Supplementary R Code

Caren Marzban Hoi Yi Ng Corinne Jones Hassan Nasif

April 18, 2021

No Re... Storage Allowed

1 Introduction to R Basics

R is a programming language and software environment for statistical computing and graphics. There are several graphical user interfaces for R including the basic R console, Rstudio and Tinn-R. Rstudio is recommended for R beginners since it is relatively straightforward to use. In Rstudio, when an incomplete command is entered, a '+' instead of a '<' will be returned. We can use Esc to exit the situation described above.

1.1 Basic Commands

```
# Lines starting with [1] represent outputs.
(57 / 276) ^ 2 + 4.3 * .001
x <- 3  # This sets the variable x to the number 3. `<-' is used to
        # assign the the value 3 to x
x  # Typing the variable, followed by return shows its value.
y <- 1 + 1
log10(y)
log2(y)
sqrt(y)
mean(y)  # Sample mean of y: measures "location" of data.
median(y)  # Sample median of y: another measure of "location."
range(y)  # Gives two numbers, min and max.
range(y)[1]  # First component of range(y), i.e., min.
range(y)[2]  # Second component of range(y), i.e., max.
min(y)  # Another way of getting the minimum value.
max(y)  # Another way of getting the maximum value.
length(y)  # Gives the size of y.
dim(y)  # Gives dimension of y when y is a matrix.
sort(y)  # Sorts all the values in y.
sd(y)  # sample standard deviation of y: measures "spread."
```

Variables can be either number, vector, matrix, dataframe, character, or logical expression.

```
is.vector(x)  # Checking if x is a vector
```

Shown below are a few ways of entering data in R.

```
x <- 1:5
y <- seq(from = 0, to = 10, by = 2)  # seq (0, 10, 2), makes a sequence of
```

```
                                    # numbers from 0 to 10 (including 0 and 10),
                                    # in steps of 2.
y <- c(34, 30, 41, 35, 21)
q()  # Quitting R session if using R on terminal.
```

1.2 Viewing Help Files

```
?median  # Shows function definition and examples.
         # Enter q to exit help page.
??median
help.search("histogram")  # Searches all of R (on your computer).
```

1.3 Acquiring Data

```
# Browsing through available data sets in R
data()  # Space bar will scroll through the data sets.
# q: Enter q to exit if using R on a terminal.
```

Example 1

Reading data from R

```
# Loading data set "cars".
data(cars)  # This loads the data.
?cars  # Gives info on data set; if using R on a terminal,
       # space-bar = scrolls, q = quit.
```

```
cars  # Simply prints all the data onto the screen.
```

```
names(cars)  # Displays the names of the variables.

[1] "speed" "dist"

dim(cars)  # Shows the dimensions of data set.

[1] 50  2
```

```
cars$speed  # Use a dollar sign to select a given column/variable, by name.
cars[, 1]  # Same as above, but selects by column number.
x <- cars$speed  # Selects speed (by name) and
                 # assigns to some variable named x.
x  # Shows speeds.
mean(x)  # Calculates the mean of speed.
sd(x)  # Calculates the standard deviation of speed.
```

Example 2

Reading data from an existing file.

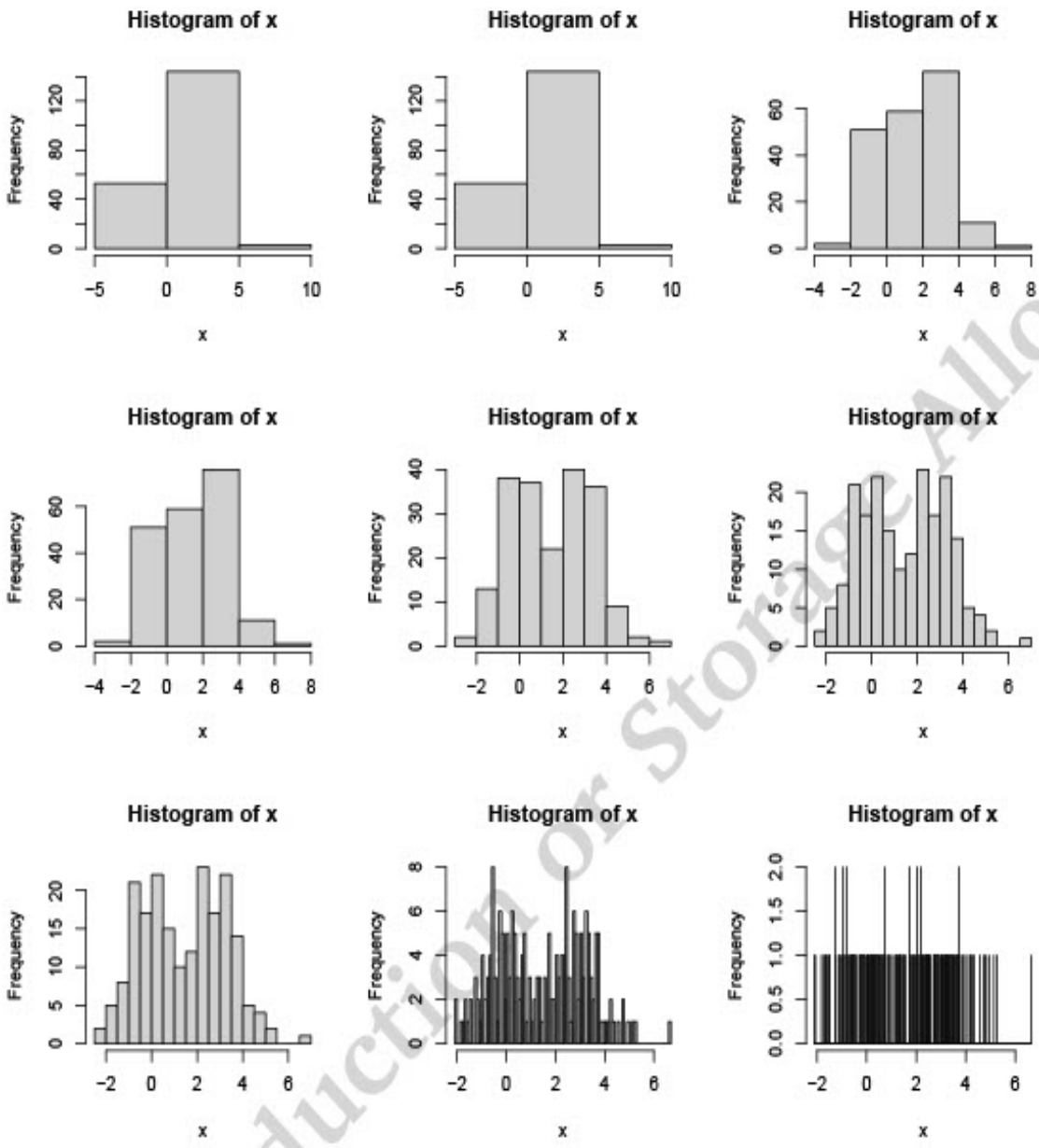
```
# The header=T tells R to ignore the first line/case in the data file.
dat <- read.table(file.choose(), header = T)
```

```
# For reading Excel files, save the file as .csv, and then read it as
dat <- read.csv(file.choose(), header = T)
# In place of file.choose(). a path of the file can be specified.
# Be aware that the path is dependent on the operating system (e.g. linux machines
# use '/' where Window machines use '\\')
# See ?read.csv for details.
```

1.4 Plotting Histograms

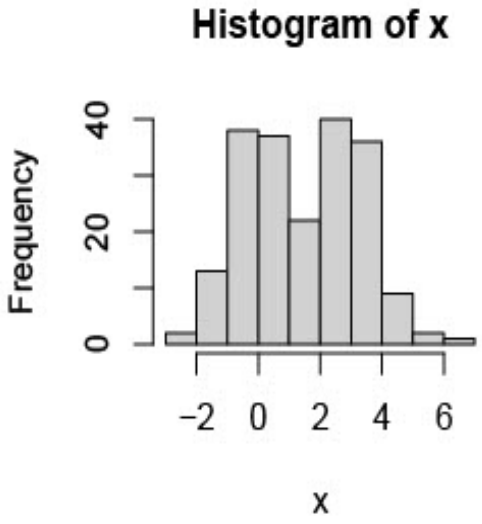
```
dat <- read.table('hist_dat.txt', header = F)
x <- dat[, 1]  # Selects the first column/variable in the data set named dat to plot.
# This command creates a 3 by 3 grid to display the plots
par(mfrow = c(3,3))

# Histograms with different bin sizes
hist(x, breaks = 2)  # Uninformative
hist(x, breaks = 3)
hist(x, breaks = 4)  # Unimodal and bell-shaped.
hist(x, breaks = 5)
hist(x, breaks = 10)  # Bimodal.
hist(x, breaks = 20)
hist(x, breaks = 30)
hist(x, breaks = 100)  # Bimodal + outlier.
hist(x, breaks = 10000)  # Uninformative.
```

No Reproduction Image Allowed




```
# R decides where to put the breaks. It even decides how many breaks,
# in spite of what is specified by "breaks". To over-ride R's preferences
# and to place the breaks in specific places,
# e.g. -3 -2 -1  0  1  2  3  4  5  6  7  :
hist(x, breaks = seq(-3, 7, by = 1))
```

No Reproduction or Storage Allowed



The above suggests that the data/variable x is probably made-up of two different groups. For example, x could be "height," in which case the two "humps" (seen at around breaks = 30 or 100 may be identified as the heights of boys and girls, respectively. This type of analysis is a simple form of datamining, i.e. trying to figure out what is in the data. In general, change the number of breaks, and look for changing patterns that may be of interest.

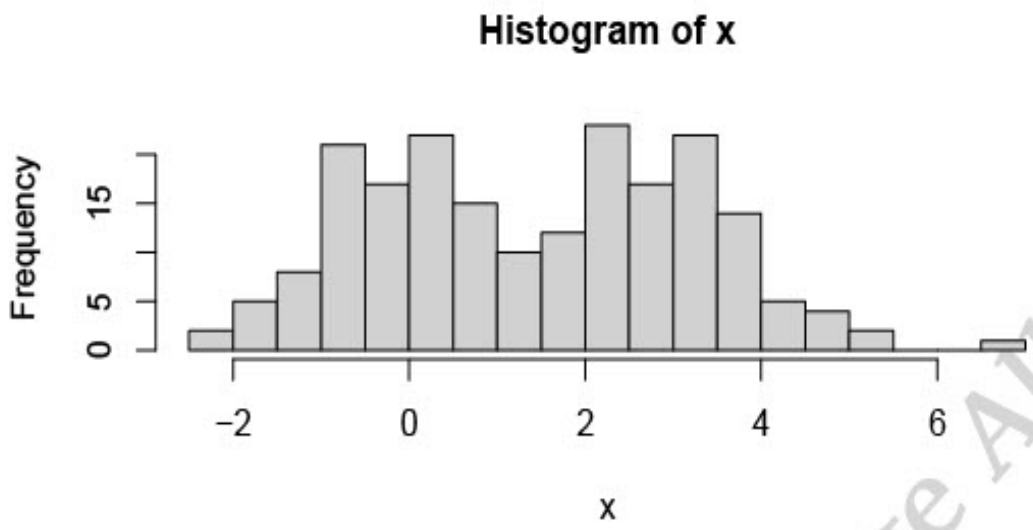
1.4.1 Making "Density Scale" Histograms

One variation on the above (frequency) histogram is the relative frequency histogram, wherein the frequencies on the y-axis are divided by the total number of cases (also called sample size). In this example, all of the bar heights will then be divided by 200. Another variation is obtained by further dividing the relative frequencies by the width of the bars. This version of a histogram is said to be a density (or density-scale) histogram. The advantage of the density histogram is that the area under it is 1. This, then, allows us to interpret the area under the histogram, between any two values of x, as the probability of obtaining an x value in that interval.

To confirm that the area is 1, check this out.

```
H = hist(x, breaks=20)  # Direct the returned values of hist() to H.
names(H)                # Show what is returned (also shown in the help pages for hist().

[1] "breaks"   "counts"   "density"  "mids"     "xname"    "equidist"
```

No Reproduction or Storage Allowed



Now, examine the values of the following:

```
 H$breaks                # Evidently, R decided to let the bin size be 0.5.

 [1] -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5
[16]  5.0  5.5  6.0  6.5  7.0

 H$density               # The y-values shown on the density histogram.

 [1] 0.02 0.05 0.08 0.21 0.17 0.22 0.15 0.10 0.12 0.23 0.17 0.22 0.14 0.05 0.04
[16] 0.02 0.00 0.00 0.01
```

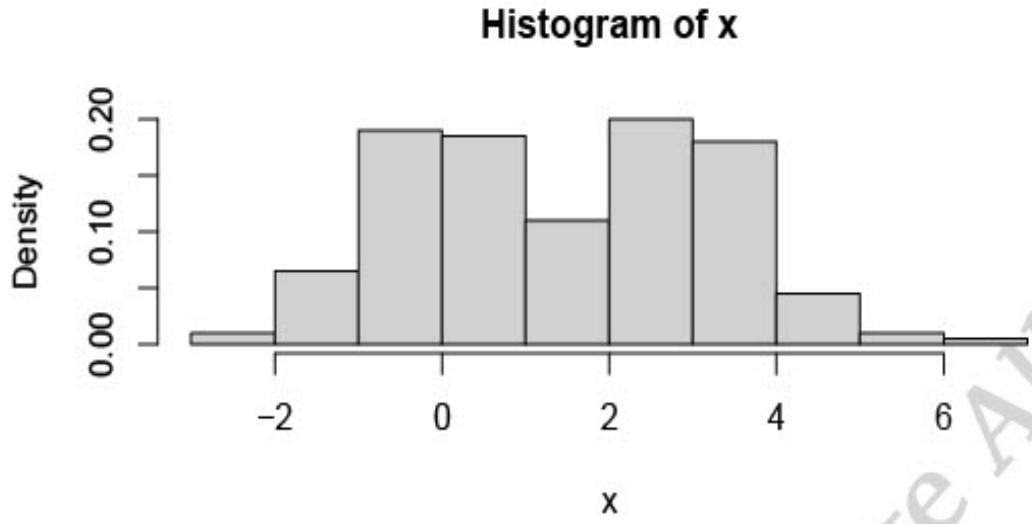
And finally, we can confirm that the total area is 1:

```
 sum(0.5 * H$density)    # = 1

[1] 1
```

The R function hist() does allow for making a density histogram:

```
hist(x, freq = FALSE)  # Density scale histogram.
```

No Reproduction/Storage Allowed



But the reason we did it the "hard way" was to learn more about what R functions return, and how to use them.

1.5 Outputting Results

```
# Making a pdf file of a graph and call the file "hello.pdf".
setwd("C:\\Temp")  # Sets the directory to C:\\Temp.
pdf("hello.pdf")  # The file "hello.pdf" is placed in C:\\Temp.
# In Rstudio, this can be done by clicking 'export' on top of the graph and
# choosing a folder to save the graph to.
par(mfrow = c(1, 2))
hist(x)
hist(x, freq = FALSE)
dev.off()  # This makes sure the pdf file is closed.
```

No Reproduction Storage Allowed

2 Distributions

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, and Poisson distributions. For a full list of the probability distributions included in base **R** and their abbreviations, refer to `https://cran.r-project.org/web/views/Distributions.html`.

The four functions we will go over are **dnorm**, **pnorm**, **qnorm**, and **rnorm**. For other distributions you may simply substitute the suffix _norm with the appropriate abbreviation of the desired distribution. The prefixes d, p, q, and r, don't change for other distributions.

2.0.1 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. 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:

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

2.0.2 pnorm

This function returns the area to the left of some value; FYI, it's also called the *cumulative distribution* function. For continuous distributions, this is the definite integral taken from the smallest possible x to the x of interest. 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.

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 area to the left of $x = 0$ for 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 middle/center (technically, median) of the standard normal distribution.

Non-reproduction or storage Allowed

2.0.3 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; **pnorm** tells you the area to the left of some specified x; **qnorm** tells you what value of x has some specified area to its left. The input is an array of area values (numerics between 0 and 1) and the output is an array of x values 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 find it:

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

If the $F_{0,1}^{-1}(0.5)$ on the left is confusing, feel free to ignore it; just know how **qnorm** works. We've confirmed that 0 is indeed the median of the standard normal distribution. Note that **pnorm** and **qnorm** are "by **R**" substitutes for the "by hand" tables you commonly use when working on your homeworks!

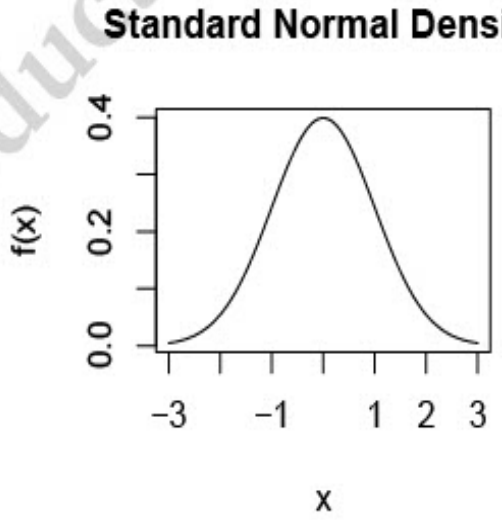
2.0.4 rnorm

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.

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.

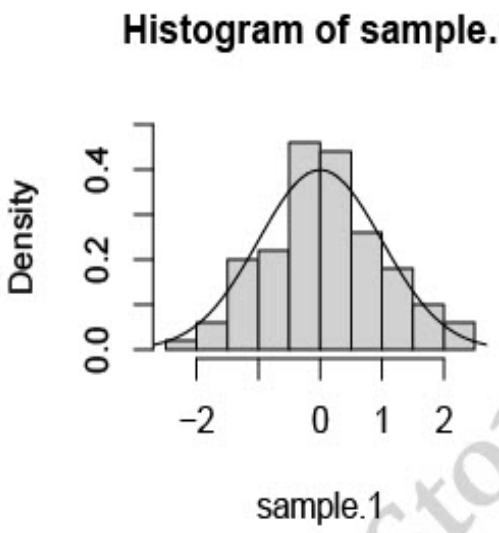
```
x <- seq(-3, 3, length = 100)
true_density <- dnorm(x, mean = 0, sd = 1)
plot(x, true_density, ylab = "f(x)", main = "Standard Normal Density", type = "l")
```



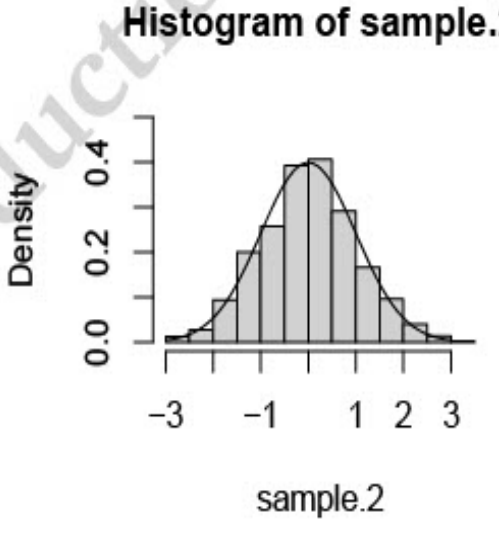
Now let's compare random samples generated by **rnorm** with varying sizes:

No Reproduction or Storage Allowed

```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)
```

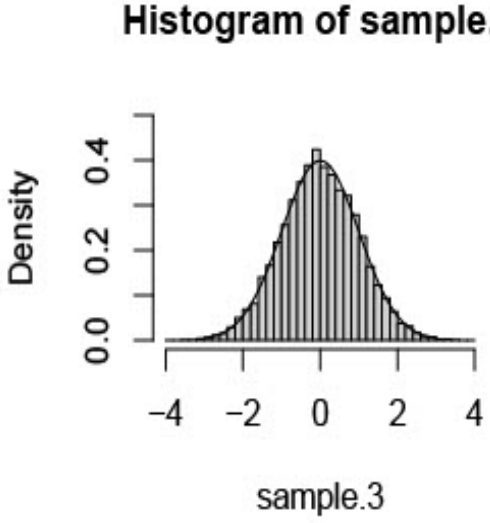



```r
sample.2 <- rnorm(1000, mean = 0, sd = 1) # Sample size 1,000
hist(sample.2, prob = TRUE, ylim = c(0, 0.5), breaks = 20)
lines(x, true_density)
```




```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)
```

No Reproduction ~ Storage Allowed



Thus we see that the greater n is, the more closely it approximates the density curve. Because each sample is random, 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.

2.0.5 Other Distributions

Note that in the normal distribution, the μ and σ parameters 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). Consequently, the parameters that define the distribution are always some of the inputs to the corresponding R functions. 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**".

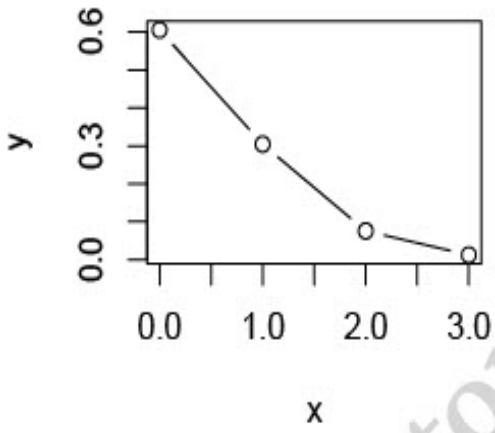
2.1 Binomial and Poisson Distribution

```
# 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

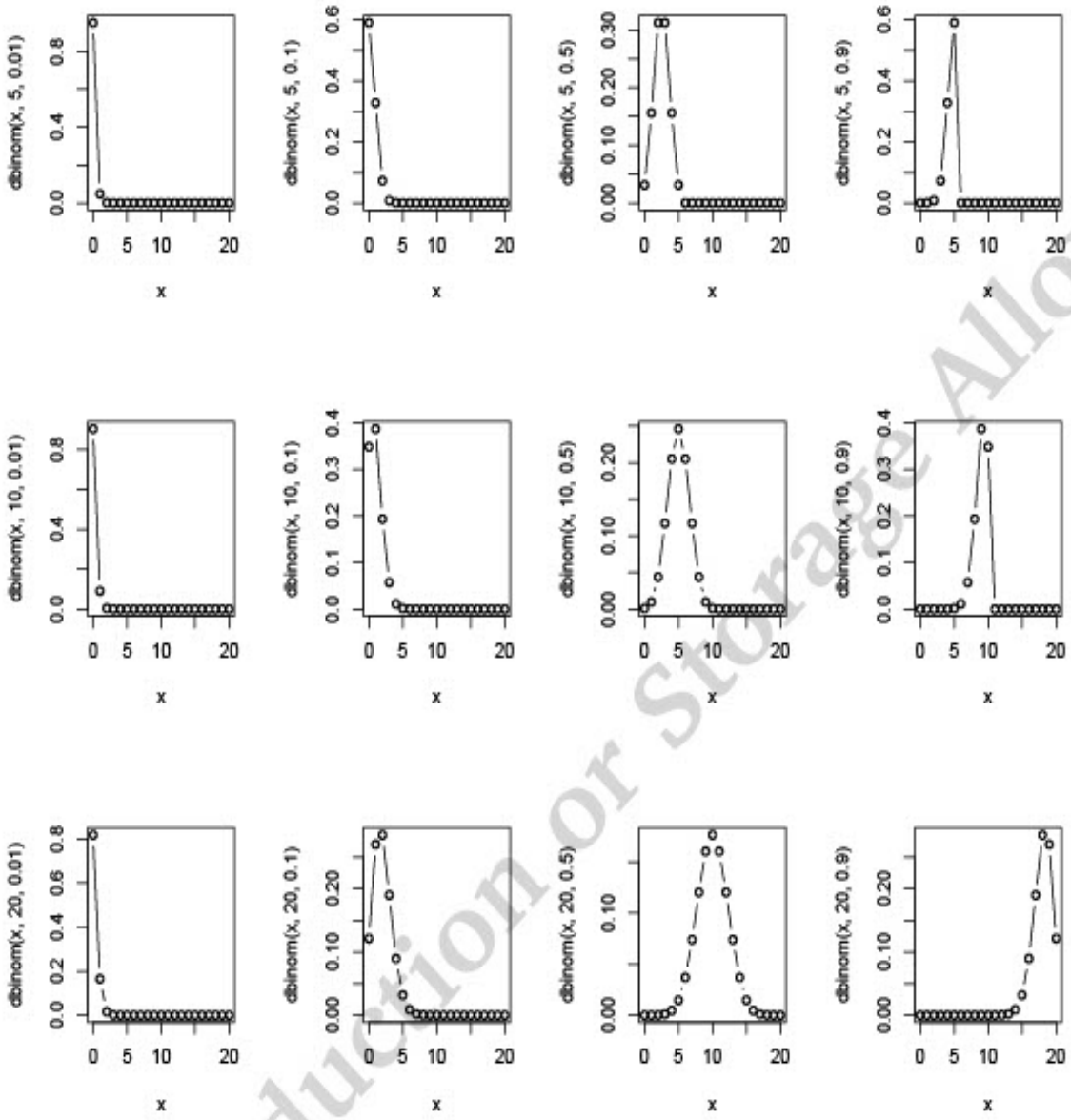
Non-commercial Storage Allowed

```r
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
```




```r
# 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
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
```

No Storage Allowed



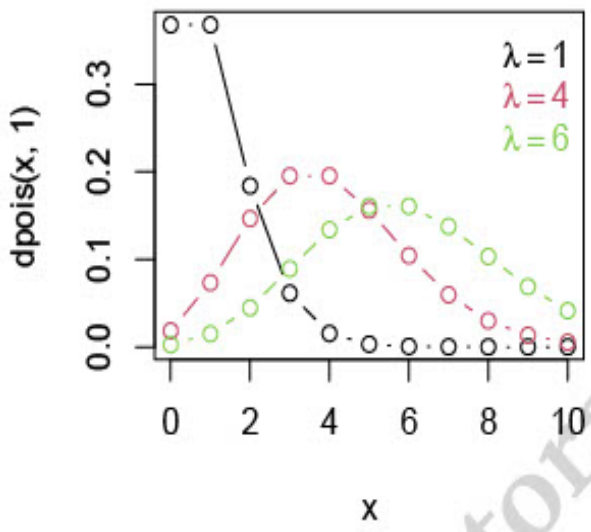
Note that we can approximate the binomial distribution with the Poisson distriution (when π is small and n is large) or the normal distribution (when π is mid-range and n is large).

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) .
```

No Reproduction or Storage Allowed

```
# See ?dnorm() for required format.
```



2.1.2 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.

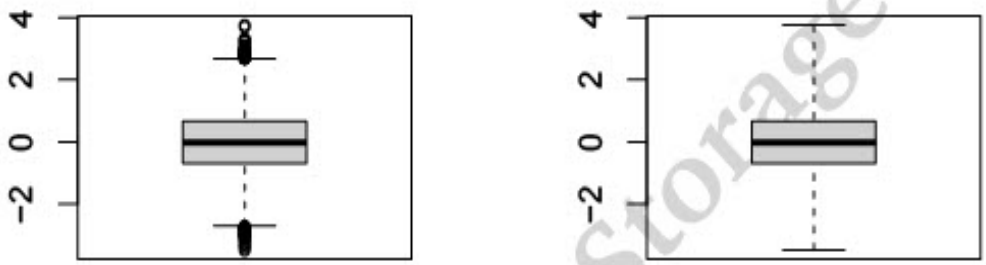
```
# Putting an "r" before the name is R's way of generating the numbers.
# For example, consider the Possion distribution, which is often used to
# model the number of some event, per unit time, or space, etc. Then,
# rpois(100,4) generates 100 numbers from the poisson distribution. So, each of these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

No Reproduction or Storage Allowed

2.2 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.

```
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.
```

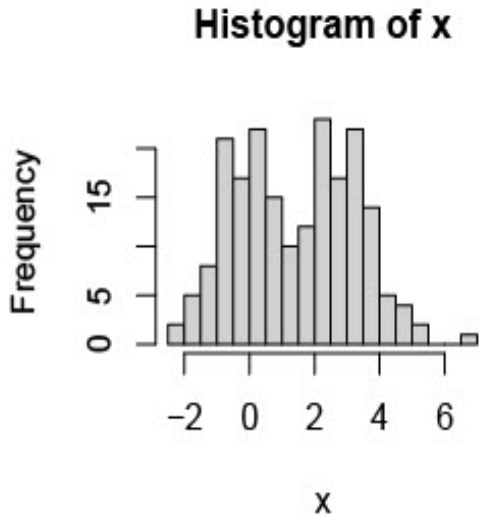


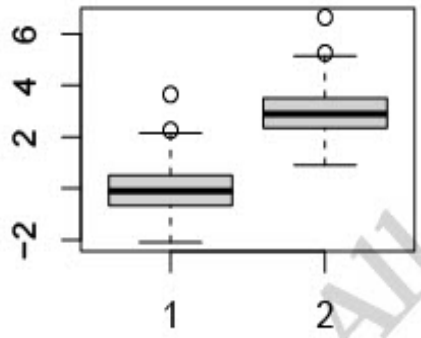
Example 1

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
```

No Reproduction or Storage Allowed



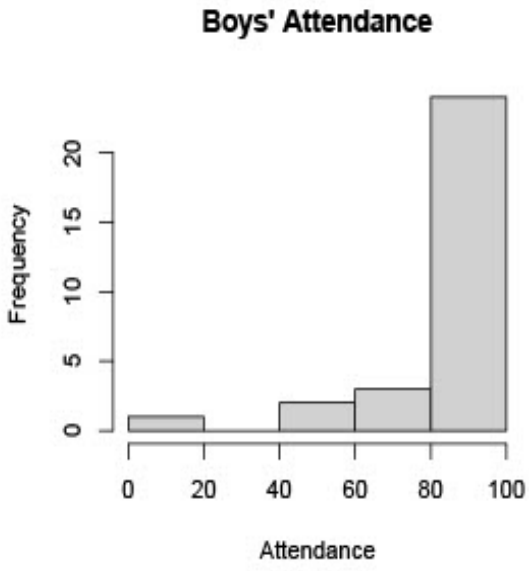


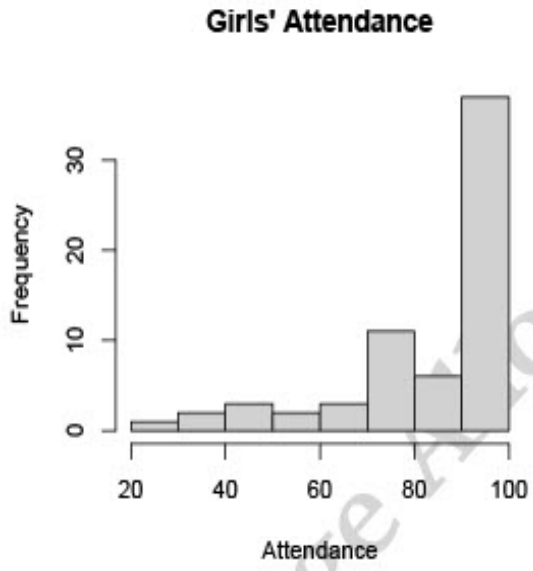
Example 2: Attendance Data

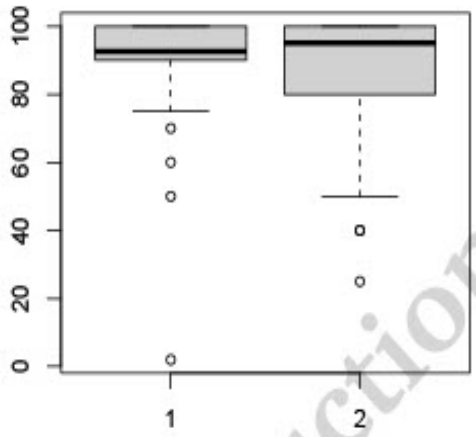
The variable of interest is the "percentage of time student attends lectures", and the two groups are boys and girls.

```
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
```

Storage Allowed







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
```

No Reproduction or Storage Allowed

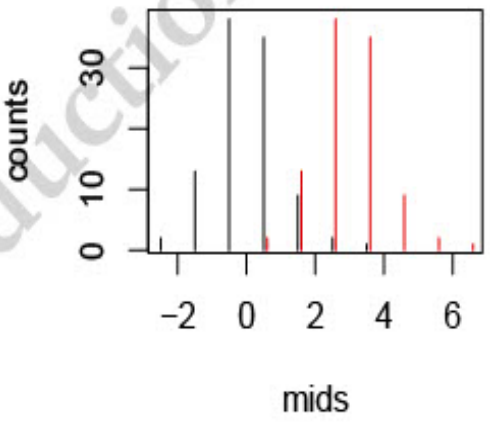
We can see that the spread is a bit wider for girls than for boys. In statistics, some interpretation is always important. For example, one might say that boys are more "consistent" across the sample.

The above analysis of comparative boxplots is extremely incomplete. For a more complete discussion, see the lecture notes. Get used to the idea that if there is "too much" overlap between boxplots (or histograms), then one has to be very careful about generalizing what the sample-based conclusions to the population; in those cases, the correct conclusion is "The data do not provide sufficient evidence for $\cdots$." In other words, the correct conclusion is "We can't tell." But, like I said, read the lecture notes.

Overlaying two histograms

Even though boxplots are truly useful when comparing multiple data sets (or distributions), there are still times that you need to show the full histograms (i.e., not summarized by boxplots). Here is one way of overlaying two histograms onto the same plot:

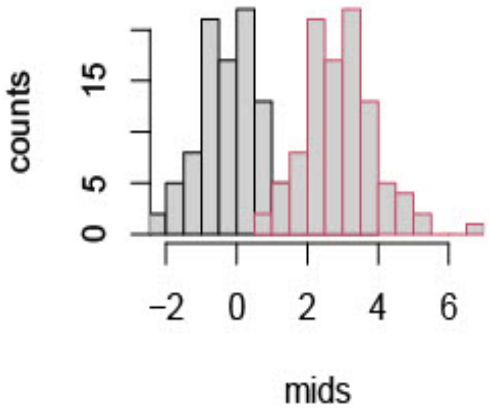
```
dat <- read.table('hist_dat.txt', header = F)
x <- dat[, 1]  # Here is all of x.
x_1 <- x[1:100]  # Put the 1st 100 cases of x in x_1,
x_2 <- x[101:200]  # Put the remainder in x_2.
a <- hist(x_1, plot = F)
b <- hist(x_2, plot = F)
x.lim <- range(c(a$mids, b$mids))
plot(a$mids, a$counts, type = "h",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

No Reproduction or Distribution Allowed



2.3 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.
# 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

# Note: 50th percentile = 0.5th quantile.

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

Storage Allowed

```
# 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
```

2.4 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()**:

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

```
[1] 1.282
```

```
# 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
```

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

2.5 Q-Q Plots

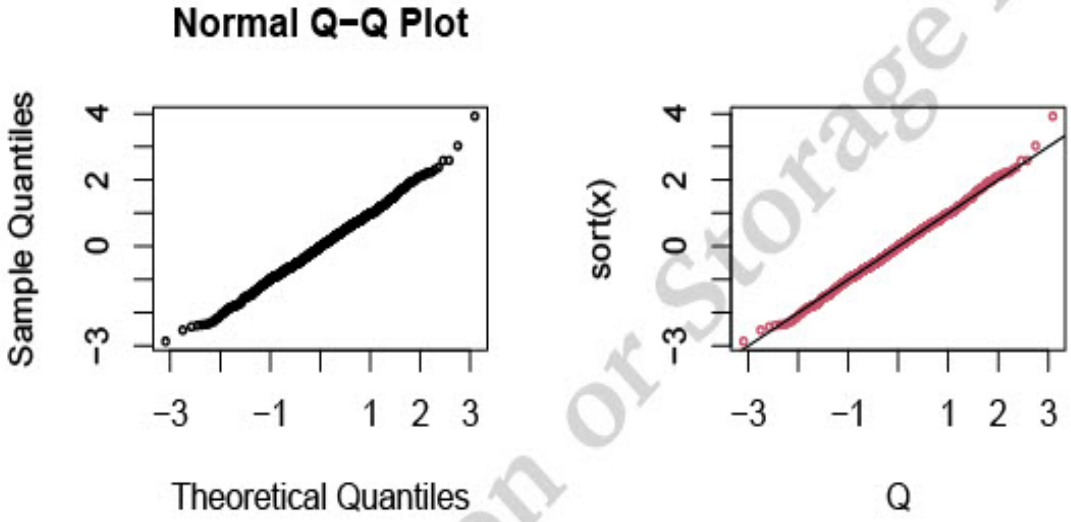
A q-q plot is a plot of sample quantiles versus distribution quantiles for some specified distribution. If the result is a relatively straight "line," then there is some evidence that the data have come from that distribution. More intuitively, there is evidence that the histogram of the data is consistent 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:"

No Reproduction for Storage Allowed

```r
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().

# Doing it by hand:
X <- seq(.5 / n, 1 - .5 / n, length = n)  # Make a sequence of n values between
                                          # (1-.5)/n and (n-.5)/n , and find
Q <- qnorm(X, mean = 0, sd = 1)  # their quantiles under standard normal.
plot(Q, sort(x), col = 2, cex = 0.5)  # plot the data (sorted) vs. the
                                      # quantiles.
abline(0, 1)  # Add a line with slope 1 and intercept 0.
```



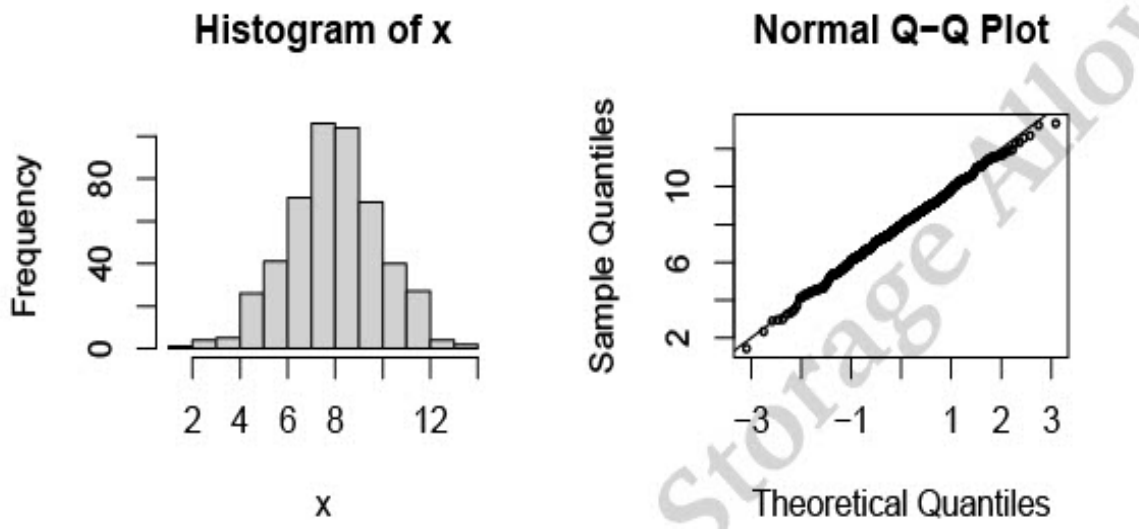
Note that the two q-q plots are the same. Make sure you understand what's done here: When we have 500 observations, each one is associated with a percentile; e.g., the smallest observation has nothing less than it, and so its percentile (rank) is 0. Meanwhile, the largest obs has everything less than it, and so, its percentile (rank) is 100%. In terms of quantiles (instead of percentiles), these two limits are 0 and 1; and that's why we have X go from 0 to 1. In fact, the numbers go from a number close to 0 to a number close to 1, namely from $.5/n$ to $1 - .5/n$; this is mostly a matter of convention for handling the lowest and the largest element in the sample. Then, Q = qnorm(X,0,1) returns the quantiles of the standard normal dist.

It's not obvious but (e.g., see the book) the slope of the "line" is the σ parameter of the normal distribution from which the data have come, and the y-intercept of the line is equal to the μ parameter of the distribution. In this case, we can see that the slope is around 1, and the y-intercept is around 0. We have confirmed this by drawing a line with slope = 1, y-intercept = 0. Note that the line we have just drawn is NOT the "best fit" line to the q-q plot. When we talk about the slope or the intercept of the q-q plot, think of them as the "visual slope" and the "visual y-intercept" instead.

Now, let's consider data coming from a normal distribution with $\mu = 8$, $\sigma = 2$ (i.e., not standard). If we continue using the quantiles of the standard normal along the x-axis, it turns out the slope will then be 2, and the intercept will be 8. In other words, the interpretation of the q-q plot is still the same - the slope is going to be close to the σ of the distribution from which the data were drawn, and the intercept will be approximately the μ of the distribution.

Not Reproduction or Storage Allowed

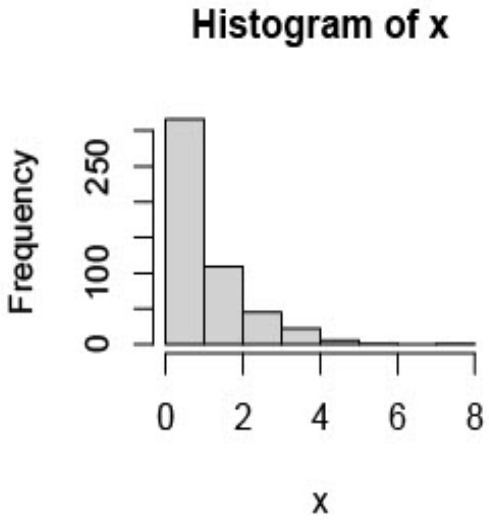
```r
n <- 500  # Sample of size 500.
x <- rnorm(n, 8, 2)  # Generate data from normal(mu=8,sigma=2).
hist(x)
# qqnorm() checks the data against a normal distribution.
qqnorm(x, cex = 0.5)  # Note the slope and intercept are about 8 and 2.
abline(8, 2)  # Add a line with slope = 2, intercept = 8.
```

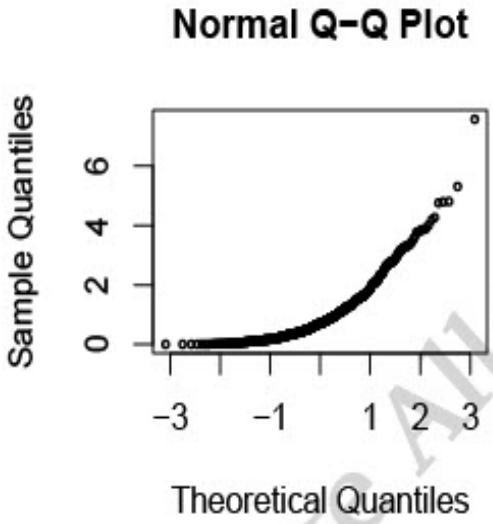


Now, let's get a sense of what a standard normal q-q plots look like for data from non-normal distributions. Recall that if the data come from a normal distribution, their q-q plot should look like a straight line, at least in the bulk of the plot; the tails usually deviate from a straight line, because there are usually few cases there anyway. A normal q-q plot is a visual method for checking whether data are normally distributed. Also, if linear, then the intercept and slope of the line can be used as estimates of the μ and σ of the normal distribution, respectively.

```r
x <- rexp(n, 1)  # Sample of size 500 from an exponential dist with lambda = 1.
hist(x)
qqnorm(x, cex = 0.5)  # Note that it is not straight at all.
```

No Reproduction or Storage Allowed

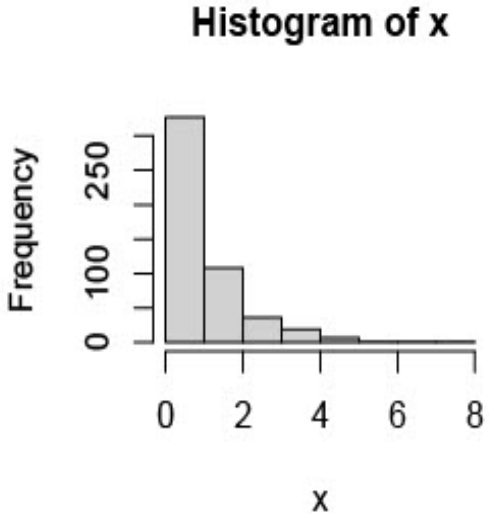


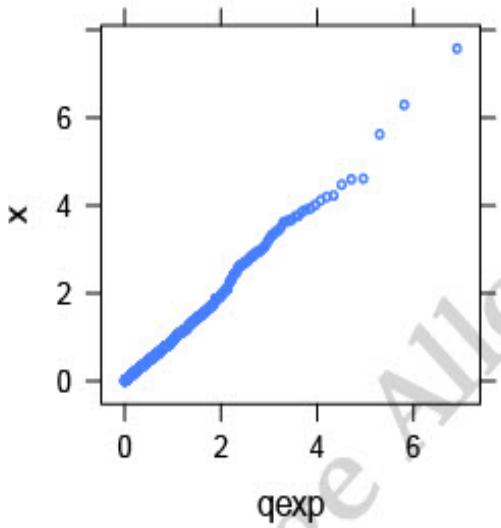


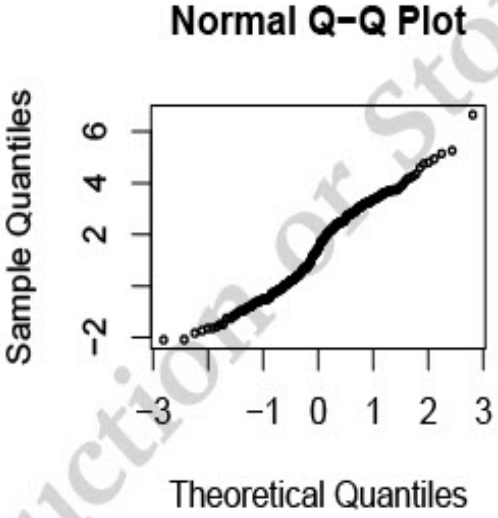
As we can see, a non-normal sample will not produce a linear pattern because qqnorm() checks the data against the standard normal distribution. But how do we identify if some data come from some other distribution, say, from the exponential distribution? The solution is to use analog of qqnorm for the exponential distribution. For example, we can plot quantiles of data versus quantiles of the exponential distribution. In R, the corresponding function is **qqmath()**, which allows for a large number of theoretical distributions.

```
library(lattice)  # Load the library that contains qqmath().
x <- rexp(500, 1)  #Sample of size 500 from an exponential dist with lambda = 1.
hist(x)
qqmath(x, dist = qexp, cex = 0.5)

# The q-q plot is a straight "line" even though the data are from an
# exponential distribution.

# Finally, recall that in section 1, we saw a bimodal histogram.
# Check out its q-q plot:
dat <- read.table('hist_dat.txt', header = F)
qqnorm(dat[, 1], cex = 0.5)
# You can see that there are two linear segments, parallel (i.e., same sigma),
# but different intercepts (i.e., different mu).
```

No Reprint • Storage Allowed







No Reproduction or Storage Allowed

2.6 Jargon

As explained in class, in statistics, we use distributions to represent populations. In fact, we can think of distributions and populations as one and the same thing. Specifically, when we talk about "a sample from a population," what we really mean is "a sample from a distribution." That jargon can be confusing for some of the distributions (especially, Binomial), but it will help if every time you hear "a sample from a distribution," you translate that into what the actual data in the sample would look like. Consider the following examples, and their translation:

"A sample of size n from a Bernoulli distribution with parameter π." Translation: n numbers, each either 0 or 1, and the proportion of 1's is around π.

"A sample of size n from a Normal distribution with parameters μ and σ." Translation: n numbers, each between $\pm\infty$, with most around μ, and a typical deviation around σ.

"A sample of size m from a Binomial distribution with parameters n and π." Translation: m numbers, each an integer between 0 and n. (Later, you'll see how π affects the sample.)

The last one is particularly confusing because of the names R uses to refer to the parameters of the Binomial distribution. Specifically, what we call the n and π parameters of the Binomial distribution are called "size" and "prob." And, to make matters worse, take a look at the help pages for **rbinom**; you'll see "rbinom(n, size, prob)," and so the n that appears in there is NOT what we call the n parameter of Binomial. I did say that it's confusing!

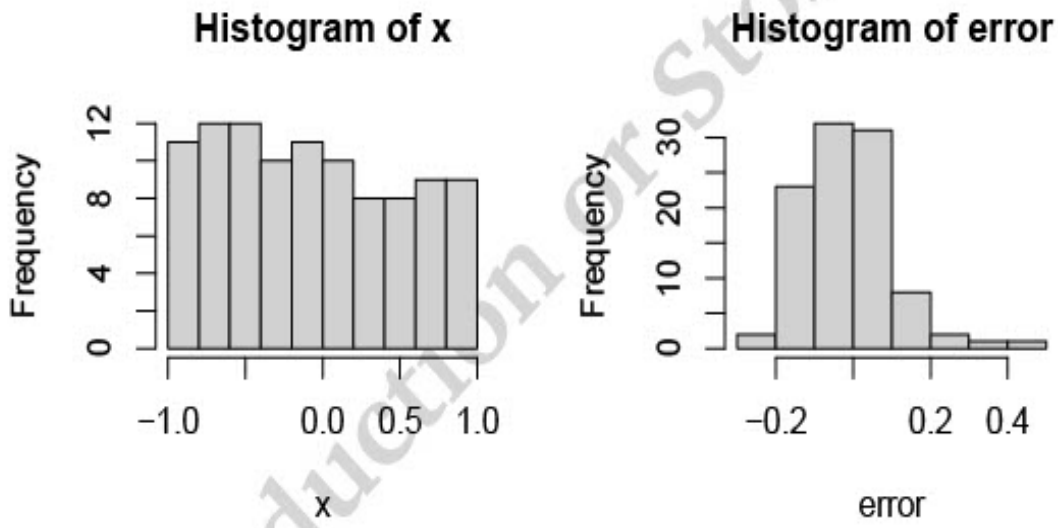
No Reproduction or Storage Allowed

3 Regression

3.1 Scatter Plots

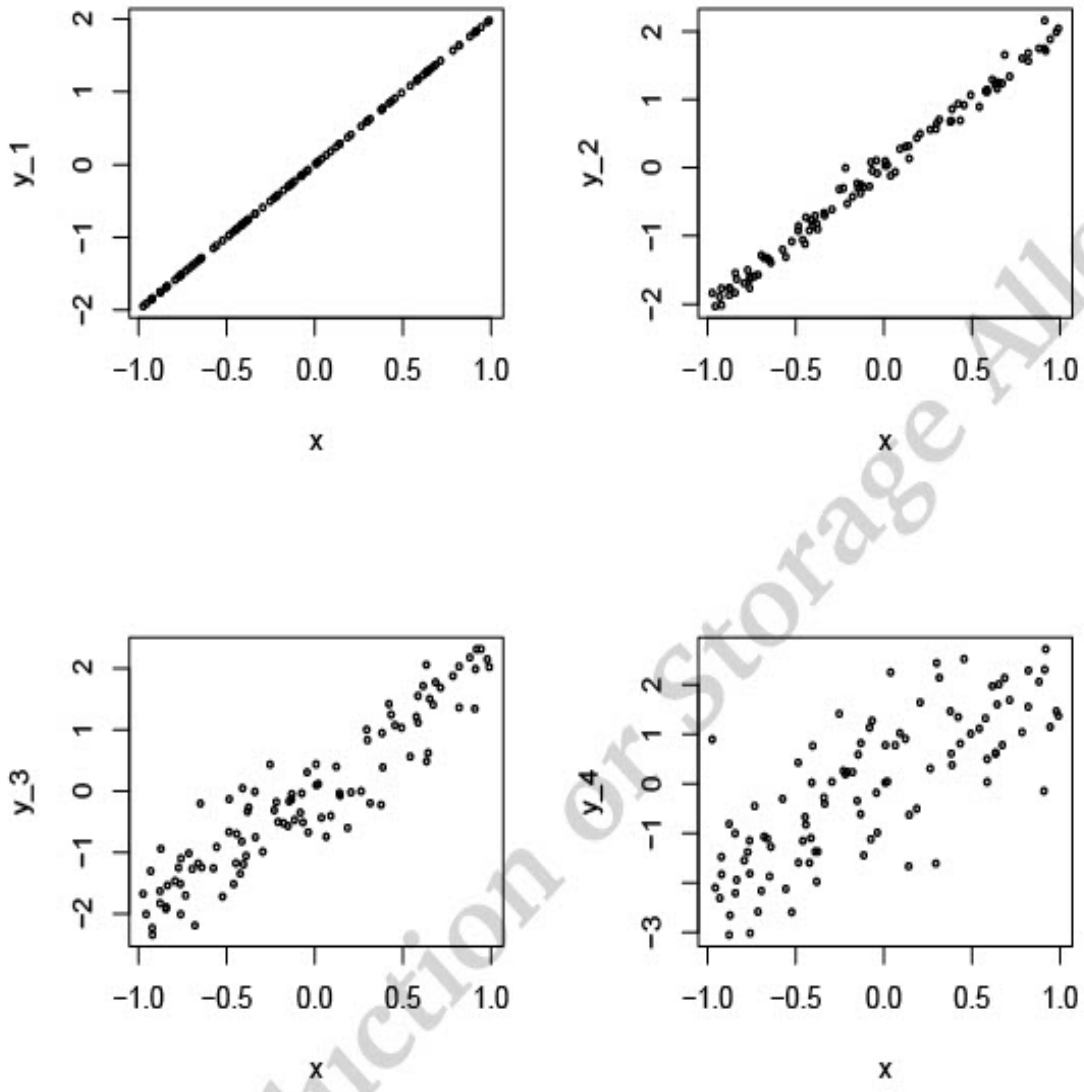
The best way of visualizing the relationship between two continuous random variables is through a scatterplot. (Just in passing, the analog for two categorical variables is the Contingency Table, also called the Confusion Matrix.) It can convey a great deal of information, including whether or not the relationship linear, and the extent of the strength of the relationship. Here, strength refers to the skinniness of the scatterplot. Let's illustrate through an example: Pick 100 random x values, and corresponding y values that have some linear association with x and change the amount of linear association by adding different amounts of "error" to y.

```
par(mfrow = c(1, 2))
x <- runif(100, -1, 1)  # Take 100 points from a uniform distribution between
                        # -1 and 1.
hist(x)  # The shape and looks uniform.
error <- rnorm(100, 0, 0.1)  # Generate a normal variable (the error), with mu=0,
                             # sigma=0.1
hist(error)  # The shape looks normal.
```




```
y_1 <- 2 * x  # Perfect linear relation between x and y.
y_2 <- 2 * x + error  # With some error added to y.
y_3 <- 2 * x + rnorm(100, 0, 0.5)  # With more error added to y.
y_4 <- 2 * x + rnorm(100, 0, 1.0)

par(mfrow = c(2, 2))
plot(x, y_1, cex = 0.5)
plot(x, y_2, cex = 0.5)
plot(x, y_3, cex = 0.5)
plot(x, y_4, cex = 0.5)  # Note that too much noise makes it hard to see
                         # the linear relationship between x and y.
```

Not for Reproduction or Storage Allowed



3.2 Correlation

To quantify the strength of the association between two continuous variables, Pearson's correlation coefficient (i.e., correlation), can be computed. It measures the 'amount of scatter' (i.e., skinniness) in a linear sense (but NOT about "the fit").

```
cor(x, y_1)  # Check against the scatterplots, to get a feeling for r (correlation).

[1] 1

cor(x, y_2)

[1] 0.995

cor(x, y_3)
```

No Reproduction or Storage Allowed

```
[1] 0.9321
```

```
cor(x, y_4)
```

```
[1] 0.7744
```

```
cor(y_4, x)  # r is symmetric.
```

```
[1] 0.7744
```

```
cor(y_4, x + 10)  # r is invariant under shifts.
```

```
[1] 0.7744
```

```
cor(x, 10 * y_4)  # r is invariant under scaling.
```

```
[1] 0.7744
```

3.2.1 Defects of Correlation

Pearson's correlation coefficient, r, can become misleading in several situations.

```
set.seed(123)  # Set a seed to get reproducable results.
x <- runif(100, 0, 1)
error <- rnorm(100, 0, 0.5)
y <- 1 + 2 * x + error
x_1 <- rnorm(100, 0, 50)
y_1 <- rnorm(100, 0, 50)
x_2 <- 1000 + rnorm(100, 0, 50)
y_2 <- 1000 + rnorm(100, 0, 50)
plot(x, y, main = 'Without Outliers', cex = 0.5)
cor(x, y)
```

```
[1] 0.7562
```

```
# Effect of outliers:
x[101] <- 0.2  # Adding one outlier can artificially reduce r.
y[101] <- 8.0
plot(x, y, main = 'With Outlier (0.2, 8.0)', cex = 0.5)
cor(x, y)
```

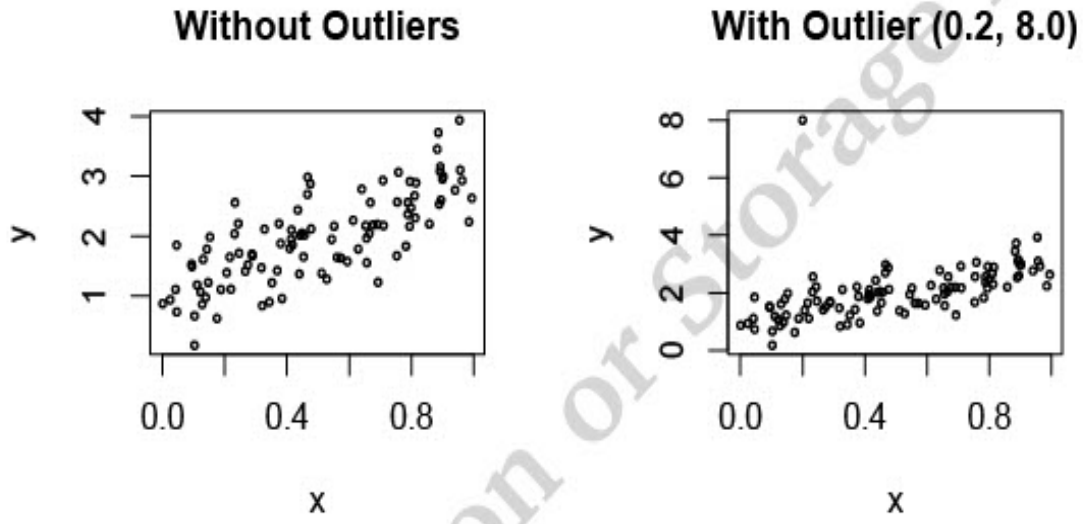
```
[1] 0.516
```

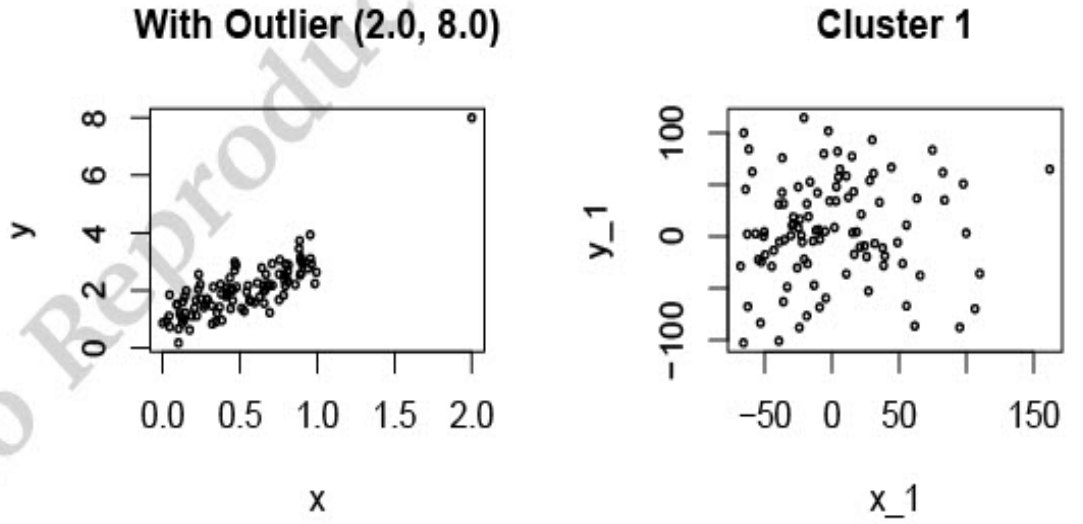
```
x[101] <- 2.0  # A different outlier can artificially increase r.
y[101] <- 8.0
plot(x, y, main = 'With Outlier (2.0, 8.0)', cex = 0.5)
cor(x, y)
```

```
[1] 0.8129
```

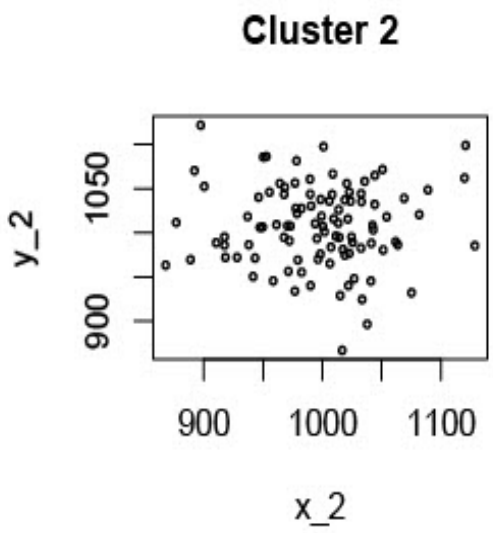
```
# Clusters can also make r meaningless.
plot(x_1, y_1, main = 'Cluster 1', cex = 0.5)
cor(x_1, y_1)  # No correlation between x and y in cluster 1
```

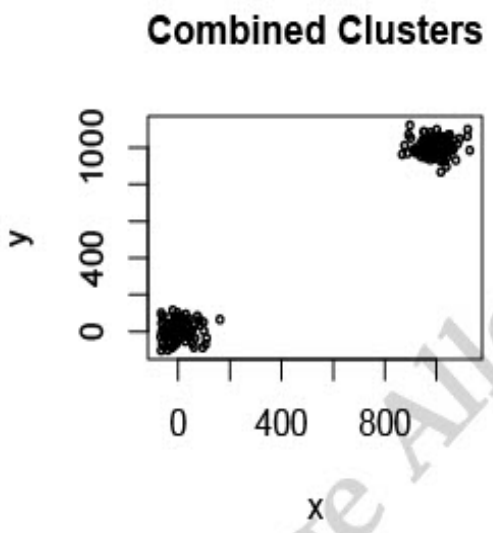
```
[1] 0.05597

plot(x_2, y_2, main = 'Cluster 2', cex = 0.5)
cor(x_2, y_2)  # No correlation between x and y in cluster 2

[1] -0.006664

x <- c(x_1, x_2)  # Combine/concatenate the 2 clusters.
y <- c(y_1, y_2)
plot(x, y, main = 'Combined Clusters', cex = 0.5)
cor(x, y)  # r incorrectly sees a correlation between x and y.

[1] 0.991
```





No Reproduction or Storage Allowed





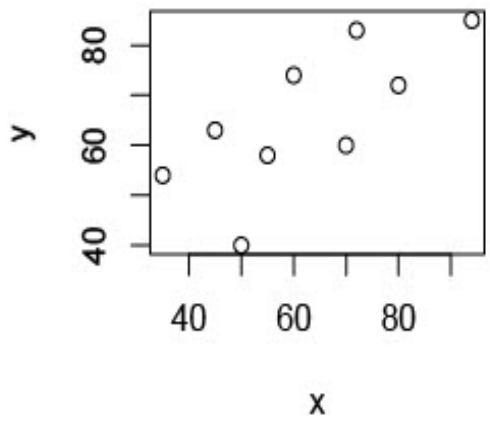
The moral is this: Use r to measure linear correlation, but always examine the data (e.g. with a scatterplot) to make sure things are okay.

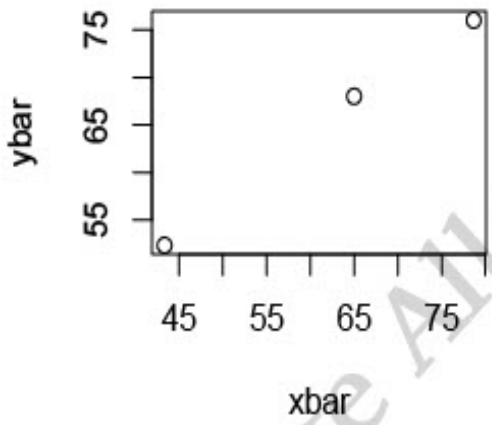
Example: Ecological Correlation

The following example illustrates another way in which the value of r can be "artificially" increased, i.e., by averaging over things before computing r. For similar reasons, regression results (disscussed in later sections) can be misleading as well.

```
dat <- read.table('3_17_dat.txt', header = TRUE)
x <- dat[, 1]
y <- dat[, 2]
z <- dat[, 3]

plot(x, y)  # Making a scatter plot.
cor(x, y)  # Moderate correlation of 0.733 between the 9 pairs.

[1] 0.7329

xbar <- numeric(3)  # Allocating space for storing the time-averaged values of x.
ybar <- numeric(3)  # and of y.

xbar[1] <- mean(x[z == 1])  # This averages x values only when time = 1.
ybar[1] <- mean(y[z == 1])

xbar[2] <-  mean(x[z == 2])  # USE UP-ARROW.
ybar[2] <-  mean(y[z == 2])

xbar[3] <-  mean(x[z == 3])
ybar[3] <-  mean(y[z == 3])

plot(xbar,ybar)  # Scatterplot of the 3 averaged pairs,
cor(xbar,ybar)  # and their extreme correlation of 0.998 .

[1] 0.9985
```

Not for storage Allowed

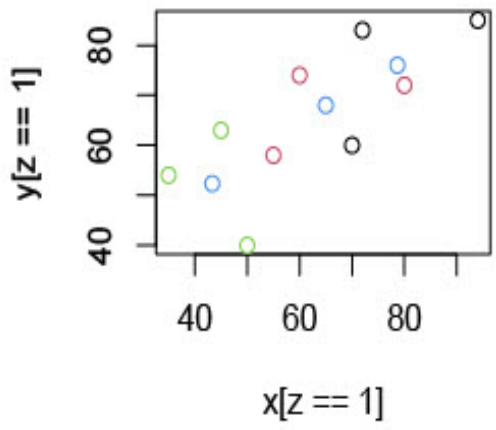




You can see clearly how it is that averaging tends to increase r, by reducing the number of points and their scatter about a line. Looking at the last scatterplot of the original data, but with the three times colored differently, you can see why this magnification of r is happening: Averaging the three pairs for each time, replaces the three points with a single point located in the "middle" of the three. In general, then, averaging tends to reduce the scatter, and hence the resulting r (called the ecological correlation):

```
plot(x[z == 1], y[z == 1], xlim = range(x), ylim = range(y))  # Scatterplot for time 1
points(x[z == 2], y[z == 2], col = 2)  # time 2 (USE UP-ARROW)
points(x[z == 3], y[z == 3], col = 3)  # time 3
points(xbar, ybar, col = 4)  # and the averaged data.

pdf("ecol.pdf")
plot(x[z == 1], y[z == 1], xlim = range(x), ylim = range(y), xlab = "x",
     ylab = "y", pch = 1, cex = 3)
points(x[z == 2], y[z == 2], col = 1, pch = 2, cex = 3)
points(x[z == 3], y[z == 3], col = 1, pch = 3, cex = 3)
dev.off()

pdf
  2
```

No Repr... Storage Allowed



3.3 OLS Regression on Simulated Data

Regression (or a line that fits the scatterplot) can be used for prediction. The function lm(), which stands for linear model, does runs a regression in R. It fits a curve through a scatterplot, or a surface through higher-dimensional data.

```
rm(list = ls(all = TRUE)) # Start from a clean slate.
set.seed(123)  # Ensures reproducable results.
x <- runif(100, 0, 1)  # x is uniform between 0 and 1.
error <- rnorm(100, 0, 1)  # Error is normal with mean = 0, sigma = 1.
y <- 10 + 2*x + error  # The real/true line is y = 10 + 2x.
plot(x, y)  # Plot the scatterplot.
cor(x, y)  # Correlation between x and y.
```

```
[1] 0.4916
```

```
model.1 <- lm(y ~ x)  # Fitting the regression.
model.1  # Note that the estimated coefficients are pretty close to the true ones
```

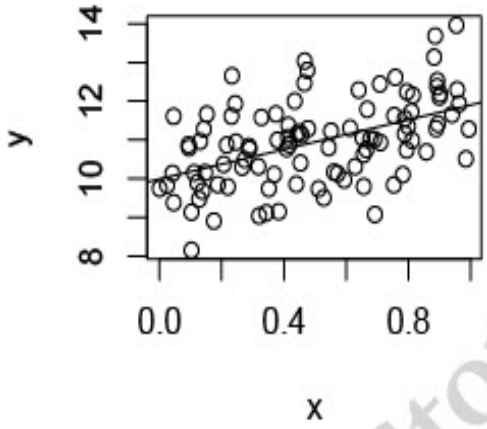
```

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
       9.99         1.91
```

```
abline(model.1)  # Superimposes the fit on the scatterplot.

# To see what else is returned by lm(), use the following command:
names(model.1)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

Storage Allowed

```
# To select one of the items returned in lm(), use the dollar sign:
model.1$coefficients

(Intercept)           x
      9.991       1.910
```

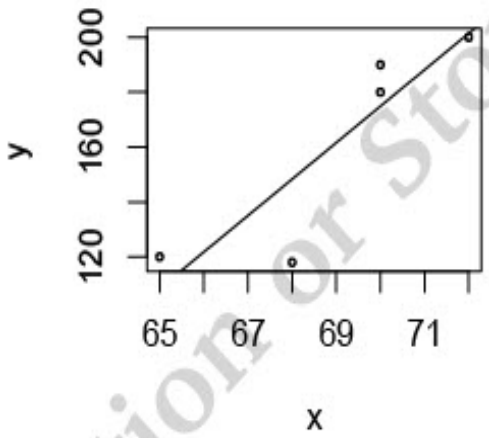


3.4 OLS Regression on "Real" Data

```
x <- c(72, 70, 65, 68, 70)  # Enter data into R.
y <- c(200, 180, 120, 118, 190)  # See 1.1 for alternative ways to enter data.
plot(x, y, cex = 0.5)
cor(x, y)

[1] 0.8892

model.1 <- lm(y ~ x)
abline(model.1)  # Draws the fit
model.1  # Returns the estimated intercept and slope.


Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)           x
     -755.1         13.3

summary(model.1)


Call:
lm(formula = y ~ x)
```

Not for Storage Allowed

```
Residuals:
     1      2      3      4      5
 -1.46   5.11  11.54 -30.31  15.11

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -755.11     272.53   -2.77    0.070 .
x              13.29       3.95    3.37    0.044 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 20.9 on 3 degrees of freedom
Multiple R-squared:  0.791,Adjusted R-squared:  0.721
F-statistic: 11.3 on 1 and 3 DF,  p-value: 0.0436
```



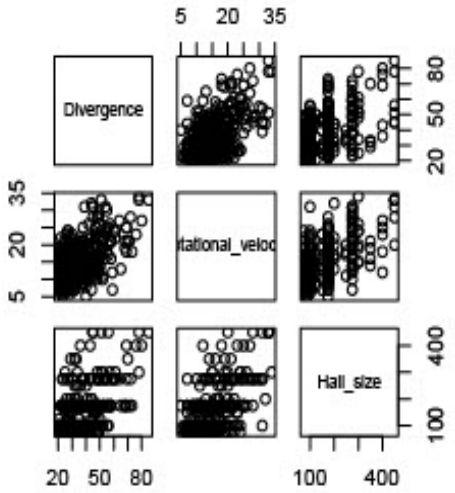
Example: Regression on Hail Data

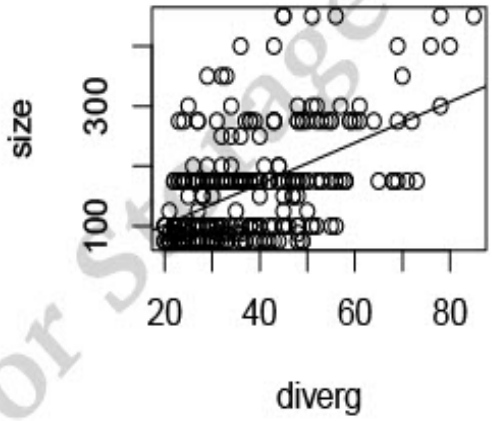
In practice, two quantities called "divergence" and "rotate" are measured by Doppler radar, while hail size is measured directly, i.e., on the ground. But if we can relate hail size to divergence and rotate, then we can predict hail size from Doppler radar. In regression lingo, size is the response (or dependent) variable, and the others are predictors (or independent variables, or covariates).

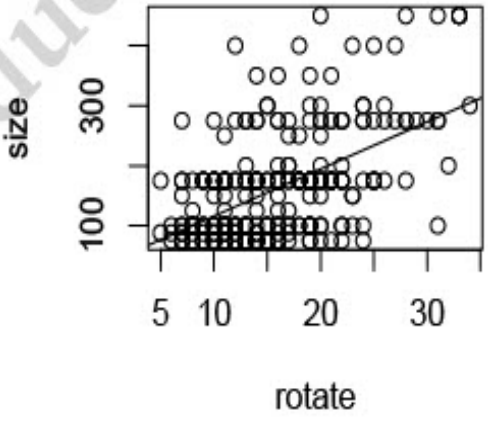
```
dat <- read.table("hail_dat.txt", header=T)

plot(dat)
cor(dat)  # This shows the correlations between ALL the vars in the hail data

                    Divergence Rotational_velocity Hail_size
Divergence              1.0000              0.5496    0.5214
Rotational_velocity     0.5496              1.0000    0.5386
Hail_size               0.5214              0.5386    1.0000

size <- dat[, 3]  # Name the 3 columns in dat. Size is in 100th-of-an-inch.
```

No reproduction or Storage allowed

```
rotate <- dat[, 2]
diverg <- dat[, 1]

model.1 <- lm(size ~ diverg)  # Regression of size and divergence.
plot(diverg, size)  # Viewing the scatterplot.
abline(model.1)  # Viewing the regression line.

model.2 <- lm(size ~ rotate)  # Regression on size and rotation.
plot(rotate, size)
abline(model.2)
```







Note that it looks like the line is not really going "through" the data; it seems like the line's slope should be larger. The fit is in fact correct. The line that intuitively (or visually) goes "through" the scatterplot is NOT the regression line, but something else called the "sd line."

No Reproduction or Storage Allowed

3.5 Analysis of Variance (ANOVA) in Regression

ANOVA decomposes SST (total sum of squares) into $SS_{explained}$ and $SS_{unexplained}$ (SSE).

$$SS_{explained} = \sum_{i=1}^{n} (\hat{y}_i - \overline{y})^2 \tag{1}$$

$$SS_{unexplained} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2}$$

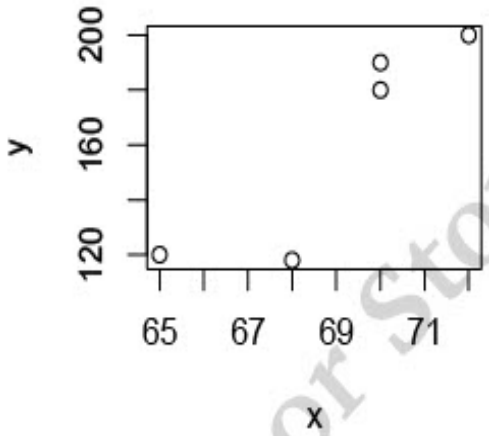
$$SST = \sum_{i=1}^{n} (y_i - \overline{y})^2 \tag{3}$$

$SS_{explained}$ is converted to a proportion called R-squared (a.k.a. coefficient of determination). It measures the proportion of the variability in y that is explained by x. It's a measure of goodness-of-fit.

```
x <- c(72, 70, 65, 68, 70) # Enter data into R.
y <- c(200, 180, 120, 118, 190) # See 1.1 for alternative ways to enter data.
plot(x, y)  # Plot the scatterplot.
cor(x, y)  # Correlation between x and y.

[1] 0.8892

model.1 <- lm(y ~ x)  # Fitting the regression.
anova(model.1)  # Note that SS_explained = 4942 and SSE = 1309

Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
x          1   4942    4942    11.3  0.044 *
Residuals  3   1309     436
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

summary(model.1)  # R-squared = 0.7906


Call:
lm(formula = y ~ x)

Residuals:
     1      2      3      4      5
 -1.46   5.11  11.54 -30.31  15.11

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -755.11     272.53   -2.77    0.070 .
x              13.29       3.95    3.37    0.044 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Not for Sale, Allowed

```
Residual standard error: 20.9 on 3 degrees of freedom
Multiple R-squared:  0.791,Adjusted R-squared:  0.721
F-statistic: 11.3 on 1 and 3 DF,  p-value: 0.0436

# The R^2 is reported as "Multiple R-squared".

# Note that the R-squared from summary() agrees with 1-(SSE/SST):
1 - (1308.9 / (4942.3 + 1308.9))

[1] 0.7906
```



$SS_{unexplained}$ (SSE) is converted to a standard deviation (of errors), and denoted as se. This standard deviation of errors is also called standard deviation about regression. Either way it is reported as "Residual standard error: 20.9". Note that it is equal to $\sqrt{\frac{SSE}{n-2}}$:

```
sqrt(1308.9 / (5 - 2))

[1] 20.89
```

In sum, R-squared and se together tell you how good the model is. R-squared tells you what percent of the variance in y can be attributed to x, and se tells you the typical error, i.e., deviation of data from the line.

To get R^2 (and nothing else), use the following command:

```
summary(model.1)$r.squared

[1] 0.7906

# Do the following to check that R's SSE really is Sum of Squared Errors:
y_hat <- predict(model.1)  # This is a quick way of getting y_hat.
y_hat  # To see the predictions.

    1     2     3     4     5
201.5 174.9 108.5 148.3 174.9
```

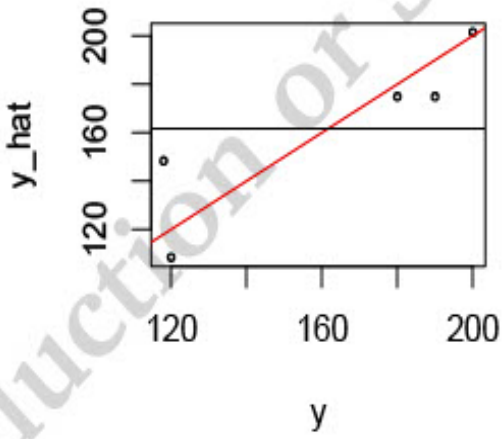
Non-reproduction or Storage Allowed

```
sum((y - y_hat) ^ 2)  # 1308.914 = SSE above.

[1] 1309
```

3.6 Visual Assessment of Goodness-of-Fit

One way to access the goodness of fit is to examine the scatterplot of predicted y versus actual y.

```
y_hat <- predict(model.1)

# Alternatively, you can use the predictions stored in lm() itself:
y_hat <- model.1$fitted.values

# Here is the scatterplot of predicted size vs. actual size:
plot(y, y_hat, cex = 0.5)

abline(0, 1, col = "red")  # Add a diagonal line.
abline(h = mean(y))  # Add a horizontal line at the mean of y (i.e., size).
```



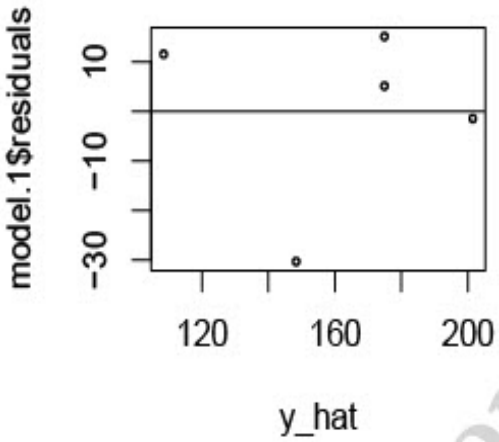
If the model were good, this scatterplot would be symmetrically spread about the red line. But, clearly our model is not good. This scatterplot (of predicted vs. actual) is often a great way of visualizing how well the model is doing. For example, we see that for smaller hail size (i.e., small x value), the predictions of size are all above the diagonal indicating that the model over-predicts the size of small hail. Looking at larger x values, it's clear that the model under-predicts the size of large hail.

If a model is completely useless, then the predictions will be symmetrically spread about the horizontal line at the mean of y. Here, we can see that our model is nearly (but not completely) useless. This kind of model diagnosis can help in coming up with a better model.

Another visual assessment tool is the residual plot. This plot checks different facet of "goodness" (or quality) than the above plot.

No Reproduction or Storage Allowed

```
# The residuals are already contained in what lm() returns.
plot(y_hat, model.1$residuals, cex = 0.5)
abline(h = 0)
```



In a good fit, these residuals (or errors) should NOT display any relationship with the predicted values. One way to confirm that there is no relationship is to compute the correlation:

```
cor(y_hat, model.1$residuals)

[1] -1.237e-16
```

The fact that the correlation is zero is not a direct reflection of the goodness of fit, because that correlation is zero by construction. If it's not zero, one has a bug! In fact, it is the identically-zero nature of this correlation which makes a plot of the residuals vs. $\hat{y}$ a useful plot to examine. The correlation between the residuals and the observed y values is not identically zero; and for that reason the corresponding scatterplot is not readily interpretable.

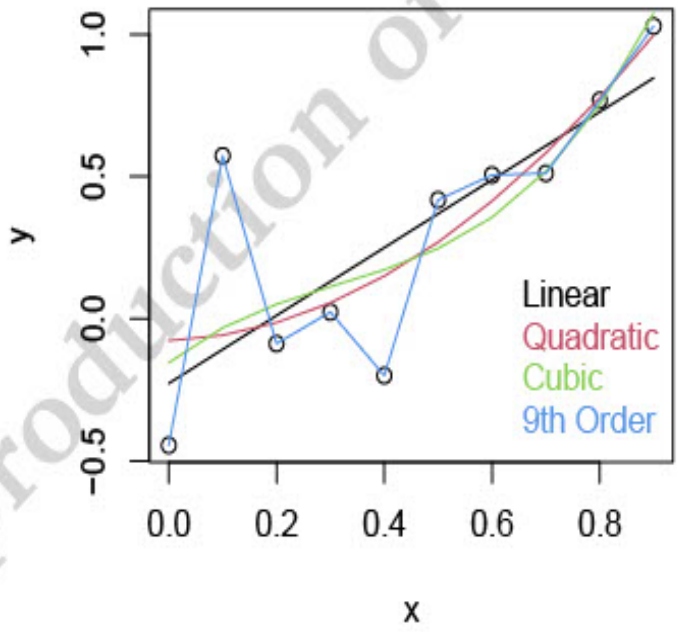
3.7 Nonlinear Fits (Linear Regression with Higher Order Terms)

Linear regression is actually NOT linear when it comes to allowing nonlinear relationships between x and y. (The term "linear" refers to the parameters of the model, i.e., the regression coefficients.) This is good news, because linear regression can fit any nonlinear data. But it's also bad news, because the ability to fit nonlinear data also allows for overfitting. In developing regression models of data it is important to assure that the model is not overfitting the data, because such a model will have poor *predictive* capability. Toward the end of this book we will see how to assess the predictive capability of a regression model. Here, let's first confirm that linear regression *can* overfit (memorize) data.

```
set.seed(12)  # Set a seed to ensure reproducable results.
x <- seq(0, 0.9, 0.1)  # Pick 10 x's between 0 and 1.
y <- x + rnorm(10, 0, 0.3)  # x and y are "truly" linear, plus error.
plot(x,y)  # Look at the data.

lm.1 <- lm(y ~ x)  # Fit the simplest regression model
```

No Reproduction - Storage Allowed

```
lines(x, lm.1$fitted.values)

lm.2 <- lm(y ~ x + I(x^2))  # Fit a regression model including the quadratic term.
lines(x, lm.2$fitted.values, col = 2) # The I() is necessary. Don't ask why!

lm.3 <- lm(y ~ x + I(x^2) + I(x^3))  # Add a cubic term.
lines(x, lm.3$fitted.values, col = 3)  # Note that the fit is getting more curvy.

lm.4 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8)
           + I(x^9))
# Fit a 9th order polynomial.
lines(x, lm.4$fitted.values, col = 4)
summary(lm.4)$r.squared  # Examine the R-squared.

[1] 1

legend('bottomright', c('Linear', 'Quadratic', 'Cubic', '9th Order'),
       text.col = c(1, 2, 3, 4), bty = 'n')

# Note that the last model will have no predictive power since it overfits the data.
```



No Reproduction or Stor ved

3.8 Model Comparison

Example: Hail Data

Note that the closer R^2 is to 1, the "better" the fit and the closer it is to 0, the worse. But do recall that because of overfitting concerns, higher R^2 does not necessarily mean better predictions on

new/future data.

```
dat <- read.table("hail_dat.txt", header = T)

x_1 <- dat[, 1]  # Divergence.
x_2 <- dat[, 2]  # Rotate.
y <- dat[, 3]  # Hail size. Size is in 100th-of-an-inch.
# Renaming the columns in dat:
colnames(dat) <- c("x_1", "x_2", "y")

lm.1 <- lm(y ~ x_1)  # Predicting size from divergence (simple regression).
summary(lm.1)


Call:
lm(formula = y ~ x_1)

Residuals:
   Min     1Q Median     3Q    Max
-126.1  -50.9  -19.8   44.8  262.6

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   33.673     13.361    2.52    0.012 *
x_1            3.417      0.334   10.23   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 76 on 280 degrees of freedom
Multiple R-squared:  0.272,Adjusted R-squared:  0.269
F-statistic:  105 on 1 and 280 DF,  p-value: <2e-16

lm.2 <- lm(y ~ x_2)  # Predicting size from rotation (simple regression).
summary(lm.2)


Call:
lm(formula = y ~ x_2)

Residuals:
   Min     1Q Median     3Q    Max
-180.9  -55.1  -11.6   36.6  268.4

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   37.268     12.508    2.98   0.0031 **
x_2            7.858      0.735   10.70   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 75 on 280 degrees of freedom
Multiple R-squared:  0.29,Adjusted R-squared:  0.288
F-statistic:  114 on 1 and 280 DF,  p-value: <2e-16
```

Not Reviewed

```r
lm.3 <- lm(y ~ x_1 + x_2) # Predicting size from both (multiple regression).
summary(lm.3)
```

```
Call:
lm(formula = y ~ x_1 + x_2)

Residuals:
   Min     1Q Median     3Q    Max
-157.4  -52.0  -12.2   35.5  261.8

Coefficients:
            Estimate Std. Error t value     Pr(>|t|)
(Intercept)   -1.179     13.684   -0.09         0.93
x_1            2.117      0.375    5.65 0.0000000401 ***
x_2            5.268      0.835    6.31 0.0000000011 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 71.2 on 279 degrees of freedom
Multiple R-squared:  0.363,Adjusted R-squared:  0.358
F-statistic: 79.5 on 2 and 279 DF,  p-value: <2e-16
```

```r
lm.4 <- lm(y ~ x_1 + x_2 + x_1:x_2)  # Multiple regression with interaction.
summary(lm.4)
```

```
Call:
lm(formula = y ~ x_1 + x_2 + x_1:x_2)

Residuals:
   Min     1Q Median     3Q    Max
-157.1  -50.8  -14.2   36.6  264.0

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  71.6091    34.7693    2.06    0.040 *
x_1           0.1989     0.9217    0.22    0.829
x_2           1.0612     2.0269    0.52    0.601
x_1:x_2       0.1030     0.0453    2.27    0.024 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 70.7 on 278 degrees of freedom
Multiple R-squared:  0.375,Adjusted R-squared:  0.368
F-statistic: 55.5 on 3 and 278 DF,  p-value: <2e-16
```

```r
lm.5 <- lm(y ~ x_1 + x_2 + I(x_1 ^ 2) + I(x_2 ^ 2)) # Multiple quadratic regression.
summary(lm.5)
```

```
Call:
```

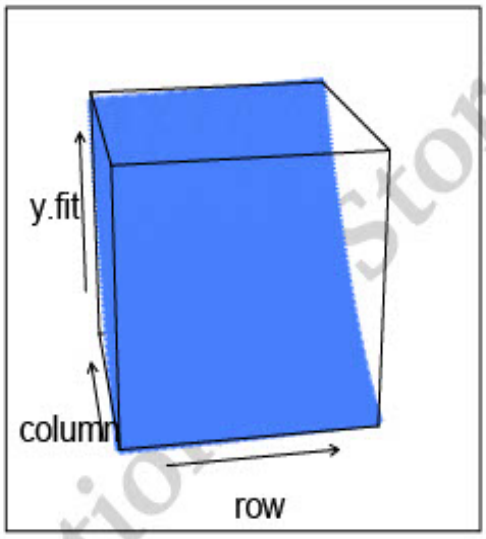
```
lm(formula = y ~ x_1 + x_2 + I(x_1^2) + I(x_2^2))

Residuals:
   Min     1Q Median     3Q    Max
-180.8  -48.6  -16.1   38.1  266.0

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  88.4961    38.1671    2.32    0.021 *
x_1           0.4789     1.6626    0.29    0.774
x_2          -1.9992     3.3297   -0.60    0.549
I(x_1^2)      0.0175     0.0186    0.94    0.346
I(x_2^2)      0.2053     0.0918    2.24    0.026 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 70.5 on 277 degrees of freedom
Multiple R-squared:  0.38,Adjusted R-squared:  0.371
F-statistic: 42.4 on 4 and 277 DF,  p-value: <2e-16
```

```
lm.6 <- lm(y ~ x_1 + x_2 + I(x_1 ^ 2) + I(x_2 ^ 2) + x_1:x_2)
summary(lm.6)
```

```
Call:
lm(formula = y ~ x_1 + x_2 + I(x_1^2) + I(x_2^2) + x_1:x_2)

Residuals:
   Min     1Q Median     3Q    Max
-179.7  -48.9  -16.1   38.0  265.7

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  89.0739    38.4006    2.32    0.021 *
x_1           0.4974     1.6695    0.30    0.766
x_2          -2.0870     3.3794   -0.62    0.537
I(x_1^2)      0.0145     0.0264    0.55    0.584
I(x_2^2)      0.1921     0.1228    1.56    0.119
x_1:x_2       0.0137     0.0843    0.16    0.872
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 70.6 on 276 degrees of freedom
Multiple R-squared:  0.38,Adjusted R-squared:  0.369
F-statistic: 33.8 on 5 and 276 DF,  p-value: <2e-16
```

```
# Plotting a surface that goes through the cloud of points in 3d.
# Suppose we decided that the best model is the most complex model, above:
lm.e <- lm(size ~ diverg + rotate + I(diverg^2) + I(rotate^2) + I(diverg * rotate))

x <- seq(min(rotate), max(rotate), length = 100)  # x and y simply define a
y <- seq(min(diverg), max(diverg), length = 100)  # grid in the x-y plane.
```

```r
f <- function(x, y) {
  r <- lm.e$coeff[1] + lm.e$coeff[2] * x + lm.e$coeff[3]*y
  + lm.e$coeff[4] * x ^ 2 + lm.e$coeff[5] * y ^ 2 + lm.e$coeff[6] * x * y
}
y.fit <- outer(x, y, f)  # y contains the "height" values of the surface,
                         # over the x-y grid; that's what outer() does. Look it up.

library(lattice)  # Loading the library that contains the function cloud().
# Making a 3d plot of the points of the plane.
cloud(y.fit, type = "p", screen = list(z = 10, x = -70, y = 0))

# Note that in spite of the nonlinearity of the regression function
# itself, i.e. with quadratic and an interaction terms, the surface is
# mostly planar in the range of our data (i.e., x and y).
```



Here is a discussion of all of the above results based on the R^2 values: It seems like

1. Rotate is a better predictor of size than diverge.
2. The two of them together make for an even better model.
3. Quadratic terms for each, make the model even better, but not by much
4. R^2 goes from 0.3629 to 0.3800.
5. An interaction term, without quadratic terms, gives a model that is comparable to what we got from a quadratic model with no interaction.
6. Quadratic and interaction terms, together, "seem" to give the best model.

Note that R^2 increases as the complexity of the model increases (adding more terms). The main question (which can be addressed only qualitatively at this point) is this: is the gain in R^2 big enough to warrant the new term, knowing that a new term can lead to over-fitting. In this example, the gain from $R^2 = 0.3629$ to $R^2 = 0.3800$ is probably NOT worth the risk of overfitting. So, we should keep the simpler model. That's called the principle of "Occam's Razor," which posits that one should go with simpler things.

No Reproduction or Storage Allowed

```
anova(lm.6)

Analysis of Variance Table

Response: y
           Df  Sum Sq Mean Sq F value        Pr(>F)
x_1         1  603829  603829  121.05       < 2e-16 ***
x_2         1  202064  202064   40.51 0.00000000081 ***
I(x_1^2)    1   13086   13086    2.62         0.106
I(x_2^2)    1   24838   24838    4.98         0.026 *
x_1:x_2     1     131     131    0.03         0.872
Residuals 276 1376791    4988
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the anova() output, there is an SS term for **each term** in the regression equation, followed by an SS term for "Residuals." The latter is what we have been calling SSE (or $SS_{unexplained}$); so, where is $SS_{explained}$? It can be obtained by adding the other SS terms. The reason R produces separate terms for each term is because R performs what is called **sequential analysis of variance**, which is a bit different from what we do. In fact, these SS terms will change depending on the order of the terms in the regression equation!

It's important to realize that all of these SS terms are measures of variability. Specifically, SST is the numerator of the sample variance of the y's, $SS_{explained}$ is the numerator of the sample variance of the **predictions**, and $SS_{unexplained}$ (SSE) is converted to variance when it's divided by $n-(k+1)$, where k is the number of parameters in the regression model. You can confirm these:

```
y_hat <- predict(lm.6) # From 9.3.
n <- nrow(dat)
(n - 1) * var(y_hat) # 843948 = 603829 + 202064 + 13086 + 24838 + 131 = SS_explained

[1] 843948
```

3.8.1 Prediction on New Data

The best way to do prediction on new data is to just attach the new data to the bottom of the old data. Suppose the new data consists of the following 2 case:

$$x_1 = 33,\ x_2 = 8$$
$$x_1 = 35,\ x_2 = 14$$

Then we can do the following:

```
n <- nrow(dat)  # number of cases in dat.
new_1 = c(33, 8, NA)  # y = NA because we don't know y for new data.
new_2 = c(35, 14, NA)
new.dat = rbind(dat, new_1, new_2) # Using row-bind to attach new data to old data.

# In the next line, we redevelop lm.4, but on the first n cases:
lm.7 <- lm(y ~ x_1 + x_2 + x_1:x_2, dat = new.dat[1:n, ])  # NOTE: dat=new.dat[1:n,].
summary(lm.7)  # Same as lm.4.
```

Non-reproductive Storage Reserved

```
Call:
lm(formula = y ~ x_1 + x_2 + x_1:x_2, data = new.dat[1:n, ])

Residuals:
   Min     1Q Median     3Q    Max
-157.1  -50.8  -14.2   36.6  264.0

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  71.6091    34.7693    2.06    0.040 *
x_1           0.1989     0.9217    0.22    0.829
x_2           1.0612     2.0269    0.52    0.601
x_1:x_2       0.1030     0.0453    2.27    0.024 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 70.7 on 278 degrees of freedom
Multiple R-squared:  0.375,Adjusted R-squared:  0.368
F-statistic: 55.5 on 3 and 278 DF,  p-value: <2e-16
```

```
colnames(new.dat) <- c("x_1", "x_2", "x_1:x_2")
predict(lm.7, newdata <- new.dat[(n+1):(n+2), ])  # Predict the last 2 cases.
```

```
  283   284
113.8 143.9
```

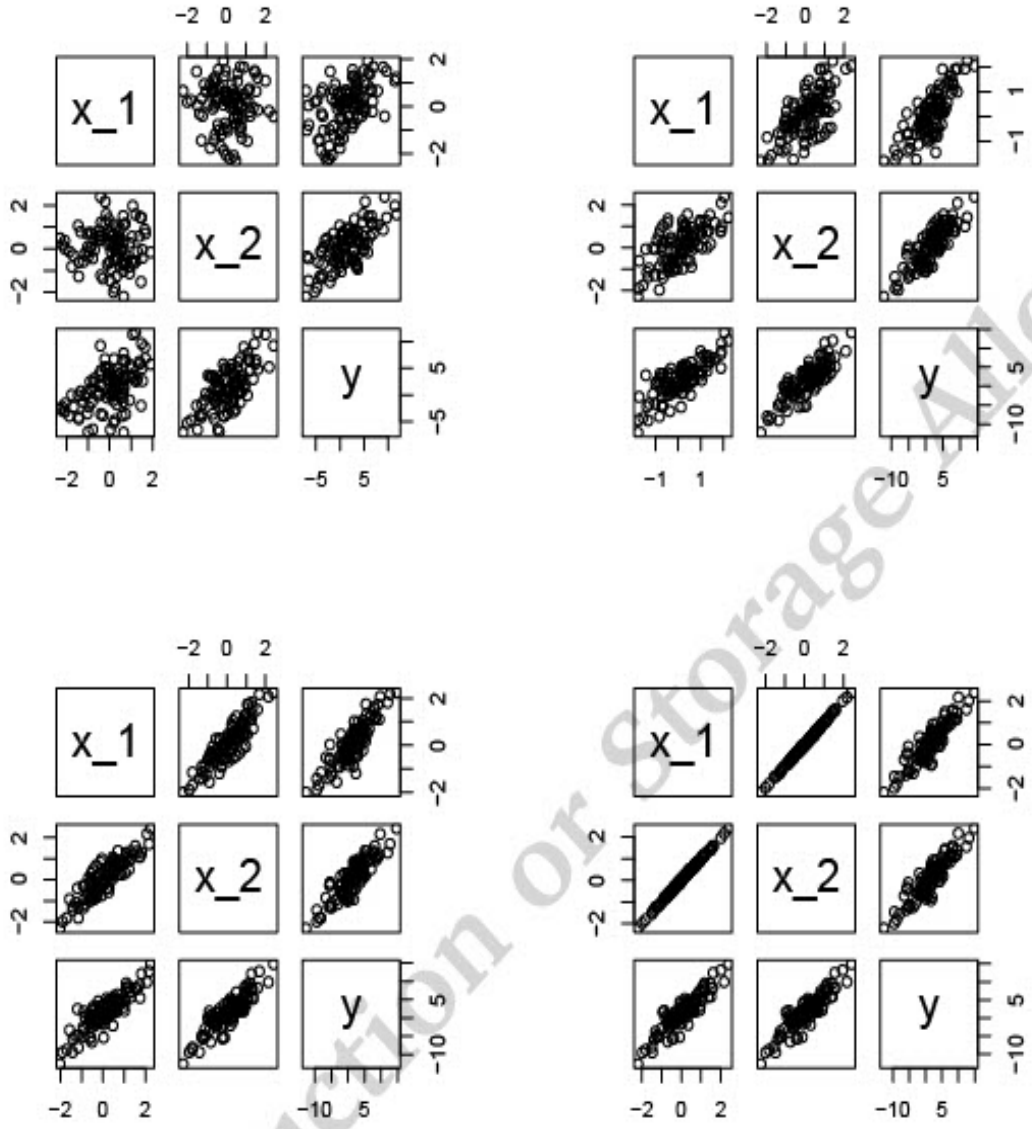
3.9 Collinearity

Another distressing issue that arises in **multiple** regression is collinearity, i.e., a linear association between the predictors themselves. One reason collinearity is distressing is that it renders the regression coefficients uninterpretable; i.e., a given beta can no longer be interpreted as the average rate of change of y with respect to a unit change in x with everything else held fixed. Insisting on that kind of interpretation, in the presence of collinearity, can lead to wrong (or even absurd) conclusions. Collinearity also makes the predictions more uncertain, but here we will focus on the effect of collinearity on the regression coefficients.

```
# To that end, we'll write an R function, which is nothing but some lines
# of code intended to be used over and over again.
make.fit <- function(r) {
# The function first makes data on x_1, x_2, and y, with collinearity
# (i.e., correlation between x_1 and x_2) equal to r.
# The input of the function is r (i.e., correlation between x_1 and x_2.
# NOT between y and anything).
# The function then fits that data using y, and returns some stats about
# the estimated regression coefficients.
  library(MASS)  # This library contains mvrnorm(); see below.
  set.seed(1)  # Ensures reproducible outputs.
  n <- 100
  # The R function mvrnorm() below takes a sample from a multivariate normal,
  # which is a higher-dimensional analog of the normal distribution.
  dat <-  mvrnorm(n, rep(0, 2), matrix(c(1, r, r, 1), 2, 2))
```

```
  x_1 <- dat[, 1]
  x_2 <- dat[, 2]
  y <- 1 + 2 * x_1 + 3 * x_2 + rnorm(n, 0,2)  # Generate y, and add noise.
  dat <- data.frame(x_1, x_2, y)  # Here is the whole data.
  plot(dat)
  lm.1 <- lm( y ~ x_1 + x_2)  # Fit a plane through the data.
  # return(lm.1) returns the whole R object lm.1.
  # return(summary(lm.1)) returns only the summary results.
  return(summary(lm.1)$coeff)  # Returns only the regression coefficients.
  }

# Examining data and the regression coefficients for different amounts of
# collinearity.
make.fit(0)  # No collinearity.

            Estimate Std. Error t value  Pr(>|t|)
(Intercept)    1.051     0.2104   4.994 2.609e-06
x_1            2.107     0.2190   9.623 8.760e-16
x_2            3.042     0.2335  13.028 4.968e-23

make.fit(0.7)  # Some collinearity.

            Estimate Std. Error t value  Pr(>|t|)
(Intercept)    1.051     0.2104   4.994 2.609e-06
x_1            2.161     0.3096   6.979 3.684e-10
x_2            2.885     0.3099   9.310 4.141e-15

make.fit(0.9)   # Extreme collinearity.

            Estimate Std. Error t value     Pr(>|t|)
(Intercept)    1.051     0.2104   4.994 0.0000026091
x_1            2.261     0.5039   4.486 0.0000199246
x_2            2.783     0.5042   5.519 0.0000002837

make.fit(0.999)

            Estimate Std. Error t value    Pr(>|t|)
(Intercept)    1.051     0.2104  4.9941 0.000002609
x_1            4.412     4.8973  0.9009 0.369846805
x_2            0.630     4.8975  0.1286 0.897910878
```

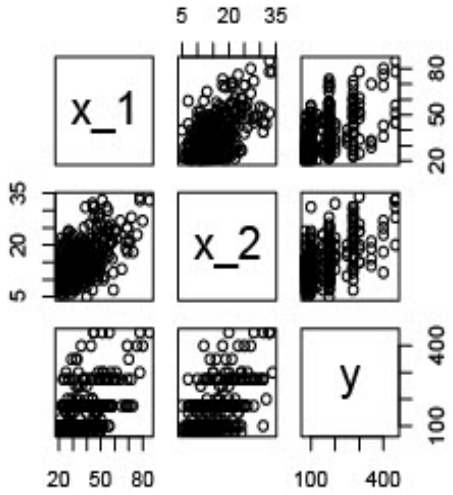
No Repr ved



When collinearity is extreme, not only are the standard errors huge, but the estimated regression coefficients themselves (β) are way off. As collinearity increases, the regression coefficients become more uncertain, and so we are unable to interpret them, like we would if there were no collinearity. The regression equation is still OK to use for predictions. But, of course, the predictions will be less certain as well. Note that in practice we don't control/adjust the data or the collinearity; all we see are the scatterplots, and based on the scatterplots between the predictors, we decide how much collinearity there is. For example, for the hail data:

```
plot(dat)
```

No Reproduction or Storage Allowed



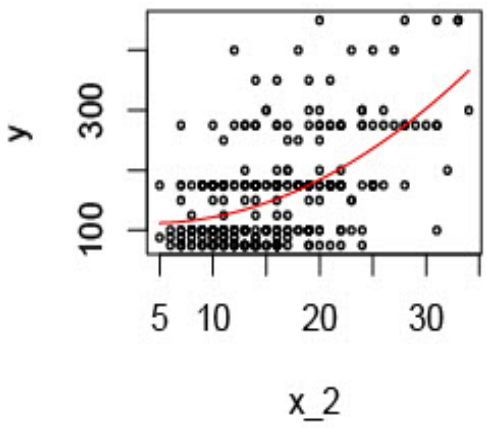
In the scatterplots, the one for x_1 versus x_2 (or vice versa) suggests some collinearity; it is not extreme, but it is present. As such, we have to be cautious in interpreting the regression coefficients. But the regression model is still okay for making predictions (as long as it does not overfit, of course).

3.10 Plotting Curved Fits on a Scatterplot

We plotted polynomial fits, but the "curves" were just the result of connecting points with straight lines, and as a result, the "curves" did not look smooth. Here is a way to get a smoother looking fit on the scatterplot.

```
# Suppose we pick a relatively simple quadratic model for the hail data:
dat <- read.table("hail_dat.txt", header = T)

x_1 <- dat[, 1] # Divergence.
x_2 <- dat[, 2] # Rotate.
y <- dat[, 3] # Hail size. Size is in 100th-of-an-inch.
lm.g <- lm(y ~ x_2 + I(x_2 ^ 2))
lm.g$coef  # Examine the regression coeffs.

(Intercept)         x_2    I(x_2^2)
   116.3519     -2.2682      0.2827

x <- seq(min(x_2), max(x_2), .01)  # Generate a fake x.
y.fit <- lm.g$coeff[1] + lm.g$coeff[2] * x + lm.g$coeff[3] * x^2
plot(x_2, y, cex = 0.5)
points(x , y.fit, col = "red", type = "l")
```

No AI Storage Allowed




```
# Alternatively, a fancier way is as follows.
x <- matrix(seq(min(x_2), max(x_2), .01), byrow = T)  # Generate a fake x .
colnames(x) <- "x_2"
plot(x_2, y)
lines(x, predict(lm.g, newdata = data.frame(x)), col=2)
```

No Reproduction or Usage Allowed

4 Sampling Distributions of the Sample Mean and Median

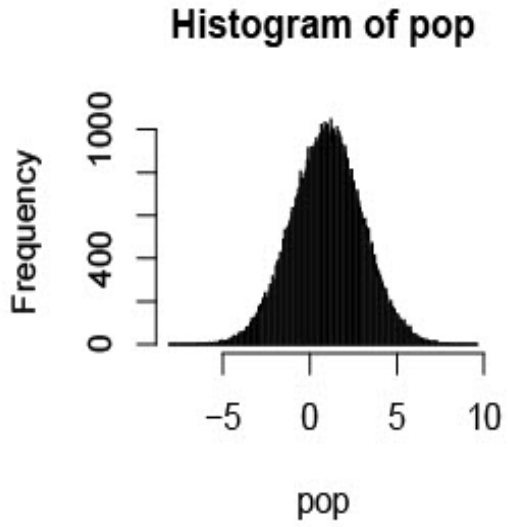
4.1 Sampling Distribution of the Mean

4.1.1 Normal Population

Instead of taking samples from a normal population, using rnorm(), we are going to take ONE huge sample from a normal population, using rnorm(), and then just treat it as our population. The main reason for this is mostly to set the stage for something called "bootstrapping," which we will study later.

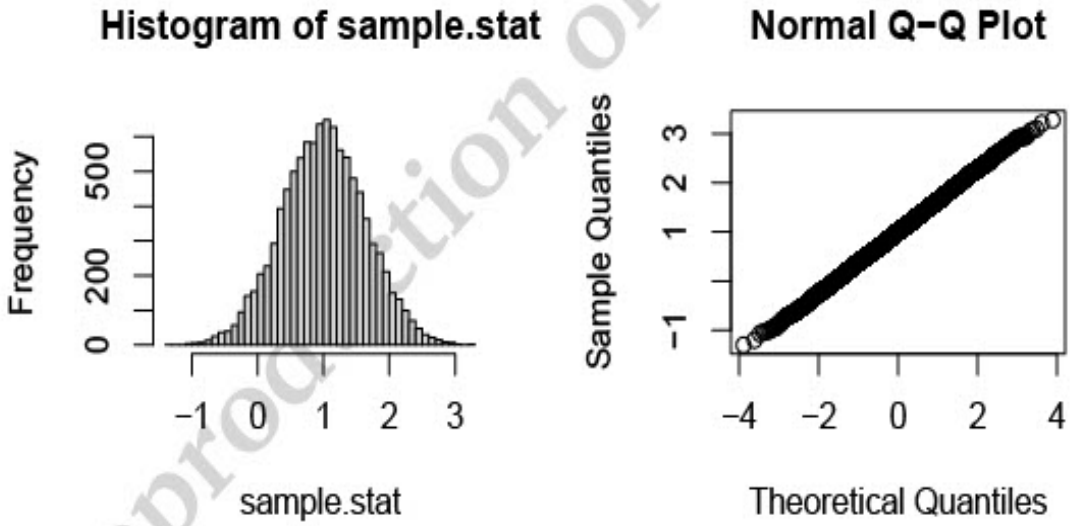
```
N <- 100000  # Let N be the population size.
pop <- rnorm(N, 1, 2)  # Take a random sample and treat it as pop.
pop.mean <- mean(pop)  # This is mu, the population mean.
pop.sd <- sd(pop)  # This is sigma, the pop standard deviation.
pop.median <- median(pop)  # Get the population median, for later.
c(pop.mean, pop.sd, pop.median)  # Print them for comparison, below.

[1] 0.9958 2.0070 1.0022

hist(pop, breaks = 400)  # This shows that the population is pretty normal.

# Experiment underlying the sampling distribution.
n.trial <- 10000  # Take 10000 samples of
sample.size <- 10  # size 10 (i.e., small) from the population.
sample.stat <- numeric(n.trial)  # Create space to store the 10000 sample means.

for (i in 1:n.trial) {
  samp <- sample(pop, sample.size, replace = T)  # Take a sample (with replacement).
  sample.stat[i] <- mean(samp)  # Compute each sample's mean.
}
mean(sample.stat)  # Compare mean of sample means

[1] 1.002

pop.mean  # with the population mean.

[1] 0.9958

sd(sample.stat)  # Compare the standard deviation of sample means

[1] 0.6333

pop.sd  # with the pop standard deviation.

[1] 2.007

pop.sd / sqrt(sample.size)  # But compare with (pop std dev)/sqrt(n)

[1] 0.6347
```

Not Reviewed



According to the Central Limit Theorem (CLT), the sampling distribution of the sample mean should be normal. To confirm:

```
hist(sample.stat,breaks = 40)
qqnorm(sample.stat)  # Pretty normal.
```



As the sample size increases, the mean of the sample means gets pretty close to the population mean, and the standard deviation of the sample means gets pretty close to the $\frac{sd(pop)}{\sqrt{n}}$. So, the CLT is confirmed.

4.1.2 Non-normal Population

```
N <- 100000
pop <- rgamma(N, 1, 1)
```

No Reproduction or Storage Allowed

```r
pop.mean <- mean(pop)
pop.sd <- sd(pop)
pop.median <- median(pop)
c(pop.mean, pop.sd, pop.median)
```

```
[1] 0.9968 0.9952 0.6955
```

```r
hist(pop, breaks = 400)  # The distribution of sample means looks non-normal.

n.trial <- 10000  # Take 10000 samples of
sample.size <- 10  # size 10 (i.e., small) from the population.
sample.stat <- numeric(n.trial)  # Space for storing the 10000 sample means.

for (i in 1:n.trial) {
  samp <- sample(pop, sample.size, replace=T)  # Take a sample (with replacement).
  sample.stat[i] <- mean(samp)  # and compute each sample's mean.
}

mean(sample.stat)  # Compare mean of sample means with population mean.
```

```
[1] 0.9985
```

```r
pop.mean
```

```
[1] 0.9968
```

```r
sd(sample.stat)  # Compare the sd of sample means with population sd.
```

```
[1] 0.3143
```

```r
pop.sd
```

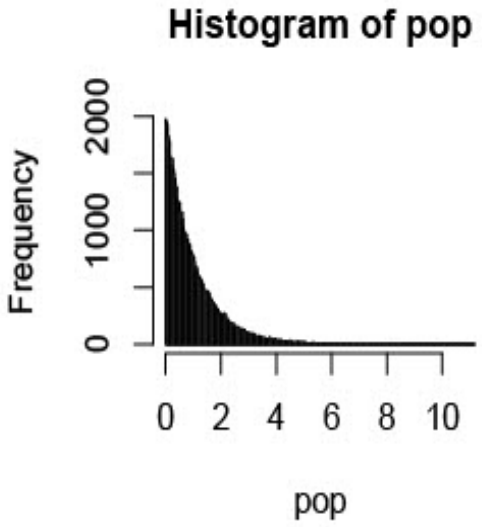
```
[1] 0.9952
```

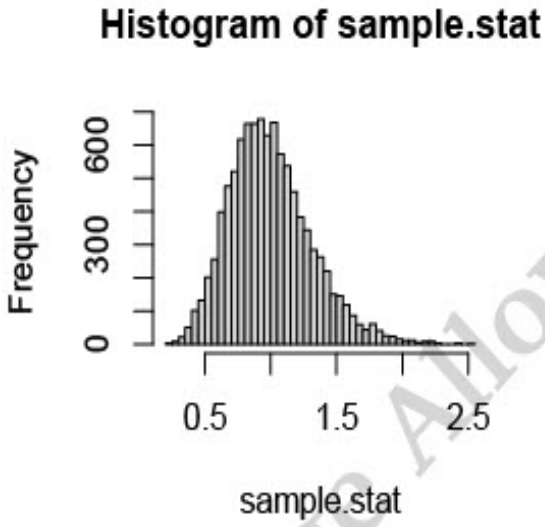
```r
pop.sd / sqrt(sample.size) # Compare with (pop sd)/root(n).
```

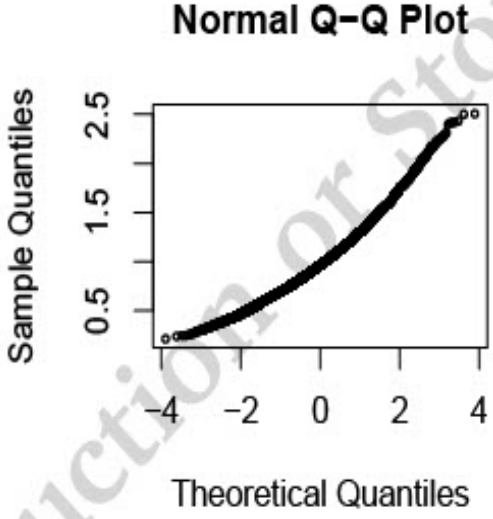
```
[1] 0.3147
```

```r
hist(sample.stat, breaks = 40)
qqnorm(sample.stat, cex = 0.5)
```

No Reproduced







When the population is NOT normal, for small samples (10) the sampling distribution of the sample mean resists looking normal; but with larger samples (100), it is normal even though the population is not normal.

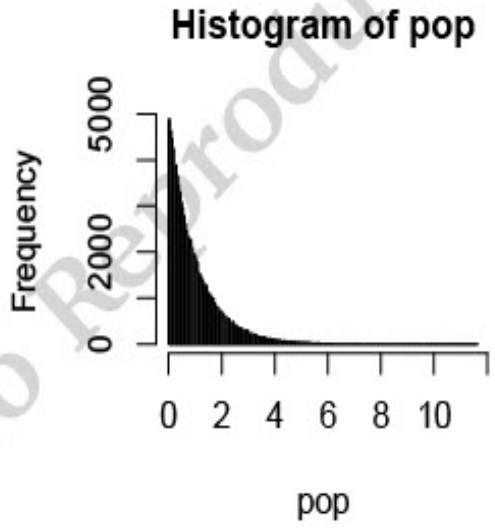
4.2 Sampling Distribution Population Median

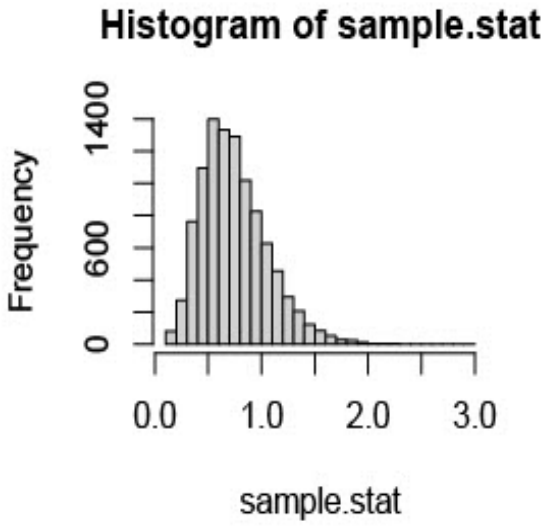
4.2.1 Non-normal Population

```
N <- 100000
pop <- rgamma(N,1,1)
pop.mean <- mean(pop)
pop.sd <- sd(pop)
pop.median <- median(pop)
c(pop.mean, pop.sd, pop.median)
```

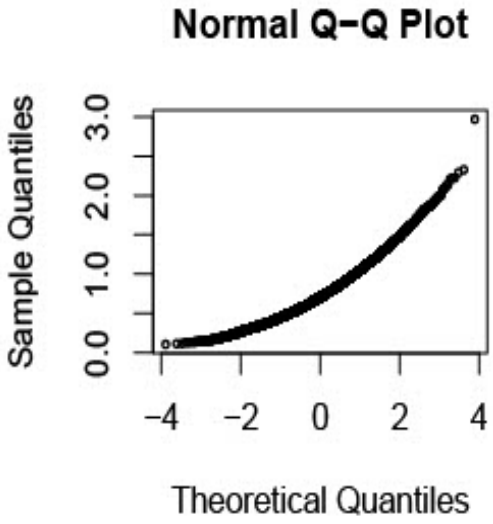
No Reproduction or Storage Allowed

```
[1] 1.0014 1.0009 0.6937

hist(pop, breaks = 400)

n.trial <- 10000  # Take 10000 samples of
sample.size <- 10  # size 10 (i.e., small) from the population.
sample.stat <- numeric(n.trial)  # Space for storing the 10000 sample medians.

for (i in 1:n.trial) {
  samp <- sample(pop, sample.size, replace = T)  # Take a sample (with replacement).
  sample.stat[i] <- median(samp)  # Compute each sample's MEDIAN.
}

mean(sample.stat)  # Compare the MEAN of sample MEDIANS with pop MEDIAN.

[1] 0.7469

pop.median

[1] 0.6937

sd(sample.stat)  # Compare the sd of sample MEDIANS with population sd.

[1] 0.3094

pop.sd

[1] 1.001

# Note that the formula sigma/root(n) applies only to the sample MEAN.

hist(sample.stat, breaks = 40)
qqnorm(sample.stat, cex = 0.5)
```





No Reproduction Allowed



The sampling distribution doesn't look too normal. But if the sample size is relatively large, the distribution of a bunch of sample medians, taken from even a non-normal population, is still normal. Most statistics (e.g., sample mean, sample median, sample standard deviation, ...) ultimately end up having a normal distribution, but some require a larger sample size.

No Reproduction or Storage Allowed

5 Confidence Interval

Now, we're going to move on from the sampling distribution, and develop the notion of a Confidence Intervals (CI). First, we will show that the formula for the CI for the population mean actually does what it is designed to do. Recall that the formula for the confidence interval (CI) for the population mean is given by:

$$\bar{x} \pm z^* \cdot \frac{\sigma}{\sqrt{n}} \tag{4}$$

and it is designed to cover the population mean in 95% of samples taken from the population. One nontrivial part of this formula is the $\frac{\sigma}{\sqrt{n}}$, also called the standard error (std err) of the sample mean. It's nontrivial because, first we have to approximate the population std (σ) with sample std, but more importantly, we have to use math to derive it. For many statistics (other than the sample mean), the std error is difficult to derive mathematically. The other nontrivial part of the CI formula is the z^*, because it is based on the fact that the sample mean has a normal distribution. Some statistics do not.

The second task is to show that we can actually get similar answers, WITHOUT using the formula for the std err of the mean, nor the assumption of normality. This is important when simple formulas for the std err do not exist, e.g., for sample median. The main idea is called The Bootstrap: We basically treat the single sample that we have in a realistic situation as if it were the population! So, instead of sampling from the population (i.e., what we did above), bootstrap re-samples from the *sample.* It's like magic, but you'll see how it works below.

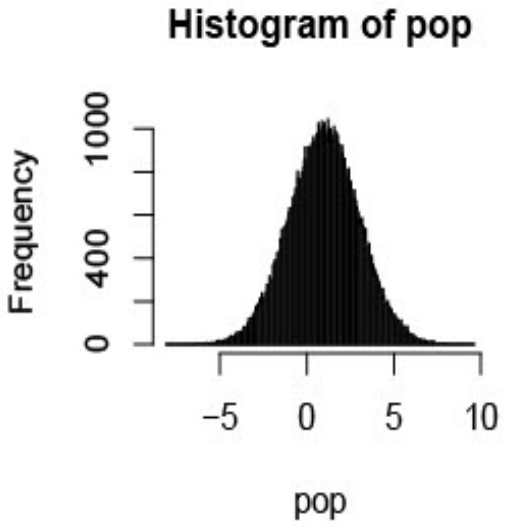
There are different kinds of CI, e.g., 1-sample, 2-sample, 1-sided, 2-sided, large-sample, small-sample, etc. And yet other kinds of CI will be covered as we proceed forward into chapters 8, 9, 10, and 11. In the following, you will also come across words like "p-value," or "hypothesis." For now, you may simply ignore them. Ch8 will introduce that method, which is equivalent to the CI method.

5.1 Confidence Interval for Population Mean

As explained at the start of the section on Sampling Distribution, instead of taking samples from a normal population, using rnorm(), we are going to take ONE huge sample from a normal population, using rnorm(), treat it as our population, and then use the R function sample() to take samples from it. The main reason for this is to set the stage for something called "bootstrapping," which we will study later.

```
# Create a normal population.
rm(list = ls(all = TRUE))
set.seed(1)
N <- 100000  # Sample size = 100000.
pop <- rnorm(N, 1, 2)  # Draw N samples from a normal distribution with
                       # mu = 1 and sigma = 2.
pop.mean <- mean(pop)
pop.sd <- sd(pop)
pop.median <- median(pop)
c(pop.mean, pop.sd, pop.median)

[1] 0.9955 2.0070 1.0016

hist(pop, breaks = 400)

sample.size <- 200  # Sample size.
sample.trial <- sample(pop, sample.size, replace = T)  # Here is a sample.
```

No reproduction or posting - Allowed



5.1.1 Calculating CI Using Formula

```
sample.stat <-  mean(sample.trial)  # Sample mean.
std.err <- sd(sample.trial) / sqrt(sample.size)  # Calculate the standard error.
sample.stat - abs(qnorm(.05 / 2)) * std.err  # Note z_star .
```

```
[1] 0.7777
```

```
sample.stat + abs(qnorm(.05 / 2)) * std.err  # Sign is correct!
```

```
[1] 1.368
```

```
# sample.stat + qt(0.05 / 2, sample.size - 1) * std.err  # For use later
# sample.stat + qt(1 - 0.05 / 2, sample.size - 1) * std.err
```

5.1.2 Calculating CI Using Built-in Function

```
t.test(sample.trial, alternative = "two.sided", conf.level = 0.95)
```

```

One Sample t-test

data:  sample.trial
t = 7.1, df = 199, p-value = 2e-11
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.7759 1.3702
sample estimates:
mean of x
    1.073
```

Non-reproduction/storage Allowed

```
# Sample.trial contains the data/measurements of x.
# "two-sided" specifies a 2-sided CI, and 0.95 is the confidence level.

# To get the confidence interval, use the following command:
t.test(sample.trial)$conf.int[1:2]

[1] 0.7759 1.3702
```

Note that answers from the two methods (by formula and by computer) are very similar. The interpretation of C.I. is that we can be 95% confident that the true mean resides in this interval.

5.2 Coverage of a Confidence Interval

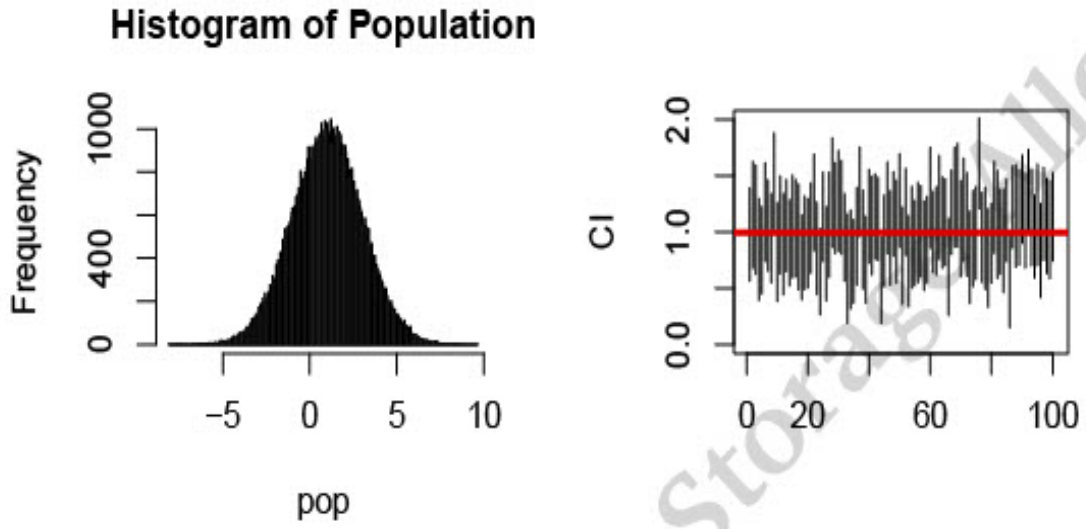
In practice, you will have only one sample (samp) (and not the population (pop)), and so you will build only one CI. But here we want to confirm that the CI, the way we compute it (i.e., with our formulas or with t.test()) covers the population mean the correct percentage of time. **This is what a CI is designed to do: to have the correct coverage.**

To do so, we will draw n.trial = 100 samples of size sample.size = 90 from the normal population, above. For each sample, we will construct the 95% CI and we will make a plot that shows all 100 CIs then count how many of them cover the population mean.

```
rm(list = ls(all = TRUE))
set.seed(1)
N <- 100000
pop <- rnorm(N, 1, 2)
pop.mean <- mean(pop)
pop.sd <- sd(pop)
pop.median <- median(pop)
c(pop.mean, pop.sd, pop.median)

[1] 0.9955 2.0070 1.0016

hist(pop, breaks = 400, main = 'Histogram of Population')

n.trial <- 100  # Number of samples to draw from population.
sample.size <- 90 # Size of each sample = 90.
CI <- matrix(nrow = n.trial, ncol = 2)  # Create space to store n.trial CIs.

for (i in 1:n.trial) {
  sample.trial <- sample(pop, sample.size)  # For each sample/trial,
  CI[i, ] <- t.test(sample.trial)$conf.int[1:2]  # compute (and keep) only CI.
}

count <- 0  # Count number of CIs that cover mu.
for (i in 1:n.trial) {
  if (CI[i, 1] <= pop.mean && CI[i, 2] >= pop.mean) {
    count <- count + 1
  }
}
count

[1] 97
```

Not for Coverage Allowed

```
plot(c(1, 1), CI[1, ], ylim = c(0, 2), xlim = c(0, 101), ylab = "CI", xlab = '',
     type = "l")
for (i in 2:n.trial) {
  lines(c(i, i), CI[i, ])  # Draw CIs (vertically).
}
abline(h = pop.mean, col = "red", lwd = 3)  # The population mean (horizontally).
```



5.3 Two-Sample, Two-Sided Confidence Interval

The following is data from a Statistics class, when students were asked their gender, and what percentage of time they attend class. We will assume percentage is normally distributed, although it is not.

```
dat <- read.table('attend_dat.txt', header = T)
attendance <- dat[, 1]
gender <- dat[, 2]
pa.boy <- attendance[gender == 0]  # Percent of time attending class for boys.
pa.girl <- attendance[gender == 1]  # Percent of time attending class for girls.

n.boys <- length(pa.boy)  # Number of boys. Same as sum(y == 0).
n.girls <- length(pa.girl)  # Number of girls. Same as sum(y == 1).

# The sample mean of these attendance rates is higher for boys than girls:
mean(pa.boy)

[1] 87.57

mean(pa.girl)

[1] 86.4
```

Suppose you wonder if the two true/population means (of attendance rate) are **different**, then, you need to build a 2-sample, 2-sided CI. We will first start by computing 1-sample, 2-sided CIs for each mean:

Not for Distribution or Storage Allowed

```
t.test(pa.boy)$conf.int[1:2]

[1] 79.95 95.19

t.test(pa.girl)$conf.int[1:2]

[1] 81.93 90.87
```

Given the huge overlap between these two confidence intervals, (and given that the two groups - boys and girls - are independent), we can conclude that the data does not provide sufficient evidence to conclude that the attendance rates of boys and girls are different.

Comparing two CIs is not the most elegant way of answering the question. If the comparison of two means (or proportions) is all we care about, then we should compute the CI for the **difference** between the population means (or proportions), i.e., a **2-sample** CI for the difference between means.

```
t.test(pa.boy, pa.girl, alternative = "two.sided")  # Default conf.level = 0.95.


Welch Two Sample t-test

data:  pa.boy and pa.girl
t = 0.27, df = 51, p-value = 0.8
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -7.558  9.891
sample estimates:
mean of x mean of y
    87.57     86.40
```

There are two interpretations:

1. We can be 95% confident that the difference between the true/population means is between -7.559 and 9.891.
2. There is a 95% probability that a 95% CI for the difference between the true means, computed from a random sample, will include the difference between the true/pop means.

Corollary:

The fact that the 2-sided CI, (-7.558, 9.891), includes zero implies that we CANNOT tell if there is a difference between the two proportions. We just cannot say anything. Note, it would be WRONG to conclude that there is NO difference between the true/population means.

5.4 Two-Sample, One-Sided Confidence Interval

Suppose you are NOT interested in whether there is a **difference** between the attendance rates of boys and girls. Instead you are interested in a "weaker" question, namely, is the attendance rate for boys **higher** than that of girls? Denote μ_1 = true/pop mean attendance rate for boys. μ_2 = true/pop mean attendance rate for girls. Then you must build the lower confidence bound for $\mu_1 - \mu_2$. (Or equivalently an upper confidence bound for $\mu_2 - \mu_1$).

```
t.test(pa.boy, pa.girl, alternative = "greater")  # "greater" = LOWER conf. bound.
                                                  # "less" = UPPER conf. bound.
```

No Reproduction Allowed

Interpretations:

1. We are 95% confident that $\mu_1 - \mu_2$ is larger than -6.11.
2. There is a 95% probability that a random 95% lower confidence bound for the difference will be lower than the true difference.

Corollary: This "interval" still includes zero. So, there is no evidence for μ_1 being greater than μ_2.

Recall that you can compute a lower confidence bound for each of μ_1 and μ_2 separately:

```
t.test(pa.boy, alternative = "greater")$conf.int[1:2]

[1] 81.24   Inf

t.test(pa.girl, alternative = "greater")$conf.int[1:2]

[1] 82.67   Inf
```

Example

We will compare the grades on a statistics midterm of those who pick up their tests within the first one or two weeks after the test to those who do not pick it up in that period of time. We use this as a proxy for attendance. The following analysis is conducted to see if there is a statistically significant difference between the means of the two groups.

```
attend <- c(9.0, 14.0, 15.0, 12.5, 13.5, 14.5, 12.5, 8.5, 17.5, 9.5, 12.0, 11.0,
            14.0, 14.5, 14.0, 21.5, 12.5, 10.5, 17.5, 5.0, 10.5, 17.5, 16.5, 19.0,
            18.0, 15.5, 13.5, 21.5, 10.5, 17.0, 18.5, 12.0, 15.0, 17.5, 11.5,
            15.5, 17.0, 17.0, 20.0, 15.5, 12.0, 13.0, 23.0, 11.5, 14.0, 13.0, 22.5,
            8.5, 11.0, 9.5, 11.5, 17.0, 11.5, 17.5, 7.5, 8.0, 14.5, 9.5, 19.0,
            16.5, 18.5, 10.5, 16.5, 14.5, 13.5, 14.5, 12.0, 17.0, 13.0, 11.0, 12.5,
            9.0, 19.0, 15.0, 16.0, 11.0, 7.0, 22.0, 13.0, 7.5, 14.5, 13.0, 18.5,
            13.0, 18.5, 10.0, 20.5, 10.5, 17.5, 13.0, 19.5, 10.0, 13.0, 19.5, 10.5,
            14.5, 11.0, 14.5, 7.0, 7.0, 9.0, 16.0, 13.0, 19.5, 15.0, 17.0, 18.0,
            10.5, 15.0, 8.5, 10.0, 14.0, 16.0, 12.5, 13.5, 17.0)
non.attend <- c(3.0, 12.5, 8.5, 18.5, 5.5, 18.5, 7.5, 13.5, 6.5, 17.0, 11.5, 13.0,
                13.0)
```

To see if the data provide evidence for the claim that $\mu_1 =$ mean of attend is higher than $\mu_2 =$ mean of non.attend, the appropriate CI is a lower confidence bound for $\mu_1 - \mu_2$, which is equivalent to testing

$$H_0 : \mu_1 - \mu_2 \leq 0$$
$$H_1 : \mu_1 - \mu_2 > 0$$

```
t.test(attend, non.attend, alternative = "greater", conf.level = 0.95)


Welch Two Sample t-test

data:  attend and non.attend
```

No Reproduction · Storage · ved

```
t = 1.8, df = 14, p-value = 0.05
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.04549     Inf
sample estimates:
mean of x mean of y
    13.98     11.42
```

This means that we can be 95% confident that the true (i.e. population) mean grade of the attending students is higher than that of the non-attending students by at least 0.045. Because 0 is not included in the CI, the "corollary" conclusion is that the mean grade of attending students is higher than that of the non-attending students. One often says that the difference is "statistically significant." (The same conclusion follows from the p-value; it's smaller than $\alpha = 0.05$, and so we can reject $H_0 : \mu_1 \leq \mu_2$ in favor of $H_1 : \mu_1 > \mu_2$.)

The result is statistically significant, but is it physically significant? That's a different question! In other words, how much higher is the mean of the attendees, and do we care? To answer that, look at the sample means of the two groups (last line of the output). The attending students' grade is $\frac{13.98-11.42}{11.42} \cdot 100 \approx 22\%$ higher than that of the non-attending students. That's big enough to be considered physically significant.

It's important to note that **statistical significance** and **physical significance** are two different concepts. The difference between the two means may be statistically significant, but it may be so small that no one really cares about it, i.e., it may be physically non-significant.

5.5 The t-distribution

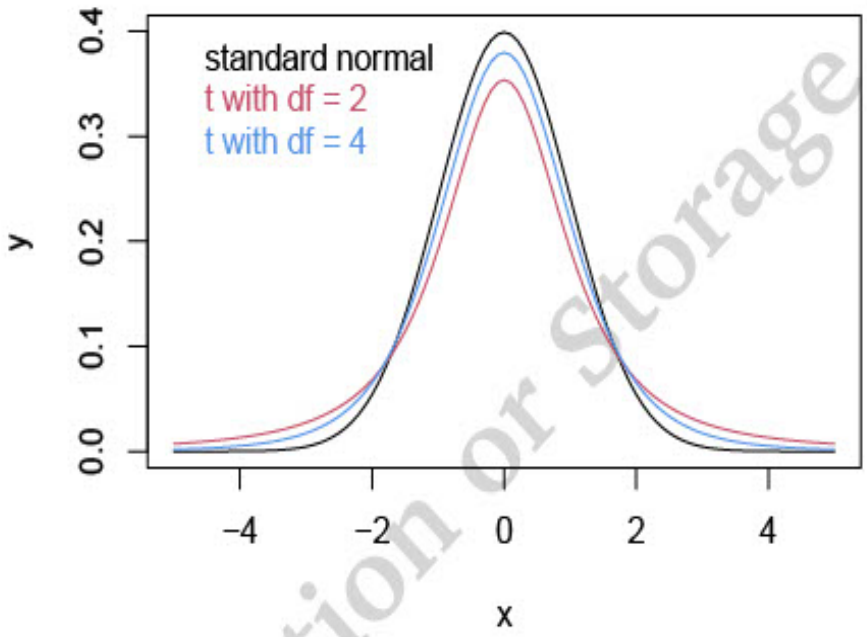
All confidence intervals require knowing areas under distributions in order to get the correct z^* and t^* in the CI formulas. Table 1 in the book gives areas under the standard normal to the **left** of any number. Table 6 in the book gives areas under the t-distribution to the **right** of any number. Note that z^* and t^* themselves are NOT given in these tables. z and t (not starred) are what we compute in the p-value approach.

In R, the analogs of pnorm(), qnorm(), and dnorm(), for the t-distribution are pt(), qt(), and dt(). For example,

```
pnorm(1.645, 0, 1, lower.tail = T)  # Area left of 1.645 under standard normal.

[1] 0.95

pt(1.645, df = 5, lower.tail = F)  # Area right of 1.645 under t with df=5.

[1] 0.08044

# The quantiles of the normal and t distributions are obtained in the following
# way:
qnorm(0.05, 0,1, lower.tail = T)  # z which has 0.05 area to its left.

[1] -1.645

qt(0.05, df = 5, lower.tail = T)  # t which has 0.05 area to its left.

[1] -2.015

# Finally, you can see what the two distributions look like:
x <- seq(-5, 5, .1)  # x going from -5 to +5 in 0.1 steps.
```

Not for Distribution or Storage. Allowed

```r
y_1 <- dnorm(x, 0, 1)  # Standard normal density.
y_2 <- dt(x, 2)  # t density with df = 2.
y_3 <- dt(x, 5)  # t density with df = 5.
plot(x, y_1, type = "l", ylab = 'y')
lines(x, y_2, col = 2)
lines(x, y_3, col = 4)
legend('topleft', c('standard normal', 't with df = 2', 't with df = 4'),
       text.col = c(1, 2, 4), bty = 'n')
```



5.6 Confidence Interval When σ is Unknown (Small Sample)

We know that the sampling distribution of the sample mean is the normal distribution with parameters μ_x and $\sigma_x/\sqrt{n}$. And so, $\frac{\bar{x}-\mu_x}{\sigma_x/\sqrt{n}}$ has a standard Normal distribution. However, if σ_x is unknown, then it has to be approximated with the sample standard deviation s; which is fine, if the sample size is large. However, for small samples, the approximation is poor, and so the sampling distribution of $\frac{\bar{x}-\mu_x}{s/\sqrt{n}}$ does not follow the standard normal, but rather the t-distribution with $n-1$ degrees of freedom where n is the sample size. To find the confidence interval, all we need to know is how to compute areas under the t-distribution between two numbers, just like we what did with the normal distribution. In R, we can replace **qnorm(.05 / 2)** with **qt(0.05 / 2, sample.size - 1)**. The results will be very similar if the sample size is large since the t-distribution converges to normal as the sample size $n \to \infty$. But for small samples (e.g., 20), the confidence interval calculated using a t-distribution will cover the population mean the correct number of times (if the population is normal), while the normal confidence interval will not. For small samples taken from non-normal populations, we do not have any formulas. We should use the bootstrap method instead; see below.

Non-reproduction or Storage Allowed

5.7 Bootstrap: CI without formulas

We have confirmed in 4.2 that the CI computed with the formula $\bar{x} \pm z^* \cdot \frac{\sigma}{\sqrt{n}}$ has the correct coverage property (about 95% of such CIs cover the true mean). But that conclusion is based on several assumptions:

1. We know what z^* to use in the formula.
2. We can approximate σ with the sample standard deviation.

But for some statistics (e.g., sample mean) we don't even have a formula for a CI. One solution to that problem is Bootstrapping.

Example: Producing the Correct CI for Mean

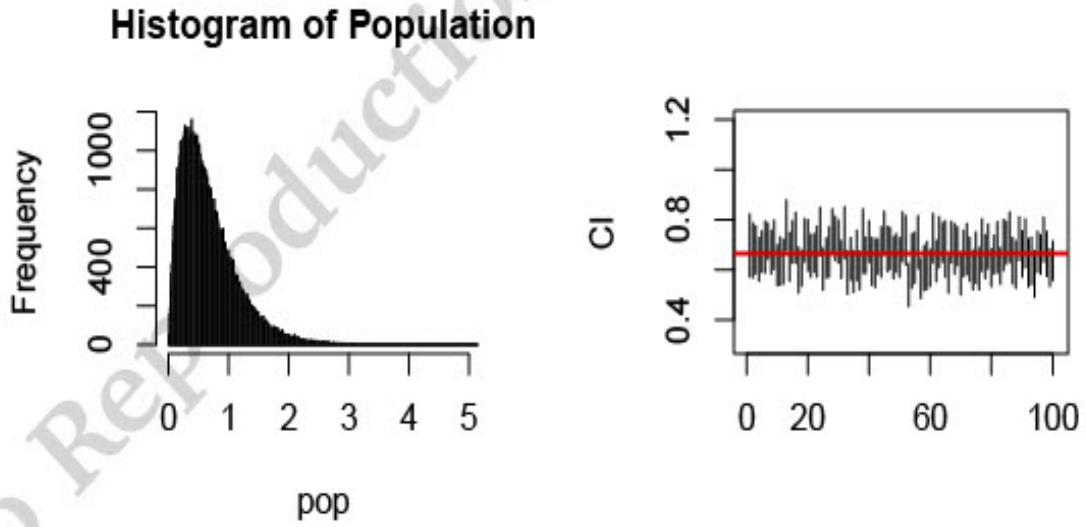
Instead, of using the formula $\frac{sd(sample)}{\sqrt{n}}$ for the standard deviation of the sampling distribution of the sample mean, we can actually build (though approximately) the sampling distribution itself. This is done by taking multiple samples - called bootstrap samples - from the single observed sample! The theory behind bootstrap argues that the std dev of this "sampling distribution" is a pretty good estimate of the standard dev of the sampling distribution of the sample mean. Armed with this approximation to the sampling distribution, we can take its appropriate quantiles to give us CI; after all, $\bar{x} \pm 1.96 \cdot \frac{\sigma}{\sqrt{n}}$ mark quantiles of the true sampling distribution). So, that's the idea: to build a histogram of the sample statistic of interest by treating the sample as if it were the population.

Now, when it comes to testing the coverage properties of a CI for some parameter, recall that we take multiple samples from a population already. So, in the bootstrap approach, we will have to take multiple (bootstrap) samples from each of the samples taken from the population. For technical reasons that we won't go into, the bootstrap samples must be taken with replacement.

```
rm(list = ls(all = TRUE))
set.seed(1)
N <- 100000
pop <- rgamma(N, 2, 3)  # Draw from gamma instead of normal.
pop.mean <- mean(pop)
pop.sd <- sd(pop)
pop.median <- median(pop)
c(pop.mean, pop.sd, pop.median)

[1] 0.6659 0.4705 0.5590

hist(pop, breaks = 400, main = 'Histogram of Population')

n.trial <- 100
sample.size <- 90
CI <- matrix(nrow = n.trial, ncol = 2)
for (i in 1:n.trial) {
  sample.trial <- sample(pop, sample.size)  # Take a sample.
  # Now, the bootstrap block (which you have to type in):
  # For each sample, take a bootstrap sample (with replacement), and compute
  # the sampling distribution of the sample means. The appropriate quantiles
  # of this sampling distribution give the confidence interval.
  n.boot <- 100  # Number of bootstrap samples, from each sample.
  boot.stat <- numeric(n.boot)
    for (j in 1:n.boot) {
```

Not for Distribution - Notes Allowed

```
      boot.sample <- sample(sample.trial, sample.size, replace = T)
      # With replacement.
      boot.stat[j] <- mean(boot.sample)  # Store the means.
      }  # End of loop over bootstrap.

  CI[i, ] <- quantile(boot.stat, c(0.05 / 2, (1 - 0.05 / 2)))
  # CI[i,] <- c(mean(sample.trial) + qnorm(.05/2) * pop.sd / sqrt(sample.size),
  #             mean(sample.trial) - qnorm(.05/2)*pop.sd / sqrt(sample.size))
  # For small sample, replace qnorm(.05/2) with qt(0.05/2, sample.size - 1).
  # CI[i, ] <- sort(boot.stat)[(n.boot + 1) * c(0.05/2, 0.95/2)]  # See Geyer.
}  # End of loop over samples.

count <- 0
for (i in 1:n.trial) {
  if (CI[i, 1] <= pop.mean && CI[i, 2] >= pop.mean)
    count <- count + 1
    }
count

[1] 95

plot(c(1, 1), CI[1, ], ylim = c(0.3, 1.2), xlim = c(0, 101), ylab="CI", xlab = '',
     type = "l")
for (i in 2:n.trial) {
  lines(c(i, i), CI[i, ])  # Draw CIs (vertically).
}
abline(h = pop.mean, col = "red", lwd = 2)  # Draw the population mean
                                            # (horizontally).
```



No Reproduction

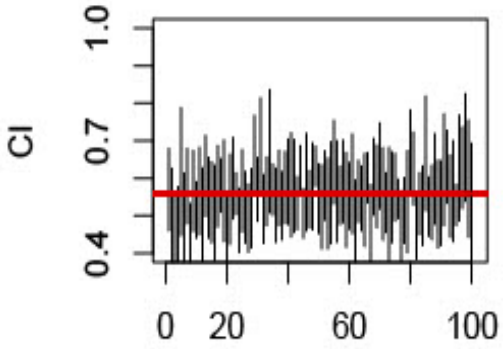
It may seem like the bootstrap method makes no assumptions, and that it will work all the time. However, it turns out that it does have some problems. Some of the problems are addressed by Schenker (1985). For example, he shows that the particular version we use above (called percentile bootstrap) gives CIs which cover the population parameter less frequently than they should, especially for small samples. For example, with a sample size of 20, a 90% CI will cover the pop mean around

78% of the time.

5.7.1 Confidence Interval for Sample Median

```
n.trial <- 100
sample.size <- 90
CI <- matrix(0, n.trial, 2)
for (i in 1:n.trial) {
  sample.trial <- sample(pop,sample.size,replace=F)
  n.boot <- 100
  boot.stat <- numeric(n.boot)
  for (j in 1:n.boot) {
    boot.sample <- sample(sample.trial, sample.size, replace = T)
    boot.stat[j] <- median(boot.sample)  # Median
    }
  CI[i, ] <- quantile(boot.stat, c(0.05 / 2, (1 - 0.05 / 2)))
}

count <- 0
for (i in 1:n.trial) {
  if (CI[i, 1] <= pop.median && CI[i, 2] >= pop.median)
    count <- count + 1
}
count

[1] 96

plot(c(1, 1), CI[1, ], ylim = c(0.4, 1), xlim = c(0, 101), xlab = '', ylab = "CI",
     type = "l")
for (i in 2:n.trial) {
  lines(c(i, i), CI[i, ])
  abline(h = pop.median, col = "red", lwd = 2)
}
```

No Reproduced



Note that the number of times that the confidence interval covers the true median is close to 95. In other words, the way we are computing a confidence interval for a population median gives us confidence intervals that cover the population median the expected number of times. In practice, when you have a **single** sample, and no population, you can use this bootstrap method to build a confidence interval for the population median.

A quick partial fix to the problem of under-coverage is proposed by Charles Geyer:

http://www.stat.umn.edu/geyer/old/5601/examp/percent.html

and it involves revising the CI line just a bit. The commented line in in the above code will let you test this idea.

Reference

Schenker, Nathaniel (1985): Qualms About Bootstrap Confidence Intervals Journal of the American Statistical Association, Vol. 80, No. 390 (Jun., 1985), pp. 360-361.

No Reproduction or Storage Allowed

6 Hypothesis Testing, Confidence Intervals and p-values

6.1 Small Sample Confidence Interval (Unknown σ_x)

For small samples, the sampling distribution of $\frac{\bar{x}-\mu_x}{s/\sqrt{n}}$ is a t-distribution with n-1 degrees of freedom. To compute that confidence interval, all we need to know is how to compute areas under the t-distribution between two numbers, which is similar to what we did with the normal distribution. To find t^*, we just replace qnorm(.05 / 2) with t(0.05 / 2,sample.size-1).

In terms of coverage, the results will be very similar if the sample size is large (e.g., 100+). For small samples (e.g., 10), the CI computed using a t-distribution will cover the population mean the correct number of times, while the CI computed using the normal distribution will not.

Note that computing the confidence interval for small samples using the t-distribution **assumes that the population is normal**. If the population is non-normal, the bootstrap method should be used instead.
Bootstrapping should be used when:

1. The population is not normal and the sample size is small.
2. No formulas for computing standard errors of the statistics of interest exist.

6.2 Confidence Intervals and Hypothesis Tests

In general, this is the way to decide how to set up the null and alternative hypothesis: Convert the statement of the problem to make it sound like "Does **data** provide evidence for blah?" Then that "blah" is what should go into H_1. The reason is that the hypothesis testing procedure starts by assuming whatever is under H_0. And so, if you are trying to see if the *data* provide evidence for X, then you should not start by assuming X is true. Similarly, if the problem asks "Does the data contradict blah?", then the blah should go into H_0.

Some problems don't readily lend themselves to that kind of translation. They ask something like "Test the prior belief that blah." In that case, the blah should go into H_0. The reason is similar to what I said above: Data provides evidence for H_1, against H_0. And "prior" means prior to data. So, any "prior belief" should go into H_0.

So far, we have learned 3 ways of constructing confidence intervals and doing hypothesis tests:

1. Using CI formulas.
2. Using bootstrapping.
3. Using the R function t.test().

The following example will focus on the last method.

Example 1: Exercise 8.28

Fusible interlinings are being used with increasing frequency to support outer fabrics and improve the shape and drape of various pieces of clothing. The article "Compatibility of Outer and Fusible Interlining Fabrics in Tailored Garments" (Textile Res. J., 1997: 137???142) gave the accompanying data on extensibility (%) at 100 gm/cm for both high-quality fabric (H) and poor-quality fabric (P) specimens:

```
H <- c(1.2, 0.9, 0.7, 1.0, 1.7, 1.7, 1.1, 0.9, 1.7, 1.9, 1.3, 2.1, 1.6, 1.8,
       1.4, 1.3, 1.9, 1.6, 0.8, 2.0, 1.7, 1.6, 2.3, 2.0)
P <- c(1.6, 1.5, 1.1, 2.1, 1.5, 1.3, 1.0, 2.6)
```

No Reproduction or Storage Allowed

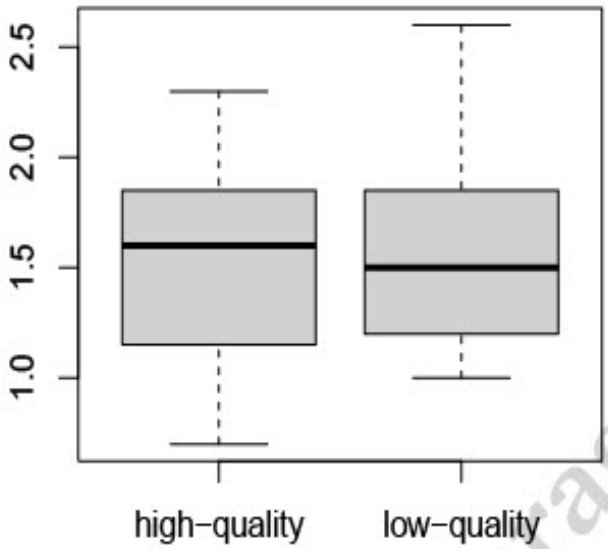
Suppose the problem asked us to estimate the true means for the the two populations separately. Then, we would compute 2-sided, 1-sample, CIs for each of the two population means.

```
t.test(H)


One Sample t-test

data:  H
t = 17, df = 23, p-value = 3e-14
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 1.321 1.696
sample estimates:
mean of x
    1.508

t.test(P)


One Sample t-test

data:  P
t = 8.5, df = 7, p-value = 0.00006
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 1.144 2.031
sample estimates:
mean of x
    1.588
```

However, if we were only comparing the means, we would not be able to tell much about the difference in the true means because there is a lot of overlap between the two CIs.

```
boxplot(H, P, names = c("high-quality", "low-quality"))
```

No Reproduction Allowed



If we are interested in the difference between the two means, it's better to compute a 2-sample CI, instead of comparing two 1-sample CIs.

Suppose the problem asks "Does the data provide evidence to support the claim that the two populations have different means?" Then, we need to construct a 2-sided, 2-sample, CI. The two hypotheses are:

$$H_0\text{: } \mu_H - \mu_P = 0$$
$$H_1\text{: } \mu_H - \mu_P \neq 0$$

```
t.test(H, P, alternative = "two.sided")


Welch Two Sample t-test

data:  H and P
t = -0.38, df = 10, p-value = 0.7
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.5404  0.3820
sample estimates:
mean of x mean of y
    1.508     1.588
```

Note that the 95% CI includes zero. Also note that p-value > 0.05. Both of these observations imply that we cannot reject the null hypothesis that the two means are equal. One often says "there is no statistically significant difference between the means of H and P." Keep reminding yourself that this does NOT mean that there is no difference; it just means that if there is a difference, your data is not seeing it.

Not for Reproduction or Storage Allowed

Also note that the sample mean of H is smaller than the sample mean of P. Suppose the problem had asked us if the data provide evidence that the population mean of H is less than the population mean of P. Then the appropriate "interval" would be a (1-sided) **upper** confidence bound for $\mu_H - \mu_P$. The two hypotheses would be:

$$H_0\text{: } \mu_H - \mu_P > 0$$
$$H_1\text{: } \mu_H - \mu_P < 0$$

```
t.test(H, P, alternative = "less")


Welch Two Sample t-test

data:  H and P
t = -0.38, df = 10, p-value = 0.4
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
   -Inf 0.2966
sample estimates:
mean of x mean of y
    1.508     1.588
```

The upper confidence bound is positive, and so the difference $(\mu_H - \mu_P)$ may be positive. So, the data do NOT provide evidence that $\mu_H - \mu_P < 0$. Although the p-value is lower than the 2-sided p-value above, it's still not less than $\alpha = 0.05$. So, the conclusion is that the data do NOT provide evidence to reject $\mu_H - \mu_P > 0$ in favor of H_1. Either way, the conclusion is the same.

Note that this (1-sided) upper confidence bound is smaller than the upper limit of the 2-sided CI. This is consistent with what confidence intervals are supposed to do, i.e., cover the true parameter some percentage of the time.

Had the problem asked us if there is evidence for $\mu_H - \mu_P > 0$, then the hypotheses would be:

$$H_0\text{: } \mu_H - \mu_P < 0$$
$$H_1\text{: } \mu_H - \mu_P > 0$$

```
t.test(H, P, alternative = "greater")


Welch Two Sample t-test

data:  H and P
t = -0.38, df = 10, p-value = 0.6
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.4549     Inf
sample estimates:
mean of x mean of y
    1.508     1.588

# Note that the two 1-sided p-values, above, do sum to 1, as they should.

# If you want to extract only the p-value from t.test(), do the following:
t.test(H, P)$p.value
```

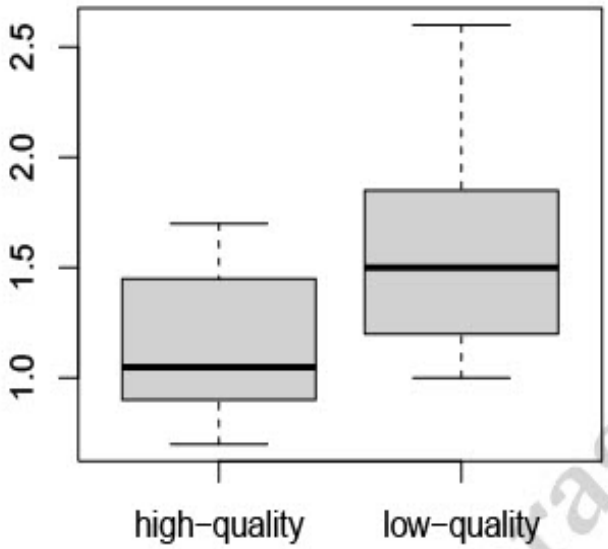
```
[1] 0.7115

# If the above command was placed inside a loop, R will not print the values on
# the screen, unless you put the whole thing in a print(), i.e.,
print(t.test(H, P)$p.value)

[1] 0.7115
```

Example 2: Paired and Unpaired Two-sample t-test

Suppose the data (from example 1) on H and P were of the same size. Assume the data on H was just the first 8 cases. Suppose the question had asked "is there a difference?"

```
H <- H[1:8]  # Keep only the first 8 cases in above H.
boxplot(H, P, names = c("high-quality", "low-quality"))
t.test(H, P, alternative = "two.sided")


Welch Two Sample t-test

data:  H and P
t = -1.9, df = 13, p-value = 0.08
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.93351  0.05851
sample estimates:
mean of x mean of y
    1.150     1.588

# Note that although the p-value is pretty small, it's still greater than
# alpha = 0.05.
```

No Reproduction Allowed



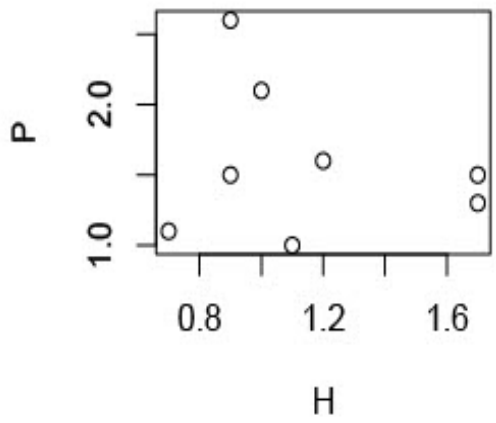
Now suppose the problem had said that the two sets of measurements, H, P, are taken on the same unit of study. For example, the two measurements are made on a given fabric, but in two different conditions, say wet and dry. Then we are dealing with paired data. Then the appropriate test would be:

```
t.test(H, P, paired = T, alternative = "two.sided")


Paired t-test

data:  H and P
t = -1.8, df = 7, p-value = 0.1
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.0252  0.1502
sample estimates:
mean of the differences
                -0.4375
```

The CI is now much wider and the p-value is much larger. So, the pairing of the data means that it provides even less evidence than otherwise. This makes sense in this example, only because the data on H and P are not paired anyway. You can see that they are not paired by looking at the their scatterplot, and noting that there is no correlation:

```
plot(H, P)
```

No Reproduction or Storage Allowed



But, if the data truly were paired, the CI for the paired data would be narrower than that of the unpaired data. Similarly, the p-value for the paired test would be smaller than the p-value from an unpaired test. That makes sense too, because by taking the difference between two columns of data, all the variability **within** each column is "subtracted out," and so the test can focus only on the variability in the **difference between** the two columns, which is all we really care about anyway.

Note that comparative boxplots of paired data are misleading. For example, it's possible that the boxplots will show a huge overlap, but each case in H is higher than the corresponding/paired case in P. If H is greater than P, case-by-case, then the conclusion that H has a higher mean than P is warranted, and yet the boxplots will simply not show that.

Example 3: Exercise 8.38

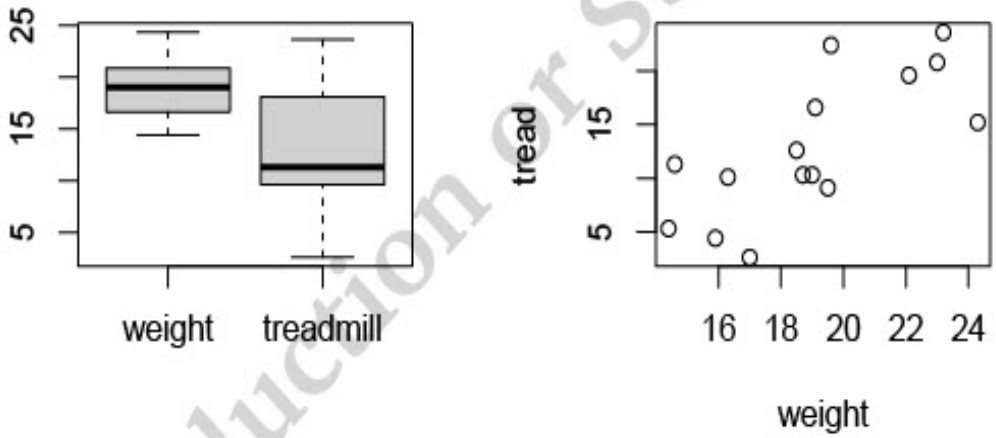
Elevated energy consumption during exercise continues after the workout ends. Because calories burned after exercise contribute to weight loss and have other consequences, it is important to understand this process. The paper "Effect of Weight Training Exercise and Treadmill Exercise on Post-Exercise Oxygen Consumption" (Medicine and Science in Sports and Exercise, 1998: 518???522) reported the accompanying data from a study in which oxygen consumption (liters) was measured continuously for 30 minutes for each of 15 subjects both after a weight training exercise and after a treadmill exercise. Carry out a formal test to decide whether there is compelling evidence for concluding that true average consumption after weight training exceeds that for the treadmill exercise by more than 5. Does the validity of your test procedure rest on any assumptions, and if so, how would you check the plausibility of what you have assumed?

First, ask yourself if the data are paired. In this problem the answer is Yes, based simply on the statement of the problem regarding how the data were collected.

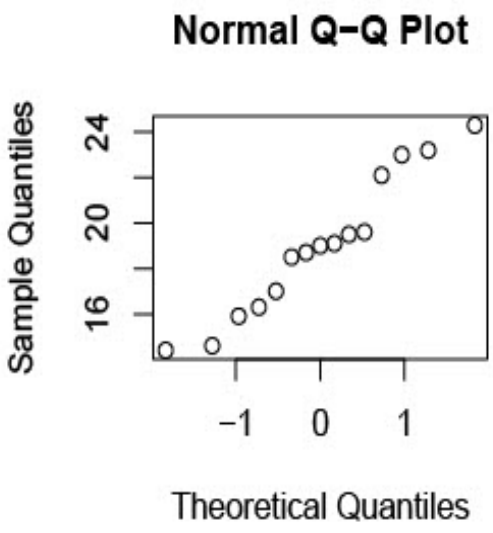
```
weight <- c(14.6, 14.4, 19.5, 24.3, 16.3, 22.1, 23, 18.7, 19, 17, 19.1, 19.6,
            23.2, 18.5, 15.9)
tread <- c(11.3, 5.3, 9.1, 15.2, 10.1, 19.6, 20.8, 10.3, 10.3, 2.6, 16.6, 22.4,
           23.6, 12.6, 4.4)

# Before doing any tests, always "look" at the data:
boxplot(weight, tread, names = c("weight", "treadmill"))
# Note that these data are paired; these boxplots do NOT reflect that fact.
# As such, the comparison of these boxplots is very misleading, and the conclusions
```

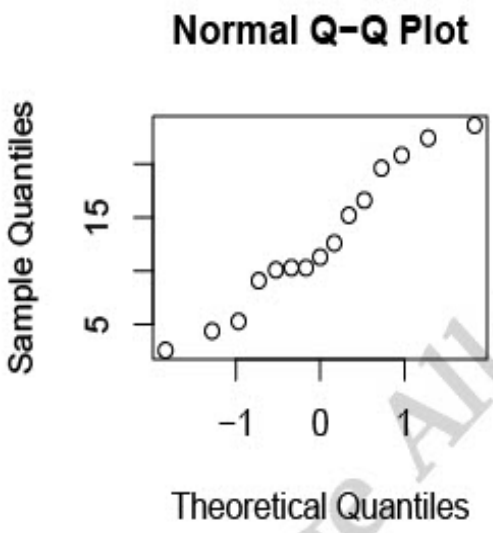
No Reproduction or Storage Allowed

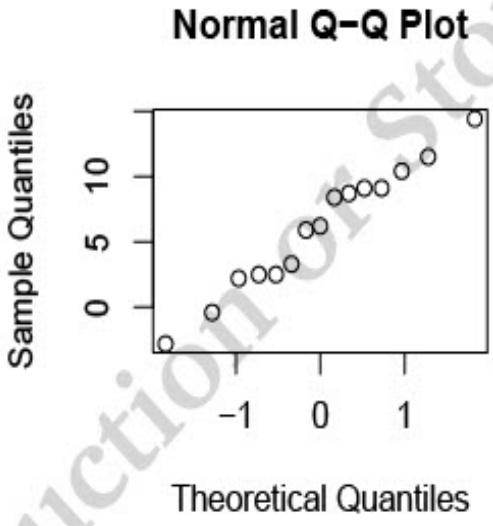
```
# can be completely wrong for paired data. But it is useful to look at for unpaired
# data.

# The scatterplot of the two variables shows a correlation
plot(weight, tread)
cor(weight, tread)

[1] 0.7419

# Now, the t.test assumes that the population is normal. So, let's see
# if our data are at least consistent with that assumption:
qqnorm(weight)
qqnorm(tread)

# These could look better! But with the small sample size we're dealing with,
# they are normal enough. Also, technically, since we need to do a paired test,
# it is the differences that should have a normal distribution.
qqnorm(weight - tread)
```



No Reproduction or Storage Allowed







Now, suppose the problem had asked if the data suggest that the mean consumption associated with weight training is higher than that associated with treadmill exercise. The hypotheses would be:

$$H_0\text{: } \mu_{weight} - \mu_{tread} \leq 0$$
$$H_1\text{: } \mu_{weight} - \mu_{tread} > 0$$

The appropriate CI or test would be the two-sample, 1-sided, t-test. But, which side - the lower or the upper confidence bound?

```
t.test(weight, tread, paired = T, alternative = "greater")


Paired t-test

data:  weight and tread
t = 4.9, df = 14, p-value = 0.0001
```

Non-Reproduction or Storage Allowed

```
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 3.902   Inf
sample estimates:
mean of the differences
                 6.067
```

This particular arrangement of arguments in t.test(), and "alternative = greater" produce the lower confidence bound for $\mu_{weight} - \mu_{tread}$. Here, it's about 3.9, and it's greater than 0. So, we would say that we are 95% confident that the true difference between the means is greater than 3.9. So, there is evidence (from the data) that μ_{weight} is greater than μ_{tread}.

Also, note that the p-value is small (below any of the common α values). So, the conclusion would be "Yes, the data do provide sufficient evidence to reject H_0 in favor of the alternative (that $\mu_{weight} > \mu_{tread}$).

Now, returning to the exact statement of the problem, it asks if there is sufficient evidence that the difference exceeds 5 (not zero). The appropriate hypotheses are:

$$H_0\text{: } \mu_{weight} - \mu_{tread} \leq 5$$
$$H_1\text{: } \mu_{weight} - \mu_{tread} > 5$$

This is how you do the t.test():

```
t.test(weight, tread, mu = 5, paired = T, alternative = "greater")


Paired t-test

data:  weight and tread
t = 0.87, df = 14, p-value = 0.2
alternative hypothesis: true difference in means is greater than 5
95 percent confidence interval:
 3.902   Inf
sample estimates:
mean of the differences
                 6.067
```

The lower confidence bound is still about 3.9 . But the conclusion is now different. Because 3.9 is lower than 5 (i.e., 5 is inside the "interval"), we CANNOT say anything! The most we can say is that there is **insufficient** evidence to conclude that the true average consumption after weight training exceeds that for the treadmill exercise by more than 5.

The p-value approach leads to the same conclusion:

The p-value for this test is different from that of the previous part. Now, the p-value is large (larger than any common value of α). As a result, the data do NOT provide sufficient evidence to reject H_0 in favor of H_1 (that $\mu_{weight} - \mu_{tread} > 5$).

```
# Computing the last p-value "by hand." First, compute the observed statistic
# (z, t, ...)
t_obs <- (mean(weight - tread) - 5) / (sd(weight - tread) / sqrt(15))
t_obs   # Note that this agrees with t_observed given in t.test().

[1] 0.8681
```

No Reproduction/Storage Allowed

```r
# According to the formulas for the paired t-test, we should find the area
# under the t-distribution to the right (see H1) of this t_observed:
pt(t_obs, lower.tail = F, df = 15-1)  # p-value = upper tail.
```

```
[1] 0.2
```

```r
# Note that gives the same p-value as t.test().
```

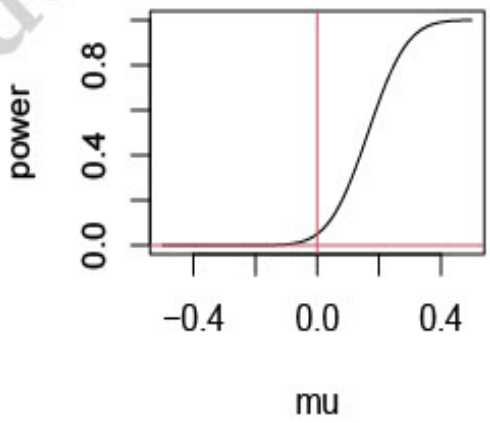
6.3 Power of A Test

Recall that the power of a test is defined as $power = 1 - \beta$ where β is the probability of making a type II error. The main reason why we care about power is that just concentrating on α (which is what most people do) has some serious and adverse consequences in decision making.

```r
n <- 100  # Sample size.
pop.sd <- 1  # Population standard deviation.
mu0 <- 0  # the null parameter.

alpha <- 0.05
a <- qnorm(1 - alpha, mu0, pop.sd / sqrt(n))  # Value of x_bar with right-area = 0.05.
a  # Note this is not 1.64, but 1.64/(sigma/root(n)).
```

```
[1] 0.1645
```

```r
mu <- seq(-0.5, 0.5, 0.01)  # Different values of mu.
power <- pnorm(a, mu, pop.sd / sqrt(n), lower.tail = F)
# Note that we are doing a one-sided test because H1: mu > mu0.
plot(mu, power, type = "l")
abline(v = 0, col = 2)
abline(h = 0, col = 2)
```



Recall what $\alpha = 0.05$ means: If we use the above procedure for testing H_0 vs. H_1 many many times, about 5% of the time we will commit a type I error. In other words, we will reject H_0 when

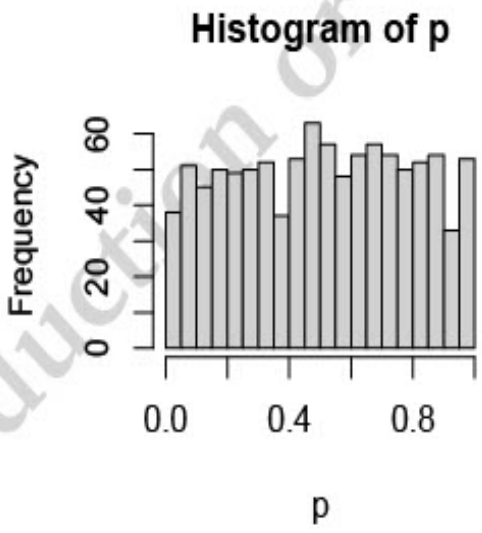
No Reproduction Allowed

it is in fact true. In this problem, we will say μ is not zero, when it really is zero. But what fraction of the time will we reject H_0 when it is in fact False? That's power. So, in this case if μ is 0.4, then nearly 99% of the time we will correctly reject $\mu = 0$.

6.4 Distribution of p-values

Have you wondered what the distribution of p-values (say, from a 2-sided, 2-sample t-test) is under the null hypothesis (of equal means)? The following simulation shows that the p-values have a uniform distribution.

```
mu.1 <- 0  # mu of population 1.
mu.2 <- 0  # mu of population 2.
n.trials <- 1000  # Number of samples to take.
p <- numeric(n.trials)  # Space for storing p-values.
for(i in 1:n.trials) {
  x1 <- rnorm(100, mu.1, 1)  # Sample of size 100 from population 1.
  x2 <- rnorm(100, mu.2, 1)  # and from population 2.
  p[i] <- t.test(x1, x2)$p.value
}
hist(p, breaks = 20, xlim = c(0, 1))  # The distribution is uniform.
range(p)  # Note that some p-values are really really small.

[1] 0.001452 0.998470
```



This result will seem either obvious or completely mysterious. It's not easy to make it intuitive, but think of it this way: If the null hypothesis is true, e.g., if there really is no difference between two population means, then what else can the distribution of p-values be? Any distribution other than uniform would have some nontrivial location (e.g., mean), or scale (e.g., std dev), which means that it cannot be a general/universal answer to the question.

Non-reproduction or S... wed

7 Chi-squared Tests for Proportions (and Independence)

The chi-squared test (chisq.test() in R) can be used in the following 3 situations:

1. Testing whether k proportions in one population are equal to k specific (NULL) values
2. Testing **homogeneity** of r populations with respect to k categories
3. Testing whether two categorical variables are independent

7.1 Chi-squared test of k proportions in 1 population

Example: Tornados and El Nino

The data are as follows:
Number of tornadic days during El Nino years: 14
Number of tornadic days during La Nina years: 28
Number of tornadic days during Normal years: 44
Total number of days: 86

Number of years classified as El Nino: 12
Number of years classified as La Nina: 17
Number of years classified as Normal: 25
Total number of years: 54

We will first assume the following:

1. The proportion of tornadoes occurring in El Nino years is equal to the proportion of El Nino years.
2. The proportion of tornadoes occurring in La Nina years is equal to the proportion of La Nina years.
3. The proportion of tornadoes occurring in Normal years is equal to the proportion of Normal years.

If the above conditions are NOT supported by data, then we can say that "Data suggests that climate has an effect on tornadic activity." To answer the question, we set up the following hypothesis:

$$H_0\text{: } p_1 = \tfrac{12}{54},\ p_2 = \tfrac{17}{54},\ p_3 = \tfrac{25}{54}$$
$$H_1\text{: At least one of the three specifications in } H_0 \text{ is false}$$

```
obs.counts <- c(14, 28, 44)  # Note the data are entered as *counts*.
p0 <- c(12/54, 17/54, 25/54)  # But the null values are given as proportions.
chisq.test(obs.counts, p = p0 )  # Make sure to always specify p = p0.


Chi-squared test for given probabilities

data:  obs.counts
X-squared = 1.8, df = 2, p-value = 0.4
```

Non-reproduction or Storage Allowed

The exact p-value is 0.3988. At $\alpha = 0.05$, since p-value $> \alpha$, we cannot reject the null hypothesis (that climate has no effect on tornadic activity) in favor of the alternative (that it does). In short, there is no evidence that climate effects tornadic activity, at $\alpha = 0.05$.

```
# To see the p.value alone or the expected counts, use the following command:
chisq.test(obs.counts, p = p0)$p.value

[1] 0.3988

chisq.test(obs.counts, p = p0)$expected

[1] 19.11 27.07 39.81

# Diagnosis: check the individual terms in the observed X^2. The way
# to do that in R is to look for residuals. However, R defines the residual
# as (observed - expected)/sqrt(expected), and so to get the terms in X^2,
# you need to square these residuals:

chisq.test(obs.counts, p = p0)$residuals^2

[1] 1.36693 0.03167 0.43993
```

In this example, we can see that the biggest residual is from El Nino. I.e., the biggest difference between observed tornadic counts and the expected counts (if there were no effect between tornadoes and climate) is in El Nino years.

7.2 Testing homogeneity

The chisq.test() function can also be used to test homogeneity of r populations with respect to k categories in each. What that means is whether the k proportions in population 1 are equal to the k proportions in population 2, are equal to the k proportions in population 3, etc. A mathematically equivalent test is whether two categorical variables are independent. In other words, let us consider the contingency table. The question can be translated to whether the column and the row variable are independent. In the following example, the two variables are education level and religiosity. The first one is measured by the highest degree earned: Jr. College, College, and Grad School, and the second one is whether the subject declares him/herself as Fundamentalist or Moderate. Note that the first variable has 3 levels, and the second variable has 2 levels. That's a 2×3 (or 3×2) contingency table.

The question we want to answer is the following: Do the data provide evidence that religiosity and education are independent? (Equivalently, do the data provide evidence that religion is not homogeneous with respect to education?) To answer the question, we will set up the following hypothesis:

H_0: Religiosity is independent of education. (Religion is homogeneous with respect to education.)
H_1: Religiosity is dependent on education. (Religion is not homogeneous with respect to education.)

```
obs.counts = matrix(c(728, 1304, 495, 1072, 2800, 1193), ncol = 3, byrow = T)
chisq.test(obs.counts)


Pearson's Chi-squared test

data:  obs.counts
X-squared = 58, df = 2, p-value = 3e-13
```

Not Reproduction for Sale

```
# The columns are highest degree earned: Jr. College, College, and Grad School.
# The rows are religious belief: Fundamentalist, and Moderate.
```

Conclusion: Given that p-value $< \alpha$, we can reject the null hypothesis in favor of the alternative. I.e., there is evidence from data that religiosity is dependent on education. (Equivalently, religion is not homogeneous with respect to education.)

```
# To see the expected counts and the individual terms in X^2:
chisq.test(obs.counts)$expected
```

```
       [,1] [,2]   [,3]
[1,]  599.1 1366  561.9
[2,] 1200.9 2738 1126.1
```

```
chisq.test(obs.counts)$residuals^2
```

```
      [,1]  [,2]  [,3]
[1,] 27.72 2.816 7.954
[2,] 13.83 1.405 3.968
```

We can see that the biggest discrepancy between expected and observed is in the first category, i.e., in Jr, College. The next biggest discrepancy is in Graduate school. In "English:" There is a relationship between religiosity and education level, and the relationship is strongest in Jr. College and Graduate school. But what is that relationship? Nothing in chi.sq answers that question. For that we need to look at the data itself. For example, we can look at the proportion of Moderates within each of the education levels:

```
obs.counts[2, ] / apply(obs.counts, 2, sum)
```

```
[1] 0.5956 0.6823 0.7068
```

One can describe the relationship by saying that Moderateness increases with education level. There may be many different explanations for this pattern (e.g., parental factor, income level, etc.), but it certainly says that there is a positive relationship between moderateness and education.

7.3 Chi-squared using basic formulas

```
obs.counts <- matrix(c(435, 58, 89, 375, 50, 84), ncol = 3, byrow = T)
total <- sum(obs.counts)
rowsum <- apply(obs.counts, 1, sum) # If unfamiliar with apply(), look-up help. Here,
colsum <- apply(obs.counts, 2, sum) # it simply finds row and column marginals.
expected <- (matrix(rowsum) %*% t(matrix(colsum))) / total
expected
```

```
      [,1]  [,2]  [,3]
[1,] 432.1 57.61 92.29
[2,] 377.9 50.39 80.71
```

```
# This is the matrix of expected counts if the populations are homogeneous
# with respect to the categories (or if the row and column variables were
# independent).
residuals <- (obs.counts - expected) / sqrt(expected)
```

```r
df <- prod(dim(obs.counts) - 1)  # This df is just the product (nrow-1)*(ncol-1).
X2 <- sum(residuals^2)  # = observed X-squared.
1-pchisq(X2, df)
```

```
[1] 0.8614
```

```r
# p-value = area under the chi-squared distribution to the right of the
# observed X-squared.
```

No Reproduction or Storage Allowed

8 F-test for 1-way ANOVA

Recall that the main question that can be addressed by 1-way ANOVA is whether the means of k samples are equal. Thus, we have the following hypothesis:

$$H_0\text{: } \mu_1 = \mu_2 = \mu_3 = \cdots = \mu_k$$
$$H_1\text{: At least two of the } \mu\text{'s are different.}$$

Example: Table 9.1 Vibration (in microns) in five groups of electric motors with each group using a different brand of bearing

	Brand 1	Brand 2	Brand 3	Brand 4	Brand 5
	13.1	16.3	13.7	15.7	13.5
	15.0	15.7	13.9	13.7	13.4
	14.0	17.2	12.4	14.4	13.2
	14.4	14.9	13.8	16.0	12.7
	14.0	14.4	14.9	13.9	13.4
	11.6	17.2	13.3	14.7	12.3
Mean:	13.68	15.95	13.67	14.73	13.08
St. dev:	1.194	1.167	.816	.940	.479
ANOVA Table					
Source	df	SS	MS	F	
Factor	4	30.88	7.72	8.45	
Error	25	22.83	.913		
Total	29	53.71			

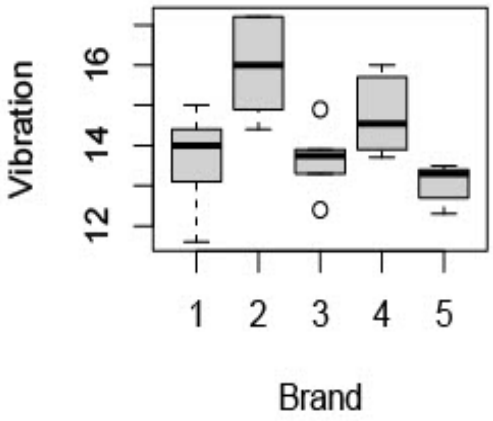
Note that the data provided by the following link are entered in a form that makes ANOVA look like regression: i.e., the 1st column is x and the 2nd column is y. Although lm() is capable of handling situations where x is discrete/categorical (in which case that x is referred to as a dummy variable), generally when one speaks of regression it is assumed that x is continuous. Regardless, in regression the reponse y is continuous, but in ANOVA it's discrete/categorical.

```
dat <- read.table("9_1_dat.txt",  header = TRUE)

aov.1 = aov(Vibration ~ as.factor(Brand), data = dat)
summary(aov.1)

                Df Sum Sq Mean Sq F value  Pr(>F)
as.factor(Brand)  4   30.9    7.71    8.44 0.00019 ***
Residuals        25   22.8    0.91
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Note that similar results can be obtained from general linear models (glm),
# which is a generalization of linear regression using the following command.

# glm.1 = glm(Vibration ~ as.factor(Brand), data = dat)
# anova(glm.1)
```

Since the p-value (.00018 < most α's) is really small, we reject the null in favor of the alternative. I.e., the data suggest that at least 2 of the means are different. One way to identify which two means are different is to at the following boxplots. This plot shows the 5-number summary at each level of x, i.e., something about the **spread** of the data.

Not for Storage Allowed

```
boxplot(Vibration ~ Brand, data = dat)
```



This allows for a visual comparison of the distribution of the 5 populations. The p-value suggests that at least 2 of the means are different. It's evident, for example, that the population means of brand 2 and 5 are probably different. But to quantify this observation, we need to do a "post hoc" analysis, an example of which is Tukey's.

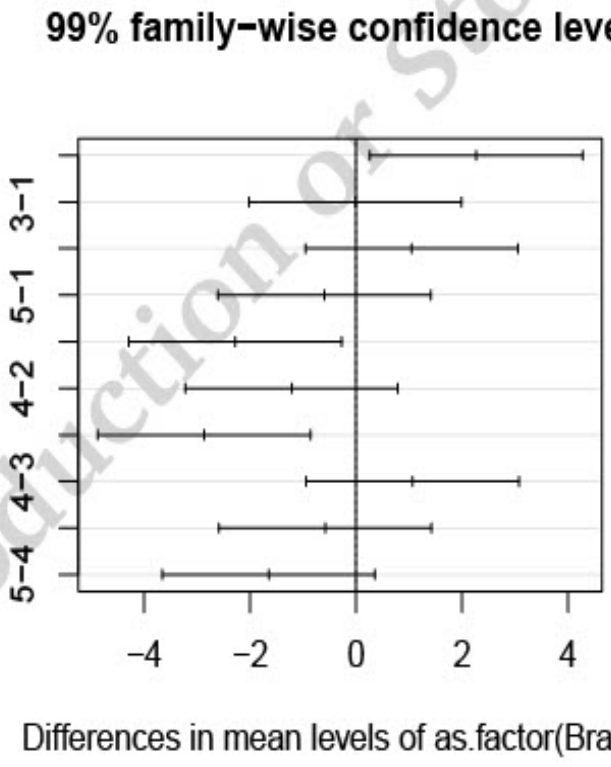
8.1 Tukey's Test

The following performs Tukey's method (section 9.3 of the textbook) for identifying the different means. It gives confidence intervals and p-values for pairwise tests of population means. Recall that if the confidence interval does NOT include zero, then we conclude that the two means being tested are different.

```
library(stats)
tuk.1 <- TukeyHSD(aov.1, conf.level = 0.99)
tuk.1

  Tukey multiple comparisons of means
    99% family-wise confidence level

Fit: aov(formula = Vibration ~ as.factor(Brand), data = dat)

$`as.factor(Brand)`
        diff     lwr     upr  p adj
2-1  2.26667  0.2595  4.2738 0.0032
3-1 -0.01667 -2.0238  1.9905 1.0000
4-1  1.05000 -0.9571  3.0571 0.3418
5-1 -0.60000 -2.6071  1.4071 0.8113
3-2 -2.28333 -4.2905 -0.2762 0.0029
4-2 -1.21667 -3.2238  0.7905 0.2107
5-2 -2.86667 -4.8738 -0.8595 0.0002
4-3  1.06667 -0.9405  3.0738 0.3268
```

Not for Distribution or Storage Allowed

```
5-3 -0.58333 -2.5905  1.4238 0.8262
5-4 -1.65000 -3.6571  0.3571 0.0445

# The lower-bound (lwr) and upper-bound (upr) are given for the difference
# in the mean of two pops. These values are affected by the confidence level in
# TukeyHSD(). Then, look at the p-values in the last column; they test
# H0 vs. H1: two means are different. At alpha = 0.01, it's evident that the
# means of brand 1 and 2 are different. Other different pairs are 3-2, and
# 5-2.  Compare these results with those on page 419; the difference is
# in the alpha level.

plot(tuk.1)  # Shows the CIs as a figure.
abline(v = 0)

# This makes it visually clear that 1-2, 3-2, and 5-2 are three different-mean
# pairs. Note that Tukey's method is done after ANOVA, and its calculations
# depend on the results of ANOVA. As such, it cannot be done before
# ANOVA. There is no point in testing the means pairwise, unless there
# is evidence that at least two of the means are different - and that's
# what the F-test does.
```



8.2 1-Way ANOVA using basic formulas

If only means and standard deviations are given from data, then ANOVA must be done using basic formulas.

For a 1-way ANOVA test, we will first compute the mean and standard deviation for the data in Table 9.1, then use the basic ANOVA equations to show that we get the same answers as above.

No Reproduction or Stor ved

```
dat <- read.table("9_1_dat.txt", header = TRUE)

attach(dat)  # The attach function loads in all variables in the data set.
k <- 5  # Number of categories.
n <- m <- s <- numeric(k)  # Space for mean and sd in each category.
for(i in 1:k) {
    n[i] <- 6  # Sample size in each category.
    m[i] <- mean(dat[Brand == i, 2])  # Mean in each category.
    s[i] <- sd(dat[Brand == i, 2])  # Standard deviation in each category.
}  # Ignore R warnings, if any.

# To do ANOVA by hand, we need n, m and s:
n

[1] 6 6 6 6 6

m

[1] 13.68 15.95 13.67 14.73 13.08

s

[1] 1.1940 1.1675 0.8165 0.9395 0.4792

# ANOVA using basic anova equations:
df.1 <- k - 1  # Numerator degrees of freedom.
df.2 <- k * 6 - k  # Denominator df.
SSB <- sum(n * (m - mean(m)) ^ 2)  # Sum of squares between groups.
SSW <- sum((n - 1) * s ^ 2 )  # Sum of squares within groups.
MSB <- SSB / df.1  # Mean-squared between groups.
MSW <- SSW / df.2  # Mean-squared within groups.
FF <- MSB/MSW  # F-ratio.
p.value <- 1-pf(FF, df.1, df.2)  # p-value.

df.1; df.2; SSB; SSW; MSB; MSW; FF; p.value

[1] 4
[1] 25
[1] 30.86
[1] 22.84
[1] 7.714
[1] 0.9135
[1] 8.444
[1] 0.0001871

# Note that results are the same as the ANOVA table above.
```

No ved

9 T-test and F-test for Regression Coefficients

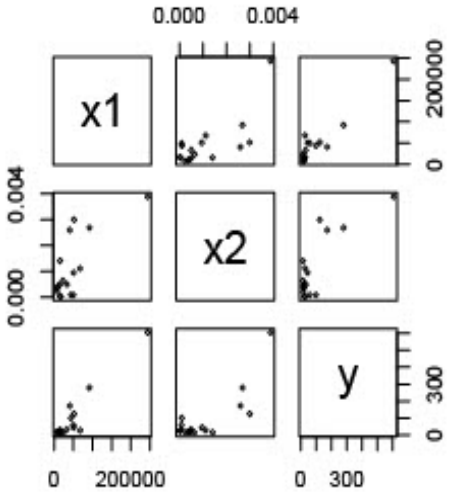
In multiple regression the F-test is for testing if **at least one** of the regression coefficients is nonzero, because then there is evidence that at least one of the predictors is useful, i.e., the model has some utility. Also, if the model has some utility, then we can try to identify which regression coefficients are the nonzero ones, because then the corresponding predictors are the useful ones. The latter step is done with a sequence of t-tests, each on a different coefficient.

Example: Problem 11.39 (in 2nd edition)

Snowpacks contain a wide spectrum of pollutants which may represent environmental hazards. The article "Atmospheric PAH Deposition: Deposition Velocities and Washout Ratios" (J. of Environmental Engineering, 2002: 186???195) focused on the deposition of polyaromatic hydrocarbons. The authors proposed a multiple regression model for relating deposition over a specified time period y to two rather complicated predictors x_1 and x_2 defined in terms of PAH air concentrations for various species, total time and total amount of precipitation. The data is on the web at:

```
dat <- read.table("11_39_dat.txt", header = TRUE)

plot(dat, cex = 0.5)  # Look at the data, and note the collinearity.
model.1 <- lm(y ~ x1 + x2, data = dat)  # Fit a linear model.
summary(model.1)


Call:
lm(formula = y ~ x1 + x2, data = dat)

Residuals:
   Min     1Q Median     3Q    Max
-111.6  -19.7   18.2   27.4   44.9

Coefficients:
               Estimate   Std. Error t value  Pr(>|t|)
(Intercept)   -33.463810    14.896258   -2.25     0.041 *
x1              0.002055     0.000295    6.98 0.0000065 ***
x2          29835.665532 13653.728296    2.19     0.046 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 44.3 on 14 degrees of freedom
Multiple R-squared:  0.923,Adjusted R-squared:  0.912
F-statistic: 84.4 on 2 and 14 DF,  p-value: 0.0000000155
```

No Reuse, Allowed



Note that the F-ratio tests for "model utility", i.e., if at least one variable is a significant predictor of y. The way it's done in practice is to compare the so-called "full model" $(y = intercept + x_1 + x_2)$, against the so-called "null model" $(y = intercept)$. The appropriate statistic is the F-ratio, which is returned in summary():

```
summary(model.1)$fstatistic  # Returns the F-ratio and degrees of freedom

value numdf dendf
84.39  2.00 14.00

summary(model.1)$fstatistic[1]  # Selects only F-ratio.

value
84.39
```

The 3 p-values appearing in the table test the full model against a model without one variable. These are based on the t-tests for testing whether the respective regression coefficient is nonzero. In this case, at $\alpha = 0.01$, it looks like x1 is the useful predictor and at $\alpha = 0.05$, both x1 and x2 are useful. We can also compute a confidence interval for each of the regression coefficients:

```
confint(model.1, level = 0.99)

                    0.5 %       99.5 %
(Intercept)    -77.807627    10.880008
x1               0.001178     0.002932
x2          -10809.336342 70480.667406

# The CI for beta1 does not include zero, but the CI for beta2 does. So the
# conclusions from the CI's are consistent with the conclusions from the p-values.
```

9.1 Confidence Interval vs. Prediction Interval

Recall that the confidence interval is a confidence interval for the population mean of y given x and the prediction interval is not a CI at all because it is not referring to a population parameter. Instead, it

Not for Redistribution Storage Allowed

gives a measure of uncertainty that an **individual** y will be in some interval. The following commands produce both CI and PI for all the cases in the data.

```
predict(model.1, interval = "confidence", level = 0.95)

         fit     lwr    upr
1  235.86924 196.450 275.29
2  162.54267 104.402 220.68
3    2.53724 -29.043  34.12
4    0.02655 -31.166  31.22
5   50.09188  23.823  76.36
6  582.93696 495.581 670.29
7  139.25214 112.823 165.68
8   33.85886   8.752  58.97
9    9.66662 -17.262  36.60
10  -4.41254 -32.630  23.81
11 -10.81166 -39.955  18.33
12  41.05158   5.602  76.50
13  -0.19225 -28.001  27.62
14  59.47189  23.224  95.72
15  71.52335  32.917 110.13
16 127.64824  75.388 179.91
17  99.29921  75.388 123.21

predict(model.1, interval = "prediction", level = 0.95)

Warning in predict.lm(model.1, interval = "prediction", level = 0.95):  predictions on
current data refer to _future_ responses

         fit      lwr    upr
1  235.86924  133.036 338.70
2  162.54267   51.182 273.90
3    2.53724  -97.553 102.63
4    0.02655  -99.942 100.00
5   50.09188  -48.452 148.64
6  582.93696  453.895 711.98
7  139.25214   40.666 237.84
8   33.85886  -64.382 132.10
9    9.66662  -89.055 108.39
10  -4.41254 -103.494  94.67
11 -10.81166 -110.160  88.54
12  41.05158  -60.326 142.43
13  -0.19225  -99.158  98.77
14  59.47189  -42.188 161.13
15  71.52335  -31.001 174.05
16 127.64824   19.242 236.05
17  99.29921    1.358 197.24
```

The output contains the predictions (first column), followed by the lower limit (second column) and upper limit (3rd column) of the prediction interval. Note that the PIs are much wider than the CIs for each of the cases in the data. Also, you may get a warning that accompanies PI; it's meant to remind you that the PI is designed to capture some percentage (e.g. 95%) of "future" y values. These individuals should not be included in the data set that was used to develop the regression equation

(i.e., the training set). Otherwise, the coverage of the PI will not be correct (95%). To get the PI for all the cases in a **new** data set, called new.dat, use the following command:

```
predict(model.1, newdata = new.dat, interval = "prediction", level = 0.95)
```

No Reproduction or Storage Allowed

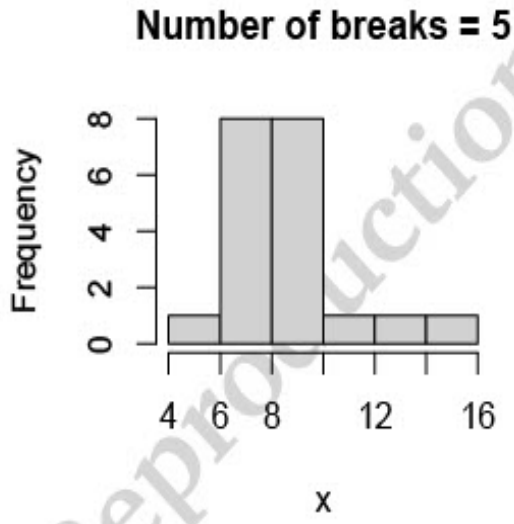
10 Appendix: Quizzes

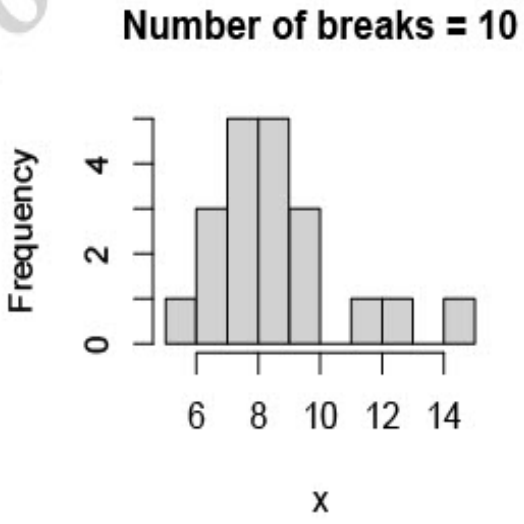
10.1 QUIZ 1

Consider the following data from problem 1.2 (Chapter 1, problem 2) in the book:
6.1 5.8 7.8 7.1 7.2 9.2 6.6 8.3 7.0 8.3 7.8 8.1 7.4 8.5 8.9 9.8 9.7 14.1 12.6 11.2
a. Write code for entering this data into R using one of the methods described in section 1.
b. Write code for plotting a frequency histogram of this data. What are some appropriate values for the quantity "breaks?"
c. Does the data appear to be reasonably symmetric?

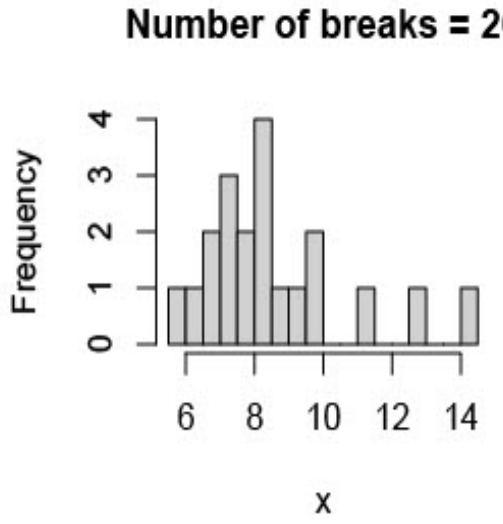
Solution

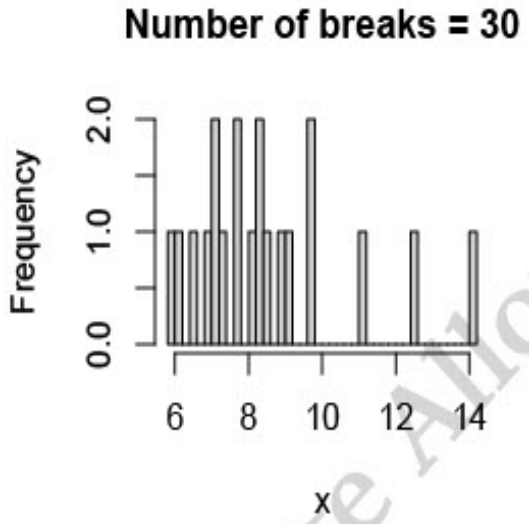
```
# a)
x <- c(6.1, 5.8, 7.8, 7.1, 7.2, 9.2, 6.6, 8.3, 7.0, 8.3, 7.8, 8.1, 7.4, 8.5,
       8.9, 9.8, 9.7, 14.1, 12.6, 11.2)

# b)
hist(x, 5, main = 'Number of breaks = 5')
hist(x, 10, main = 'Number of breaks = 10')
hist(x, 20, main = 'Number of breaks = 20')
hist(x, 30, main = 'Number of breaks = 30')
# Based on the above, any value between 10 and 20 seems appropriate.
```





No Reproduction or Distribution Allowed





c) No.

Moral: Generally, the shape of a histogram does change with the number (and location) of the breaks. But, skewness is one attribute of a histogram that generally does not change significantly with the number of breaks.

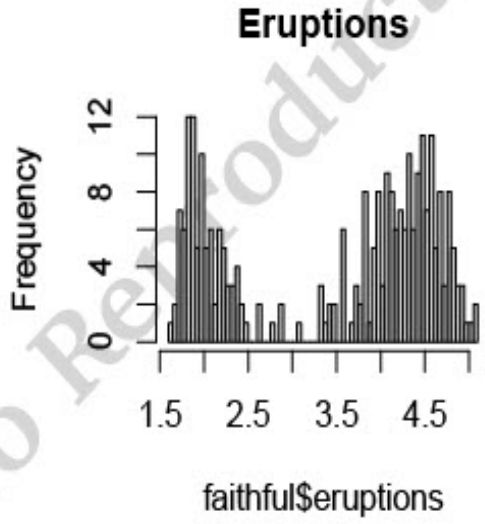
No Reproduction or Storage Allowed

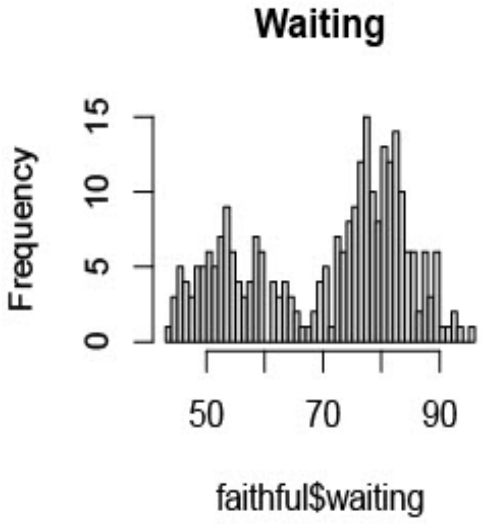
10.2 QUIZ 2

Write code to
a) Identify the names of the two attributes of the R data called "faithful."
b) Compute the mean of one of the two attributes.
c) Plot a frequency histogram for that attribute. Set breaks = 50.
d) Provide at least one explanation (in words) of that histogram.

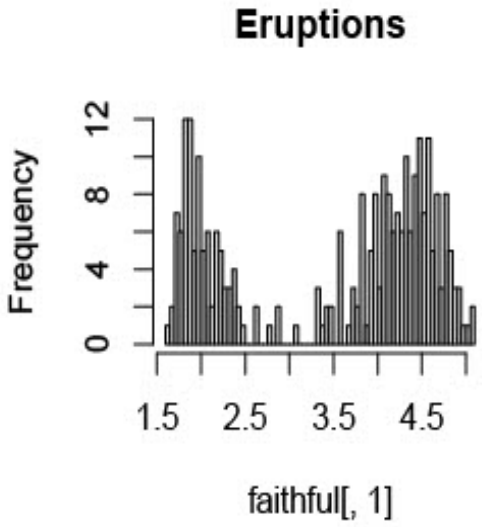
Solution

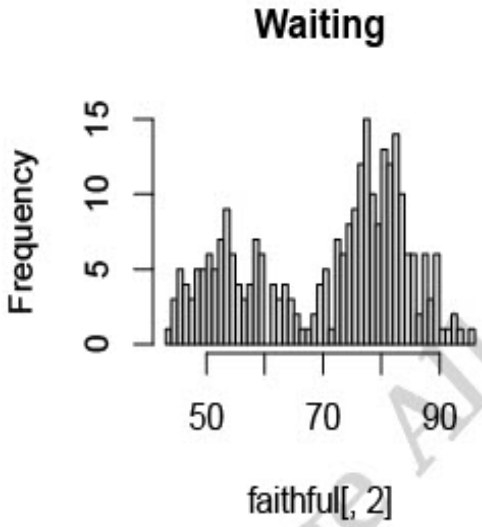
```
# a)
names(faithful)

[1] "eruptions" "waiting"

# b)
mean(faithful$eruptions) # or

[1] 3.488

mean(faithful$waiting)

[1] 70.9

# c)
hist(faithful$eruptions, breaks = 50, main = 'Eruptions') # or
hist(faithful$waiting, breaks = 50, main = 'Waiting')

# or
hist(faithful[,1], breaks = 50, main = 'Eruptions')
hist(faithful[,2], breaks = 50, main = 'Waiting')
```





No Reproduction Allowed





d) Here are some examples of explanations:

- The typical/likely duration of the eruptions is about 2 or 4.5 minutes.
- The average duration of an eruption (around 3) is actually not typical.

Similarly for "waiting time"

- The typical/likely waiting time between eruptions is about 50 or 80 minutes.
- The average duration of an eruption (around 3) is actually not typical.

Moral: There is a lot of information in the shape of a histogram, and in some situations the mean of a quantity is not a very relevant summary measure. The slight discrepancy between the two p-values is because the variances are not equal in this example anyway.

No Reproduction or Storage Allowed

10.3 QUIZ 3

Write code to

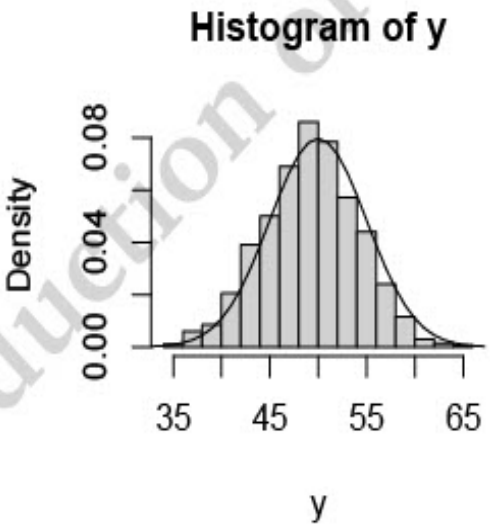
a) Simulate/generate the number of heads in an experiment where 100 fair coins are tossed 1000 times. Assign the 1000 numbers to a variable called y. Hint: this is just one line of code.

b) Plot the histogram of y on a density scale. Hint: one line of code; use the default number of breaks.

c) Overlay on the histogram a plot of the binomial distribution with $n = 100$, $\pi = 0.5$, for x values ranging from 0 to 100. Hint: use lines().

Solution

```
# a)
y <- rbinom(1000, 100, 0.5)
# b)
hist(y,freq = F)
# c)
x <- c(0:100)
lines(x, dbinom(x, 100, 0.5))
```



Moral: Note the agreement between the distribution (according to dbinom()) and the histogram of the sample taken from it (rbinom(1000,100,0.5). The slight relative shift is because of the way R places the histogram bins.

No Reproduction or S e Allowed

10.4 QUIZ 4

Write code to

a) Take 200 points from a uniform distribution between -1 and 1, and call it x.

b) Define a new variable, y, equal to x, but with additional normally distributed noise with $\mu = 0$, $\sigma = 0.5$.

c) Define a new variable, z, equal to x, but with additional normally distributed noise with $\mu = 0$, $\sigma = 0.5$.

d) Compute the correlation coefficient between x and y, x and z, and y and z.

Solution

```
# a
x <- runif(200, -1, 1)
# b
y <- x + rnorm(200, 0, .5)
# c
z <- x + rnorm(200, 0, .5)
# d
cor(x, y)

[1] 0.7447

cor(x, z)

[1] 0.7455

cor(y, z)

[1] 0.574
```

Note, in general, correlation does not satisfy the transitive property. I.e., if x is correlated with y, and x is correlated with z, that does not imply that y and z are equally correlated. In fact, it's even possible to have $r(x,y) > 0$, $r(x,z) > 0$, and $r(y,z) < 0$.

No Reproduction Allowed

10.5 QUIZ 5

We have seen how the correlation coefficient can be "deceived," giving the wrong answer if the scatterplot has clusters. Regression has similar problems. Here we will demonstrate something called Simpson's paradox, which is demonstrated in the figure shown below.

The figure shows two clusters with 50 points in each. The first cluster is made from a uniform x (between 1 and 3), and normally distributed error ($\mu = 0$, $\sigma = 0.3$) about $y = -3 + x$. The other cluster is made from a uniform x (between -3 and 1), and normally distributed error ($\mu = 0, \sigma = 0.3$) about $y = 3 + x$.

Clearly, for each cluster, x and y are related, and so, we can do regression on x and y, leading to the two positively sloped lines. Now, if we had not looked at the scatterplot to see the two clusters, and had instead done regression on the whole data, we would get the straight line with negative slope. In one case the lines have positive slope, while in the other the line has negative slope - hence, the paradox!

Write code to reproduce the figure, showing
1) the two clusters.
2) and the three regression lines.

Note: Don't worry about the axis labels and the colors. Also, the precise location of each dot in your scatterplot will be different than the one shown in the figure, but don't worry about that either.

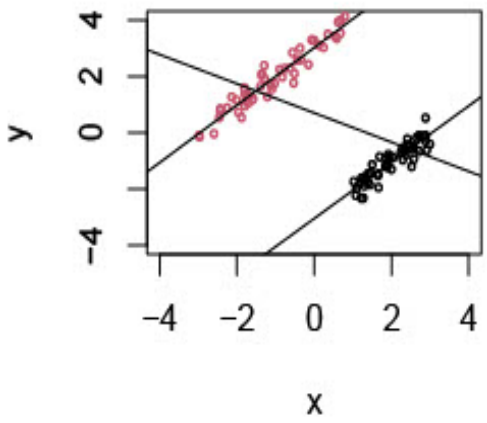
R Hints:

1) $z = c(u, v)$ combines the arrays u and v to make array z.
2) points() overlays points over plot(); see section 2.9.2 (on ecological correlation), if you don't remember.

Solution

```
N <- 50
sigma <- 0.3

x1 <- runif(N, 1, 3)
y1 <- x1 - 3 + rnorm(N, 0, sigma)
lm.1 <- lm(y1 ~ x1)  # Regression for the 1st cluster.

x2 <- runif(N, -3, 1)
y2 <- x2 + 3 + rnorm(N, 0, sigma)
lm.2 <- lm(y2 ~ x2)  # Regression for the 2nd cluster.

x <- c(x1, x2)
y <- c(y1, y2)
lm.3 <- lm(y ~ x)  # Regression for combined data.

plot(x1, y1, xlim = c(-4, 4), ylim = c(-4, 4), xlab = "x", ylab = "y", cex = 0.5)
points(x2, y2, col = 2, cex = 0.5)
abline(lm.1)
abline(lm.2)
abline(lm.3)
```

Not for Reproduction or Storage Allowed



Moral: If there exist clusters in a scatterplot, even the direction of a relationship can be reversed!

No Reproduction or Storage Allowed

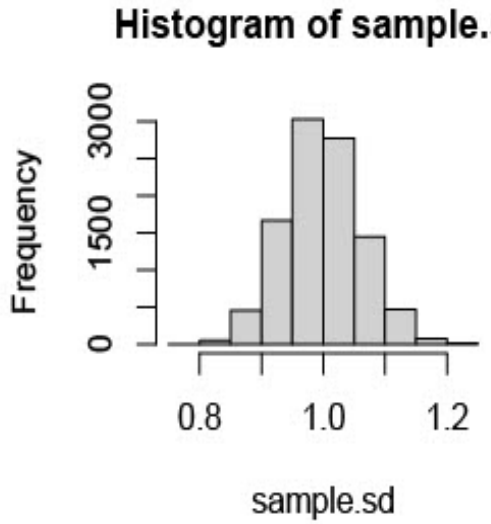
10.6 QUIZ 6

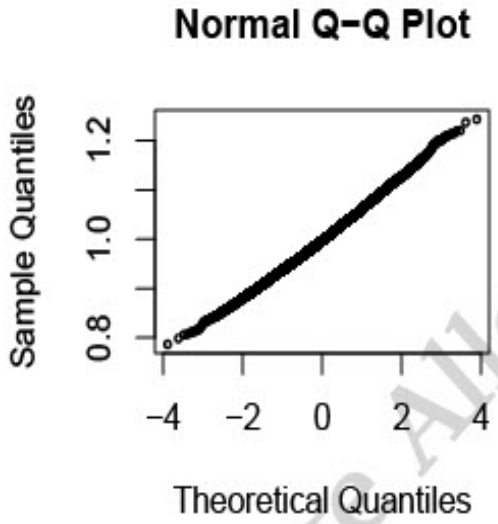
Write code for making the empirical sampling distribution (i.e., a histogram) of the sample standard deviation. Use the non-normal population examined in previous sections, and set the sample size to 500.

Solution

```
rm(list = ls(all = TRUE))  # Clear workspace
N <- 100000
set.seed(1)
pop <- rgamma(N, 1, 1)

n.trial <- 10000  # Take 10000 samples of
sample.size <- 500  # of size 500 from the population.
sample.sd <- numeric(n.trial)  # Space for storing the 10000 sample sd.

for (i in 1:n.trial) {
  samp <- sample(pop, sample.size, replace = T) # Draw a sample (with
                                                # replacement).
  sample.sd[i] <- sd(samp)  # Compute each sample's standard deviation.
}
hist(sample.sd)

# The moral of this exercise is that
# 1. One can make a sampling distribution for any statistic, even the sample std dev.
# 2. and that the sampling distribution of the sample std dev is again
# normal to very good approximation, even when the population itself is not.
# Take a look at

qqnorm(sample.sd, cex = 0.5)

# This normality will help us, later, to make inferences about the
# population standard deviation.
```

No Reproduced





No Reproduction or Storage Allowed

10.7 QUIZ 7

Edit the bootstrap code (with non-normal population) so that it will compute the CI for the sample median.

```
rm(list = ls(all = TRUE))
set.seed(1)
N <- 100000
pop <- rgamma(N, 2, 3)
pop.mean <- mean(pop)
pop.sd <- sd(pop)
pop.median <- median(pop)
c(pop.mean, pop.sd, pop.median)
hist(pop, breaks = 400, main = 'Histogram of Population')

n.trial <- 100
sample.size <- 90
CI <- matrix(nrow = n.trial, ncol = 2)
for (i in 1:n.trial) {
  sample.trial <- sample(pop, sample.size)
  n.boot <- 100
  boot.stat <- numeric(n.boot)
    for (j in 1:n.boot) {
      boot.sample <- sample(sample.trial, sample.size, replace = T)
      boot.stat[j] <- mean(boot.sample)
      }
  CI[i, ] <- quantile(boot.stat, c(0.05 / 2, (1 - 0.05 / 2)))
}

count <- 0
for (i in 1:n.trial) {
  if (CI[i, 1] <= pop.mean && CI[i, 2] >= pop.mean)
    count <- count + 1
    }
count

plot(c(1, 1), CI[1, ], ylim = c(0.3, 1.2), xlim = c(0, 101), ylab="CI", xlab = '',
     type = "l")
for (i in 2:n.trial) {
  lines(c(i, i), CI[i, ])
}
abline(h = pop.mean, col = "red", lwd = 2)
```

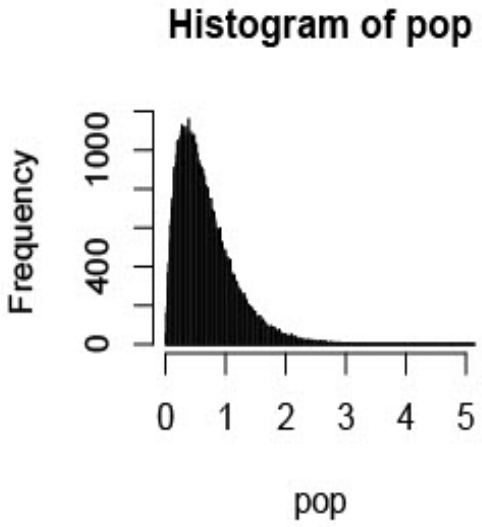
Solution

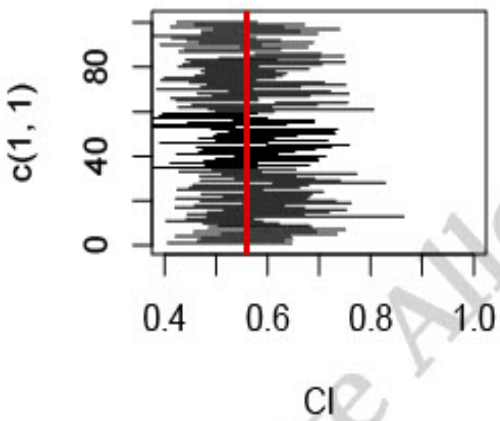
```
rm(list = ls(all = TRUE))
set.seed(1)
N <- 100000
pop <- rgamma(N, 2, 3)  # gamma instead of normal.
pop.mean <- mean(pop)
```

No Re ved

```
pop.sd <- sd(pop)
pop.median <- median(pop)
c(pop.mean,pop.sd,pop.median)

[1] 0.6659 0.4705 0.5590

hist(pop, breaks = 400)

n.trial <- 100
sample.size <- 90
CI <- matrix(0, n.trial, 2)

for (i in 1:n.trial) {
  sample.trial <- sample(pop, sample.size, replace = F)
  n.boot <- 100
  boot.stat <- numeric(n.boot)
  for (j in 1:n.boot){
    boot.sample <- sample(sample.trial, sample.size, replace = T)
    boot.stat[j] <- median(boot.sample)  # MEDIAN
  }

  CI[i,] <- quantile(boot.stat, c(0.05 / 2, (1 - 0.05 / 2)))
}

count <- 0
for (i in 1:n.trial) {
  if (CI[i,1] <= pop.median && CI[i,2] >= pop.median)  # MEDIAN
    count = count+1
  }
count

[1] 91

plot(CI[1, ], c(1, 1), xlim = c(0.4, 1), ylim = c(0, 101), xlab = "CI",
     type="l")
for (i in 2:n.trial)
  lines(CI[i, ], c(i, i))
  abline(v = pop.median, col = "red", lwd = 3)  # population MEDIAN

# Again, it's pretty close to 95. In other words, the way we are computing
# a CI for a population median gives us CIs that cover the population median
# the expected number of times. So, in practice, when you have a SINGLE
# sample, and no population (of course), you can use this bootstrap method
# to build a CI for the population median.
```

No





No Reproduction or Storage Allowed

10.8 QUIZ 8

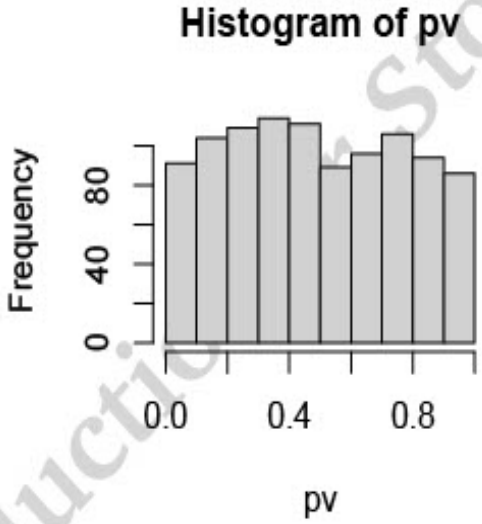
Using rnorm(), take 1000 samples of size 500 from a normal distribution with parameters 0 and 1. For each of the 1000 samples, do a t.test of H_0: $\mu = 0$ vs. H_1: $\mu \neq 0$, and store the resulting p-value. Finally, compute the histogram of the 1000 p-values.

R hint:

pv = numeric(1000) makes a vector of size 1000.

Solution

```
pv <- numeric(1000)
for(i in c(1:1000)){
  x <- rnorm(500, 0, 1)
  pv[i] <- t.test(x, mu = 0, alternative = "two.sided")$p.value
}
hist(pv)
```



Moral: Note that the distribution looks uniform between 0 and 1. In fact, it turns out that the distribution of p-values under H_0 is in fact uniform between 0 and 1. Quite generally it's strange, but true. E.g. when $\mu = 0$, you may get a sample with p-value below even $\alpha = 0.01$, which means that you would reject $\mu = 0$. This is a type I error, and α is the probability of that error.

No Reproduction or Storage Allowed

10.9 QUIZ 9

You've tossed a 6-sided die 60 times, and have recorded the following frequency table for obtaining each of the 6 numbers on the die.

Face	1	2	3	4	5	6
Frequency	11	8	7	13	12	9

You are interested in whether this data provide sufficient evidence to conclude that the die is not fair. You would set-up the following hypotheses.

H_0: The die is fair H_1: The die is not fair

a) Use chisq.test() and report the p-value.
b) State your conclusion in words (at $\alpha = 0.01$).

Solution

If H_0 is true, then $\pi_1 = 1/6$, $\pi_2 = 1/6$ ... $\pi_6 = 1/6$ under H_1, at least one of these values of π_0 is incorrect.

```
# a
obscounts <- c(11, 8, 7,13, 12, 9)
pi0 <- c(1/6, 1/6, 1/6, 1/6, 1/6, 1/6)
chisq.test(obscounts, p = pi0)


Chi-squared test for given probabilities

data:  obscounts
X-squared = 2.8, df = 5, p-value = 0.7

# Alternatively, chisq.test(obscounts)
```

b) Given that this p-value is greater than α, we cannot reject H_0 (the die is fair) in favor of H_1 (the die is unfair). Said differently, the data do not provide sufficient evidence to reject H_0 (the die is fair) in favor of H_1 (the die is unfair).

No Reproduction Image Allowed

10.10 QUIZ 10

Suppose we are trying to see if the processing speed of two computers is the same or not. To decide, we run a given program 5 times on computer A, and 4 times on computer B. The following are the resulting data on the speed:

Computer A: 2.3, 1.9, 1.9, 2.2, 2.1
Computer B: 1.8, 1.9, 2.4, 1.8

a) Write code to compute a p-value according to the most appropriate method in section 5. Report the p-value.

b) At $\alpha = 0.05$, what is the conclusion? (i.e., what can you say about the speed of the two computers)?

c) Write code to compute a p-value according to a 2-sample t-test. Report the p-value.

Solution

```
# a
y <- c(2.3, 1.9, 1.9, 2.2, 2.1, 1.8, 1.9, 2.4, 1.8)
x <- c(1, 1, 1, 1, 1, 2, 2, 2, 2)
aov.1 <- aov(y ~ as.factor(x))
summary(aov.1)
```

```
             Df Sum Sq Mean Sq F value Pr(>F)
as.factor(x)  1  0.025  0.0245    0.46   0.52
Residuals     7  0.376  0.0536
```

b) Because p-value $= 0.52 > \alpha$, we cannot say anything about whether the two true average computer speeds are the same or not.

```
# c
t.test(c(2.3, 1.9, 1.9, 2.2, 2.1), c(1.8, 1.9, 2.4, 1.8),
       alternative = "two.sided")
```

```
Welch Two Sample t-test

data:  c(2.3, 1.9, 1.9, 2.2, 2.1) and c(1.8, 1.9, 2.4, 1.8)
t = 0.64, df = 4.8, p-value = 0.6
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.3228  0.5328
sample estimates:
mean of x mean of y
    2.080     1.975
```

Moral: The F-test, for the case of 2 populations is equivalent to a 2-sample, 2-sided t.test (with equal variance assumption). In other words, the F-test is a multi-population generalization of the t-test.

Not for Distribution Usage Allowed

10.11 QUIZ 11

Confidence intervals are designed to cover a population parameter some percentage of the time. Prediction intervals are designed to cover the true individual y values some percentage of time. In this quiz, we want to illustrate that CIs DO NOT cover the true individual y values the correct percentage of time. To that end, write code to

a) Make 100 cases of data on (x, y), where x is from a uniform distribution between (-1, 1), and $y = 2 \cdot x + e$, where e (for error) is from a normal distribution with $\mu = 0$, $\sigma = 0.1$.

b) Perform simple linear regression of y on x, and compute the CI for all the cases.

c) Count the number of times that the CI covers the true y values

Solution

```
# a
x <- runif(100, -1, 1)
y <- 2*x + rnorm(100, 0, 0.1)

# b
lm.1 <- lm(y ~ x)
CI <- predict(lm.1, interval = "confidence", level = 0.95)

# c
cnt <- 0
for (i in 1:100) {
  if (CI[i,2] <= y[i] && CI[i,3] >= y[i])
    cnt = cnt + 1
  }
cnt

[1] 25
```

Moral: CIs do not cover the true y values the correct percentage of time.

No Reproduction Allowed