the points with lines.
                        # See ?plot for more options for line types
```




```
# Plotting the mass function for different values of n and pi.
# Note the n and pi values that produce normal-looking distributions,
# and those that produce Poisson-looking distributions.
par(mfrow = c(3, 4))  # A 3 by 4 matrix of figures.
x <- 0:20
plot(x, dbinom(x, 5, 0.01), type = "b")  # n=5, pi=0.01
```

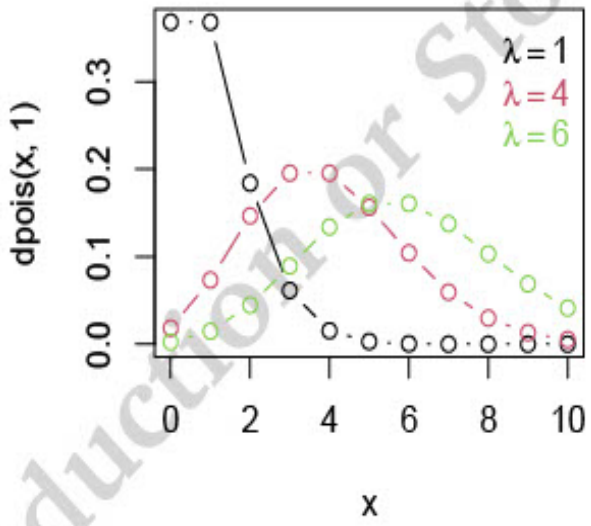
```r
plot(x, dbinom(x, 5, 0.1), type = "b")  # n=5, pi=0.1. Use UP-ARROW to get
                                        # most recent run commend
plot(x, dbinom(x, 5, 0.5), type = "b")  # n=5, pi=0.5
plot(x, dbinom(x, 5, 0.9), type = "b")  # n=5, pi=0.9

plot(x, dbinom(x, 10, 0.01), type = "b")  # n=10, pi=0.01  USE UP-ARROW
plot(x, dbinom(x, 10, 0.1), type = "b")  # n=10, pi=0.1
plot(x, dbinom(x, 10, 0.5), type = "b")  # n=10, pi=0.5
plot(x, dbinom(x, 10, 0.9), type = "b")  # n=10, pi=0.9

plot(x, dbinom(x, 20, 0.01), type = "b")  # n=20, pi=0.01
plot(x, dbinom(x, 20, 0.1), type = "b")  # n=20, pi=0.1
plot(x, dbinom(x, 20, 0.5), type = "b")  # n=20, pi=0.5
plot(x, dbinom(x, 20, 0.9), type = "b")  # n=20, pi=0.9
```



Note that we can approximate the binomial distribution with the Poisson distriution (when π is small and n is large) or the normal distribution (when π is mid-range and n is large).

The shape of the Poisson distribution depends on the parameter λ.

```
par(mfrow = c(1, 1))  # One figure on whole page.
x <- 0:10
plot (x, dpois(x, 1), type = "b")  # "b" stands for "both"
                                                  # points and lines.
lines(x, dpois(x, 4), type = "b", col=2, main = 'lambda = 4')  # USE UP-ARROW
lines(x, dpois(x, 6), type = "b", col=3, main = 'lambda = 6')  # lines() adds lines
                                                               # on existing plot.

legend('topright', c(expression(lambda == 1), expression(lambda == 4),
                     expression(lambda == 6)), text.col = c(1, 2, 3), bty = 'n')
# Similarly, dnorm(x, mu, sigma) produces the density function Normal(mu,sigma) .
# See ?dnorm() for required format.
```



2.2 Simulation from Mass and Density Functions

In this section, we will present how to generate data that follow the binomial distribution; i.e., simulate the tossing of a coin, without actually tossing coins. For example, shown below is a way to generate 200 numbers from a binomial:

```
rbinom(200, 10, 0.5)  # format = rbinom(number of tosses, n, pi).
                      # See ?rbinom for more.
```

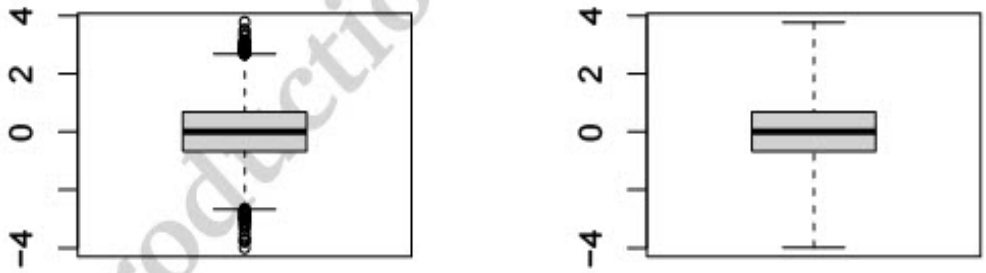
Effectively, you just tossed 10 fair coins, 200 times, each time noting the number of heads out of 10. This way, you can do a lot of experiments on the computer, without actually doing the experiment! If the coin is not fair, then just change the parameter π.

```r
# Putting an "r" before the name is R's way of generating the numbers.
# For example, consider the Possion distribution, which is often used to
# model the number of some event, per unit time, or space, etc. Then,
# rpois(100,4) generates 100 numbers from the poisson distribution. So, each of these
# 100 numbers could be the "number of people arriving at a teller, per hour",
# if the average number of people arriving per hour is 4.
rpois(100, 4) # generates 100 numbers from the Poisson distribution.

# Similarly, the following draws a single sample of size 10000 from a normal
# distribution with mu=0 and sigma=1.
x <- rnorm(10000, 0, 1)
hist(x, breaks = 200)  # Checks the histogram and it looks pretty normal
```

2.3 Boxplots

A boxplot of data is a way of summarizing the data into five numbers that capture the shape of the histogram. The five numbers are the minimum, 25th percentile, median, 75th percentile, and maximum.

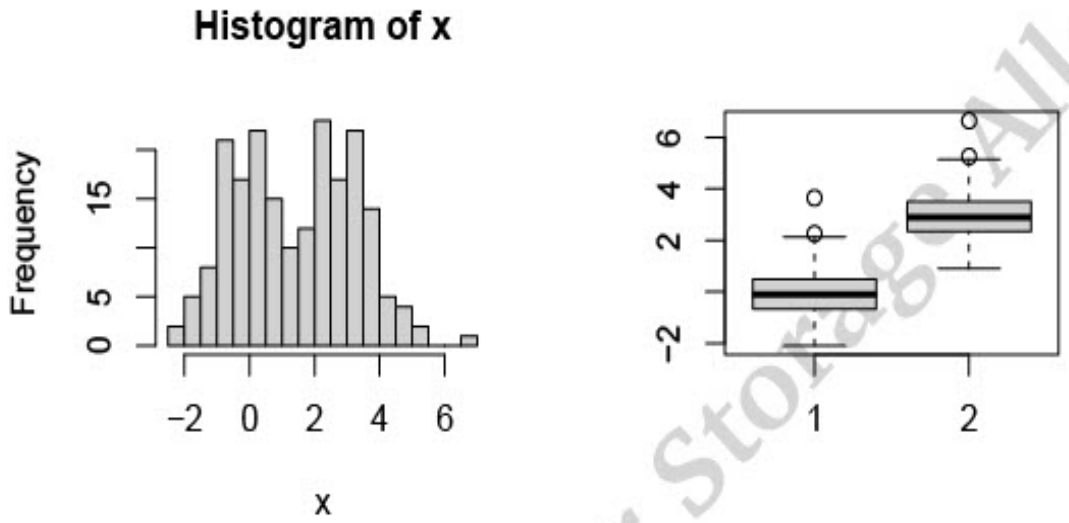
```r
x <- rnorm(10000, 0, 1)
par(mfrow = c(1, 2))
boxplot(x, cex = 0.7)  # Circles at the end of boxplot are outliers according to some
           # criterion.
boxplot(x, range = 0)  # Suppresses outliers.
```



Example 1

Now, recall the bimodal histogram we saw before in hist data. It was bimodal because two separate data files were joined, each one with 100 cases in it. We can separate the two and boxplot them, separately:

```
dat <- read.table('hist_dat.txt', header = F)
x <- dat[, 1]  # All of x.
x_1 <- x[1:100]  # Put the 1st 100 cases of x in x_1,
x_2 <- x[101:200]  # Put the remainder in x_2.
par(mfrow = c(1, 2))
hist(x, breaks = 20)  # Draw a histogram
boxplot(x_1, x_2)  # Draw boxplots
```

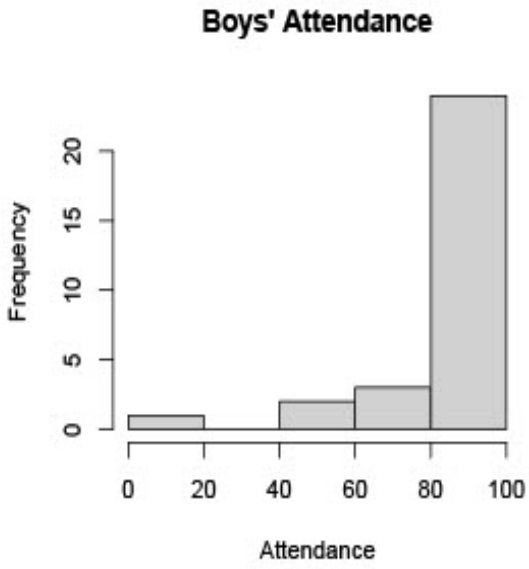


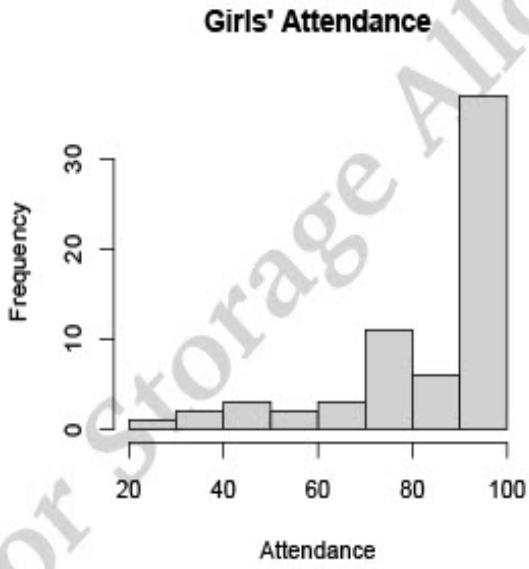
Example 2: Attendance Data

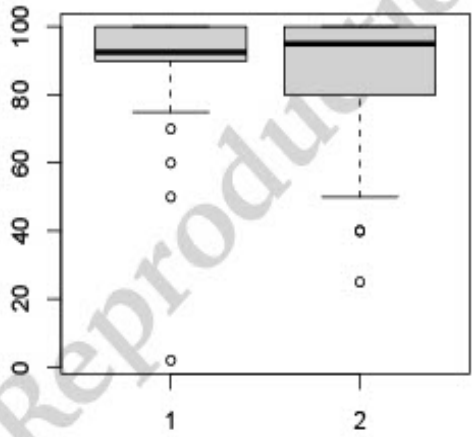
The variable of interest is the "percentage of time student attends lectures", and the two groups are boys and girls.

```
dat <- read.table('attend_dat.txt', header = T)
x <- dat$attendance
y <- dat$Gender

par(mfrow = c(2, 2))
# A way of selecting cases in x that correspond to some value of y.
hist(x[y == 0], main = "Boys' Attendance", xlab = 'Attendance')
hist(x[y == 1], main = "Girls' Attendance", xlab = 'Attendance')
boxplot(x[y == 0], x[y == 1])

# Look at the two sample means to see if there is a difference between
# boys and girls with respect to their attendance.
mean(x[y == 0])  # Sample mean attendance for girls.

[1] 87.57

mean(x[y == 1])  # Sample mean attendance for boys.

[1] 86.4
```

```
# The y=0 group (girls) has the higher sample mean than the y=1 group (boys);
# (87.6 vs. 86.4). But the medians are reversed (92.5 vs. 95):
median(x[y == 0])  # Sample median attendance for girls.

[1] 92.5

median(x[y == 1])  # Sample median attendanceand for boys.

[1] 95
```







There are several sources of complexity in comparing two groups:

1. Sample mean or median measure only "center" or "location" of data.
2. They measure 2 different notions of "center;" and there are many others.

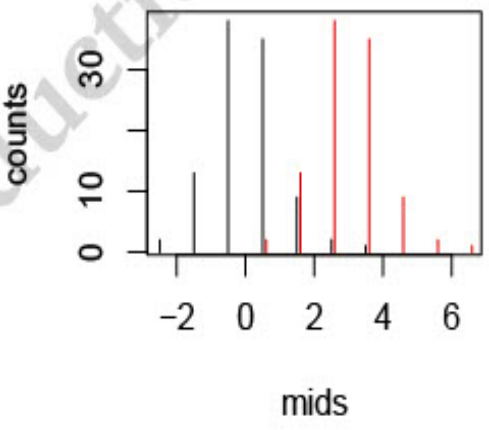
3. Measures of location (e.g., mean, median) do not capture all characteristics of the sample. The spread is equally important.

```
# One measure of spread is the sample standard deviation:
sd(x[y == 0]) # Sample standard deviation of attendance for girls

[1] 20.41

sd(x[y == 1]) # Sample standard deviation of attendance for boys.

[1] 18.02
```

We can see that the spread is a bit wider for girls than for boys. In statistics, some interpretation is always important. For example, one might say that boys are more "consistent" across the sample.

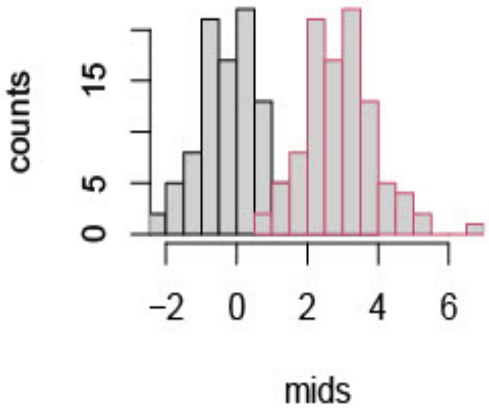
Overlaying two histograms:

```
dat <- read.table('hist_dat.txt', header = F)
x <- dat[, 1]  # Here is all of x.
x_1 <- x[1:100]  # Put the 1st 100 cases of x in x_1,
x_2 <- x[101:200]  # Put the remainder in x_2.
a <- hist(x_1, plot = F)
b <- hist(x_2, plot = F)
x.lim <- range(c(a$mids, b$mids))
plot(a$mids, a$counts, type = "h", xlim = x.lim, xlab = 'mids', ylab = 'counts')
lines(b$mids + 0.1, b$counts, type = "h", col = "red")  # The shift of 0.1 avoids
```



That was the hard way! But it shows the inner-workings of hist(). The easy way is:

```
hist(x_1, breaks = 20, xlim = range(x_1, x_2), xlab = 'mids', ylab = 'counts', main = '')
hist(x_2, breaks = 20, add = T, border = 2)
```



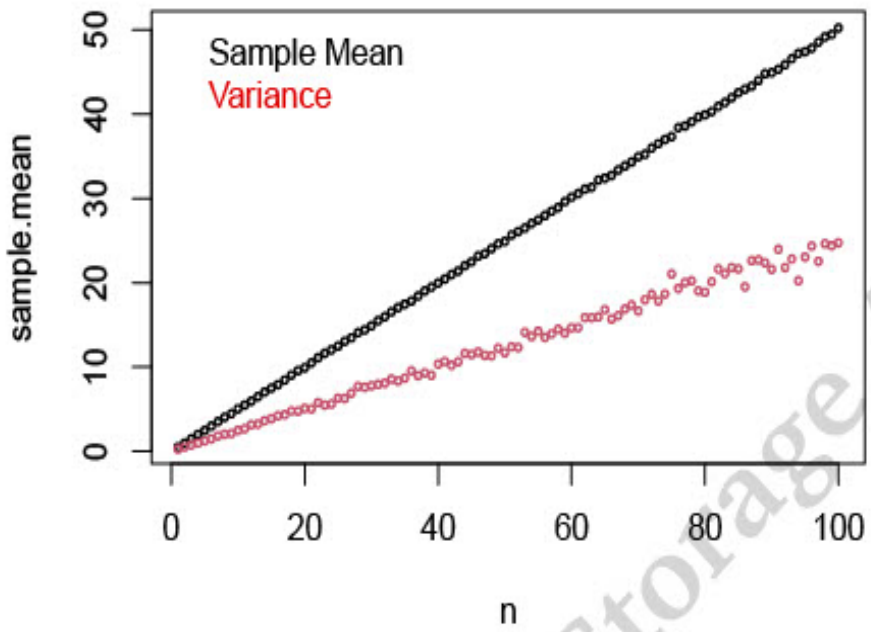
2.4 Binomial Distribution

Recall that the mean and variance of the binomial distribution are given by $n\pi$ and $n\pi(1-\pi)$. Note that they grow linearly with the sample size n. For example, if NG = number of girls in a random sample of size n, then as n increases the typical value of NG increases (obviously), and the variability (across samples) of NG also increases (not obvious). We will confirm this mathematical result with a simulated coin toss example.

Example 1

Toss n fair coins and count the number of heads. Then, repeat 1000 times. You'll get 1000 numbers, each of which is the number of heads out of n. What's the sample mean of these 1000 numbers? What's their variance (spread)? How do these values vary with n?

```
n.trials <- 1000  # Number of repeats.
n <- 1:100  # Values of n to explore.
sample.mean <- numeric(100)  # Generate a vector to store the results
sample.var  <- numeric(100)
for (i in n) {
  head.counts <- rbinom(n.trials, i, 0.5)  # Number of heads in each repeat.
  sample.mean[i] <- mean(head.counts)  # Mean number of heads in 1000 repeats.
  sample.var[i] <- var(head.counts)  # Variance of the 1000 repeats.
  }
plot(n, sample.mean, cex = 0.5) # cex controls the size of points.
points(n, sample.var, col = 2, cex = 0.5)  # points() adds new values on an
                                           # existing plot.
legend('topleft', c('Sample Mean', 'Variance'), text.col = c('black', 'red'), bty = 'n')
```



From the example above, we can see that mean and variance grow linearly with n. This is consistent with the n-dependence in the results $E[x] = n\pi$ and $Var[x] = n\pi(1-\pi)$. It makes sense for the mean to grow linearly - the expected number of heads out of 10 tosses is about 5, while that for 20 tosses is about 10, etc. Now, recall that standard deviation is the square root of the variance. Therefore, standard deviation grows at rate $\sqrt{n}$, which is slower than the mean. In other words, the **spread/uncertainty** of the number of heads out of n grows slower than the mean itself does. This is one of those breaks that nature has given us, because it allows us to effectively "reduce error" (i.e., std. dev.), relative to the "signal" (i.e., mean) by simply increasing sample size.

2.5 Sample Quantile

Suppose our sample/data consists of the following 11 numbers:

```
x <- c(-10, 50, 30, 20, 0, 40, 70, 60, -20, 80, 10)

# The median of this data is 30, as confirmed by
median(x)

[1] 30

# That's because about half of the cases are smaller than 30, and about
# half of the data are larger than 30. (I say "about" because 11 is not
# exactly divisible by 2). The way to see that is to sort the data
sort(x)

 [1] -20 -10   0  10  20  30  40  50  60  70  80

# Note that 5 numbers are lower than 30, and 5 numbers are larger than 30.
```

```r
# There are three other ways of stating this result:

# - 50% of the cases are less than 30.

# - The 50th percentile of the data is 30.

# - The 0.5th quantile of the data is 30, as confirmed by
quantile(x, probs = 0.5)
```

```
50%
 30
```

```r
# Note: 50th percentile = 0.5th quantile.

# Similarly, 10th percentile = 0.1th quantile, and 90th percentile = 0.9th
# quantile.


# Instead of focusing on "50% of the cases," we can ask about 10% of the cases.
# That would be the number in the data that has 10%  of the 11 cases below it,
# i.e., -10, because there is 1 case below -10, as confirmed by

quantile(x, probs = 0.1)
```

```
10%
-10
```

2.6 Distribution Quantile

In this section, we will review the notion of distribution quantile. Recall Table 1 in the textbook, which gives us the area under the standard normal distribution to the left of some number. For example, $z = 1.285$ has about 90% of the area to its left. That means that about 90% of the values of z (ranging from $-\infty$ to $+\infty$) are less than 1.285. In other words, the 90th percentile (or 0.9th quantile) of the standard normal distribution is about 1.285. Table 1 is more precisely encoded into the R function **qnorm()**:

```r
# Finding the 0.9 quantile of a standard normal distribution
qnorm(0.9, mean = 0, sd = 1, lower.tail = TRUE)
```

```
[1] 1.282
```

```r
# Similarly, we can find the 0.1th, 0.2th, 0.3th, ..., 0.9th quantile:
sequence <- seq(0.1, 0.9, by = 0.1)
qnorm(sequence, mean = 0, sd = 1, lower.tail = TRUE )
```

```
[1] -1.2816 -0.8416 -0.5244 -0.2533  0.0000  0.2533  0.5244  0.8416  1.2816
```

```r
# This way, we can compute any quantile of the standard normal distribution.
# In fact, we can find any quantile of any distribution.
```

2.7 Distributions in R

R has a family of functions that allow you to analyze the properties of various known probability distributions easily. In this explanation, we will focus on this family of functions for the normal distribution, but note that these commands analogously exist for most distributions, including (but not limited to) the Exponential, Binomial, Poisson, and T distributions. You may simply substitute the suffix _norm with the appropriate abbreviation of the desired distribution. Please refer to `https://cran.r-project.org/web/views/Distributions.html` for a full list of the probability distributions included in base **R** and their abbreviations.

The four functions we will go over are **dnorm**, **pnorm**, **qnorm**, and **rnorm**.

2.7.1 dnorm

This function returns the value corresponding to the probability *density (mass)* function for continuous (discrete) distributions. For the normal distribution, it returns the y-value on the bell curve when given a value for x and parameters μ and σ. In other words, it plugs x into the following density function for the normal distribution, given values for μ and σ:

$$f_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

As a result, **dnorm** has 3 main inputs: x, mean, sd. x must be an array of numerics corresponding to the values you want plugged into the density function. In discrete distributions, x must be an array of integers. Mean corresponds to μ above, and sd corresponds to σ above, both numerics. Note that,if sd is less than or equal to 0, you will get an error. In R, the default values for mean and sd are 0 and 1,respectively, in all of the *norm* functions. This corresponds to the standard normal distribution. The output for **dnorm** is an array of the same size/shape as the input x.
Let's find the density value for $x = 0$ in the standard normal distribution:

$$\begin{aligned} f_{0,1}(0) &= \textbf{dnorm}(x = 0, \text{mean} = 0, \text{sd} = 1) \\ &= 0.39894 \end{aligned}$$

2.7.2 pnorm

This function returns the value of the *cumulative distribution* function for a probability distribution. For continuous distributions, this is the definite integral taken from the minimum of the density function's support to a point x. For discrete distributions, the integral is replaced with a summation. More specifically, for the normal distribution, this is:

$$F_{\mu,\sigma}(x) = \int_{-\infty}^{x} f_{\mu,\sigma}(x)dx$$

As you may remember from calculus, this computes the area under the curve of the density function f, and this area is found from the minimum up until the point x. We refer to this area as the quantile for the point x. For some intuition, a return of 0 indicates that input corresponds to the minimum of the distribution. Similarly, a return of 1 indicates that the input corresponds to the maximum of the distribution.

Similar to **dnorm**, the three inputs to **pnorm** are an array of numerics x (integers for discrete distributions), the mean (μ), and the sd (σ). The output is similarly an array of the same shape/size of the input x, where the values are always numerics between 0 and 1.

Let's find the quantile value for $x = 0$ in the standard normal distribution:

$$\begin{aligned} F_{0,1}(0) &= \textbf{pnorm}(0, \text{mean} = 0, \text{sd} = 1) \\ &= 0.5 \end{aligned}$$

The value 0.5 means that 0 is the median of the standard normal distribution.

2.7.3 qnorm

This returns the value of the *quantile* function at a given quantile value. This can be thought of as the inverse of the **pnorm** function, where you have the quantile value and you want to know what value of x corresponds to that quantile. The input is an array of quantile values (numerics between 0 and 1) and the output is an array of numerics (integers for discrete distributions) of the same size/shape as the input.

Let's say we didn't know what the median of the standard normal distribution was. We can use qnorm to help us find it:

$$\begin{aligned} F_{0,1}^{-1}(0.5) &= \textbf{qnorm}(0.5, \text{mean} = 0, \text{sd} = 1) \\ &= 0 \end{aligned}$$

We've confirmed that 0 is indeed the median of the standard normal distribution, but we already knew that from our previous **pnorm** example. Note that **pnorm** and **qnorm** are great "by **R**" substitutes for the tables you commonly use when working on your homeworks!

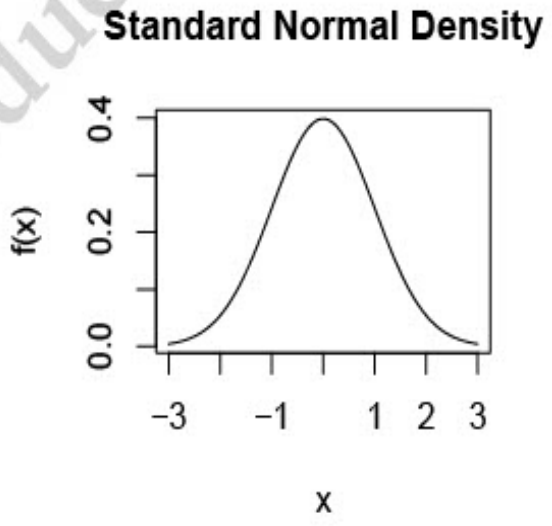
2.7.4 rnorm

This is the most unique of the functions. **rnorm** generates a random sample from a normal distribution. The mean and sd inputs remain the same, but the primary input n is an integer that represents the size of the random sample desired. The output is thus an array of length n.

This function generates random samples from the normal distribution using a technique called Markov Chain Monte Carlo. For a reasonably large n, if you were to produce the histogram of x, it would look like the shape of the normal distribution (density) curve. That is what we mean when we say "take a sample of size n from a normal distribution."

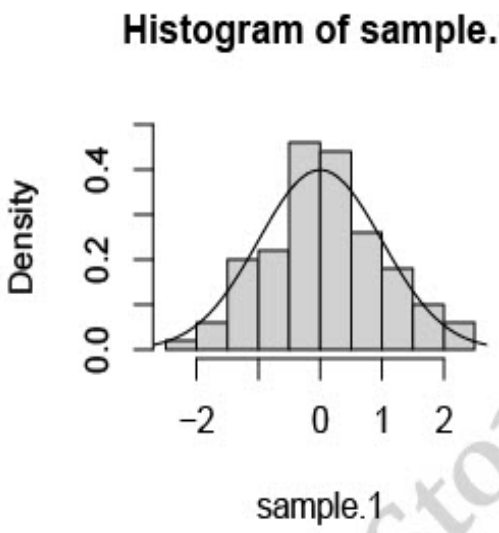
Let's see this in action with the standard normal distribution. We'll first plot the true density curve by using the **dnorm** function.

```
x <- seq(-3, 3, length = 100)
true_density <- dnorm(x, mean = 0, sd = 1)
plot(x, true_density, ylab = "f(x)", main = "Standard Normal Density", type = "l")
```

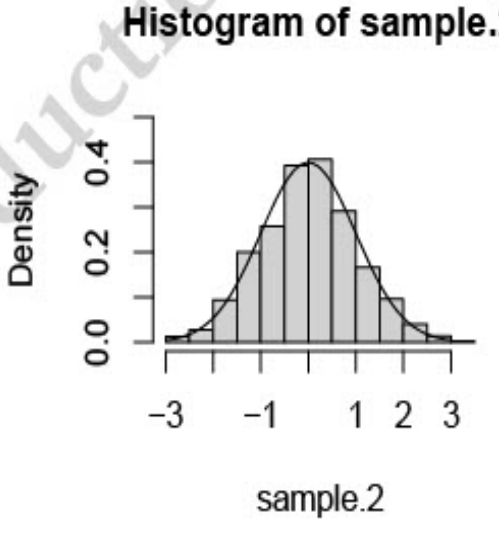


Now let's compare random samples generated by **rnorm** with varying sizes:

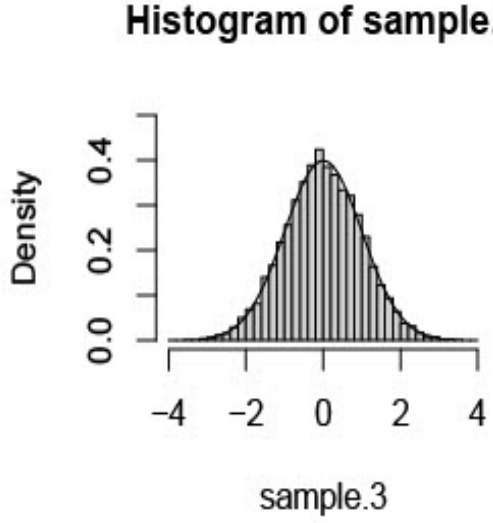
```r
set.seed(123) # IMPORTANT for reproducing the same results shoum here

sample.1 <- rnorm(100, mean = 0, sd = 1) # Sample size 100
hist(sample.1, prob = TRUE, ylim = c(0, 0.5), breaks = 10)
lines(x, true_density)
```




```r
sample.2 <- rnorm(1000, mean = 0, sd = 1) # Sample size 1,000
hist(sample.2, prob = TRUE, ylim = c(0, 0.5), breaks = 20)
lines(x, true_density)
```




```r
sample.3 <- rnorm(10000, mean = 0, sd = 1) # Sample size 10,000
hist(sample.3, prob = TRUE, ylim = c(0, 0.5), breaks = 50)
lines(x, true_density)
```



Thus we see that the greater n is, the more closely it approximates the density curve. As you may have realized, the output of **rnorm** is different with each run. It is best to set a seed so that your results are reproducible, which is especially important when publishing results that rely on a random number generator, or even when debugging your code.

2.7.5 Other Distributions

Note that in the normal distribution, the mean (μ) and standard deviation (σ) uniquely define the distribution. However, in other distributions, other parameters must be specified which then uniquely define the distribution (such as p and n for the Binomial distribution). The parameters that define the distribution are the always inputs, in place of mean and sd in the examples above. If you're curious about a specific function, please use the R help pages. These can easily be accessed by inserting a question mark before a function, such as "**?pnorm**".

2.8 Q-Q Plots

A q-q plot is a plot of sample quantiles versus distribution quantiles for some specified distribution. If the result is a relatively straight "line," then there is some evidence that the data have come from that distribution. More intuitively, there is evidence that the histogram of the data is consistent with the specified distribution. Most often when people talk about a q-q plot, they are assuming that the distribution is the standard normal distribution. So one plots sample quantiles (along the y-axis) versus quantiles of the of the standard normal (along the x-axis). The R corresponding function is **qqnorm**().

Example

Now we take a sample from a standard normal distribution and use qqnorm() to make the q-q plot for the sample. Then, we will make the q-q plot "by hand:"

```
n <- 500  # Sample size = 500.
x <- rnorm(n, 0, 1)  # Sample from a normal distribution with mu = 0, sigma = 1.

qqnorm(x, cex = 0.5)  # The q-q plot, according to qqnorm().
```