Chapter 4

BPR

6/16/2020

Figure 4.1. Testing the R installation.

# This chunk contains the commands we used to test our R installation.  
# You should copy and paste these lines, one at a time, into R.  
c(1, 2, 3, 4, 5) -> integers  
integers

## [1] 1 2 3 4 5

mean(integers)

## [1] 3

Figures 4.4 - 4.7: These come from a separate file that you create.

Code Snippet 4.1.

# The name in the previous line is for "Code Snippet 4.1." I'll follow this   
# convention all the way through this document. "F4.n" will be for code used  
# to produce a figure, but which is not discussed in the text. "T4.n" will be  
# used for code that produces a table.  
  
# Read some sample data, and store it in a variable. Notice that the name of   
# the variable, "sample\_data\_df" describes the data (sample data) and   
# describes the type of data ("data frame").  
read.csv("https://raw.githubusercontent.com/barneyricca/CSRforSS/master/Data/sample.csv", # URL of data file  
 header = TRUE) -> # The first row has the column headers  
 sample\_data\_df # Name of the variable that stores the data  
sample\_data\_df # Display the data

## x y  
## 1 1 2.1  
## 2 2 4.3  
## 3 3 6.1  
## 4 4 8.3  
## 5 5 10.1  
## 6 6 12.3  
## 7 7 14.1

lm(y ~ x, # Create a linear model (a.k.a. regression)  
 # with y as the dependent variable and x  
 # x as the independent variable.  
 data = sample\_data\_df) # Use x and y columns from sample\_data\_df

##   
## Call:  
## lm(formula = y ~ x, data = sample\_data\_df)  
##   
## Coefficients:  
## (Intercept) x   
## 0.1857 2.0000

This code shows up as Figure 4.8. Sample code chunk.

read.csv("https://raw.githubusercontent.com/barneyricca/CSRforSS/master/Data/sample.csv", # URL of data file  
 header = TRUE) -> # The first row has the column headers  
 sample\_data\_df # Name of the variable that stores the data  
sample\_data\_df # Display the data

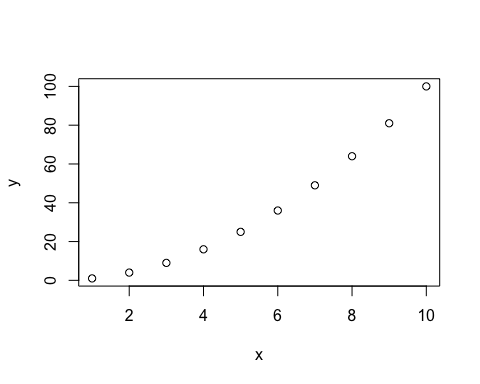
## x y  
## 1 1 2.1  
## 2 2 4.3  
## 3 3 6.1  
## 4 4 8.3  
## 5 5 10.1  
## 6 6 12.3  
## 7 7 14.1

lm(y~x, # Create a linear model (a.k.a. regression)  
 # with y as the dependent variable and x  
 # x as the independent variable.  
 data = sample\_data\_df) # Use x and y columns from sample\_data\_df

##   
## Call:  
## lm(formula = y ~ x, data = sample\_data\_df)  
##   
## Coefficients:  
## (Intercept) x   
## 0.1857 2.0000

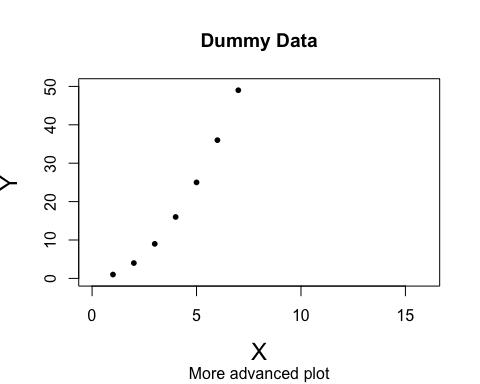
Code Snippet 4.2 and Figure 4.9

1:10 -> x # Put some data (the integers 1 to 10) into variable x.  
x^2 -> y # Put the square of x into y.  
plot(x,y) # Make a graph of y versus x.



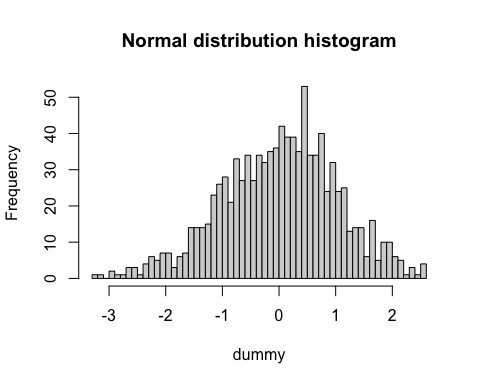
Code Snippet 4.3 and Figure 4.10. A “better” plot.

1:10 -> x # Put some data into variable x  
x^2 -> y # Put the square of x into y  
plot(x, y,  
 main = "Dummy Data", # Change the title above the plot  
 sub = "More advanced plot", # Subtitle  
 xlab = "X", # set the x-axis label  
 ylab = "Y", # set the y-axis label  
 cex.lab = 1.5, # make the labels 50% bigger  
 xlim = c(0,16), # set the x-axis range   
 ylim = c(0,50), # set the y-axis range  
 pch = 16, # Use a solid dot, not an open circle  
 cex = 0.8) # make the dots 80% of normal size



Code Snippet 4.4 and Figure 4.11

rnorm(n = 1000, # Draw 1000 times from a normal distribution  
 mean = 0, # with a mean of zero  
 sd = 1) -> # and a standard deviation of 1  
 dummy # and assign it to variable dummy  
hist(dummy, # Make a histogram of variable dummy  
 breaks = 50, # using 50 bins  
 main = "Normal distribution histogram")



Code Snippet 4.5 and Table 4.2.

cbind(1:5, 6:10, 111:115) -> # Combine 1:5, 6:10, and 111;111  
 # as three columns ("column bind")  
 bad\_table # and store in the variable "bad\_table".  
bad\_table # Print an ugly table

## [,1] [,2] [,3]  
## [1,] 1 6 111  
## [2,] 2 7 112  
## [3,] 3 8 113  
## [4,] 4 9 114  
## [5,] 5 10 115

kable(x = bad\_table, # Print bad\_table  
 format = "simple", # No shading  
 row.names = FALSE, # No row names  
 col.names = c("A", "B", "C"), # Add (bold) column names  
 align = 'c', # Center the numbers  
 caption = "Output") # Caption above the table

Output

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 1 | 6 | 111 |
| 2 | 7 | 112 |
| 3 | 8 | 113 |
| 4 | 9 | 114 |
| 5 | 10 | 115 |

Code Snippet 4.6

2 + 3 # Addition (2 plut 3 = 5)

## [1] 5

8 - 5 # Subtraction (8 minus 5 = 3)

## [1] 3

8 \* 7 # Multiplication (8 times 7 = 56)

## [1] 56

16 / 2 # Division (16 divided by 2 = 8)

## [1] 8

17 %/% 2 # Integer division quotient. 2 goes into 17 8 times

## [1] 8

17 %% 2 # modulo, a.k.a. "remainder." 17/2 has remainder 1.

## [1] 1

sqrt(81) # Square root of 81

## [1] 9

2 ^ 5 # Power (2 raised to the 5th power)

## [1] 32

exp(3) # Exponential

## [1] 20.08554

log(8, 2) # log base 2 of 8

## [1] 3

log(1000, 3) # log base 3 of 1000

## [1] 6.28771

mean(1:10) # Find the mean of the integers 1 to 10

## [1] 5.5

sd(1:10) # Find the standard deviation of the integers 1 to 10

## [1] 3.02765

median(1:10) # Find the median of the integers 1 to 10

## [1] 5.5

min(1:10) # Find the minimum of the integers 1 to 10

## [1] 1

max(1:10) # Find the maximum of the integers 1 to 10

## [1] 10

Code Snippet 4.7

1:15 -> data1 # Put the integers 1 through 15 into data1  
seq(from = 2, # Put a sequence, starting at 2...  
 to = 30, # ...going up to 30...  
 by = 2) -> # ...counting by twos...  
 data2 # ...into the variable data2  
data1 # Display data1 (i.e., 1 through 15)

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

data2 # Display data2 (2, 4, 6, ... 30)

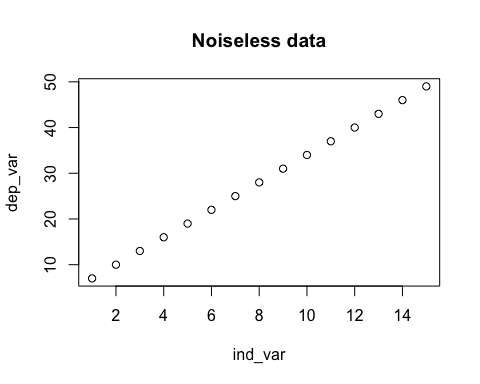
## [1] 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30

data1 + data2 # Add these together and display

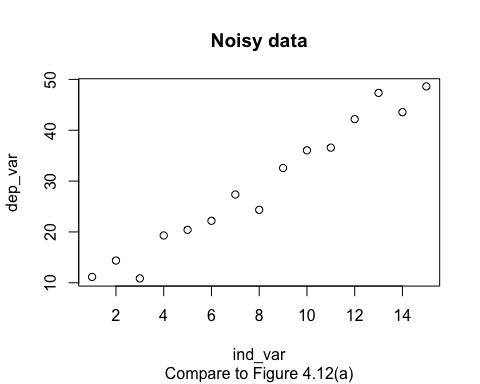
## [1] 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45

Code Snippet 4.8 and Figure 4.12(a)

1:15 -> # Put the integers 1:15 into  
 ind\_var # the independent variable  
   
3 \* ind\_var + 4 -> dep\_var # Use a linear relationship to create the  
 # dependent variable  
plot(ind\_var, dep\_var, # Make a scatterplot  
 main = "Noiseless data") # with a title.

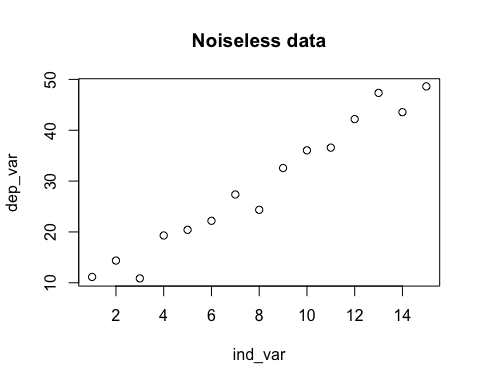


set.seed(seed = 42) # Look at the "Sampling" section in Chapter 4.  
runif(n = 15, # Get 15 random numbers  
 min = -5, # from -5  
 max = 5) + # to 5  
 dep\_var -> # add them to the original  
 dep\_var # and store the results back in  
 # the original dependent variable.  
plot(ind\_var, dep\_var, # Make a scatterplot  
 main = "Noisy data", # with a title and subtitle:  
 sub = "Compare to Figure 4.12(a)")



Code Snippet 4.9, Tables 4.3 and 4.4, and Figure 4.13

1:15 -> # Put the integers 1:15 into  
 ind\_var # the independent variable  
   
3 \* ind\_var + 4 -> dep\_var # Use a linear relationship to create  
 # the dependent variable  
set.seed(seed = 42) # Look at the "Sampling" section in   
 # Chapter 4.  
runif(n = 15, # Get 15 random numbers  
 min = -5, # from -5  
 max = 5) + # to 5  
 dep\_var -> # add them to the original  
 dep\_var # and store the results back in  
 # the original dependent variable.  
plot(ind\_var, dep\_var, # Scatterplot  
 main = "Noiseless data") # with a title



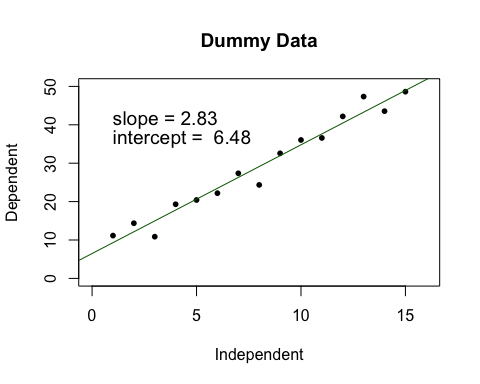
lm(dep\_var ~ ind\_var) -> # Regress dependent variable on the  
 # independent variable  
 y\_lm # and store the results in y\_lm  
summary(y\_lm) # Summarize the linear model (ugly)

##   
## Call:  
## lm(formula = dep\_var ~ ind\_var)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -4.7812 -1.1594 0.6106 1.6199 4.0616   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 6.4764 1.3682 4.734 0.000391  
## ind\_var 2.8314 0.1505 18.816 8.19e-11  
##   
## Residual standard error: 2.518 on 13 degrees of freedom  
## Multiple R-squared: 0.9646, Adjusted R-squared: 0.9619   
## F-statistic: 354.1 on 1 and 13 DF, p-value: 8.189e-11

kable(coefficients(summary(y\_lm)), # Nicer printing of the regression  
 # coefficients,  
 digits = 3) # rounded to 3 digits

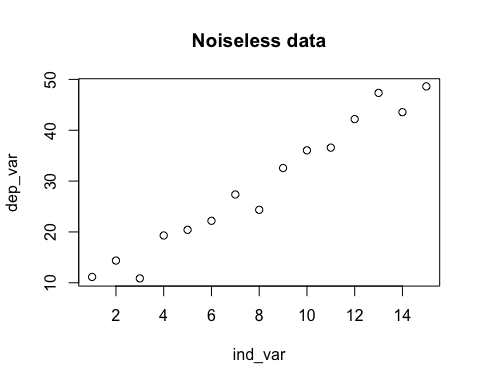
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Estimate | Std. Error | t value | Pr(>|t|) |
| (Intercept) | 6.476 | 1.368 | 4.734 | 0 |
| ind\_var | 2.831 | 0.150 | 18.816 | 0 |

{ # Put the scatterplot, line & text  
 # all on the same plot  
 plot(ind\_var, dep\_var, # Plot the data  
 main = "Dummy Data", # Change the title above the plot  
 xlab = "Independent", # set the x-axis label  
 ylab = "Dependent", # set the y-axis label  
 xlim = c(0,16), # set the x-axis range   
 ylim = c(0,50), # set the y-axis range  
 pch = 16, # Use a solid dot, not an open circle  
 cex = 0.8) # make the dots 80% normal size  
 abline(a = y\_lm$coef[1], # Add a line to the plot, with  
 # intercept from the linear model,  
 b = y\_lm$coef[2], # slope from the linear model,  
 col = "darkgreen") # and using a dark green color.  
 text(1, 40, # Put some text info on the plot @ (1,40)  
 labels = paste( # The label will have, pasted together,  
 "slope =", # "slope ="  
 round(y\_lm$coef[2], # and the value of the slope from the  
 # regression,  
 digits = 2)), # rounded to 2 decimal places.  
 adj = c(0,0), # Lower left point at (1,40) on graph  
 cex = 1.2) # 20% bigger text  
 text(1, 35, # Put some more text on the plot  
 labels = paste( # Paste together  
 "intercept = ", # "intercept = "  
 round(y\_lm$coef[1], # the value of the intercept from the  
 # regression  
 digits = 2)), # rounded to two decimal places.  
 adj = c(0,0),  
 cex = 1.2) # 20% bigger text  
}

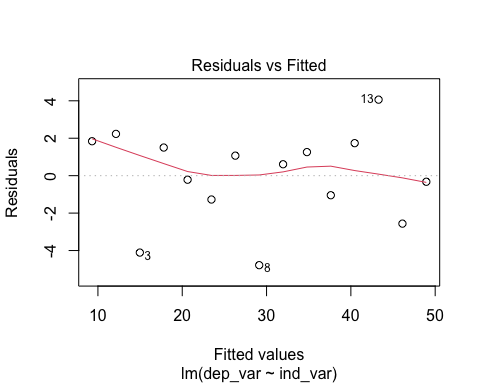


Code Snippet 4.10 and Figure 4.14

1:15 -> # Put the integers 1:15 into  
 ind\_var # the independent variable  
   
3 \* ind\_var + 4 -> dep\_var # Use a linear relationship to create  
 # the dependent variable  
set.seed(seed = 42) # Look at the "Sampling" section in   
 # Chapter 4.  
runif(n = 15, # Get 15 random numbers  
 min = -5, # from -5  
 max = 5) + # to 5  
 dep\_var -> # add them to the original  
 dep\_var # and store the results back in  
 # the original dependent variable.  
plot(ind\_var, dep\_var, # Scatterplot  
 main = "Noiseless data") # with a title



lm(dep\_var ~ ind\_var) -> # Regress dependent variable on the  
 # independent variable  
 y\_lm # and store the results in y\_lm  
  
plot(y\_lm, # Plot the results of the linear model.  
 # Notice that the plot() command is  
 # pretty smart. It detects that y\_lm  
 # is a linear model and plots  
 # appropriately!  
 which = 1) # There are four plots; use the first.



# Type ?plot.lm at the command prompt  
 # in the console, and press <Enter>  
 # to see what the others are.

Code Snippet 4.11

read.csv(  
 "https://raw.githubusercontent.com/barneyricca/CSRforSS/master/Data/ANOVA.csv", # Read in some synthetic data;  
 header = TRUE, # First row has the column names;  
 stringsAsFactors = FALSE) -> # Don't worry...just include this.  
 anova\_data # Store the data.  
  
# Let's find the different in means between the groups, using the next  
# two commands. The second command takes a bit of unpacking, but it is  
# a useful trick. We use [] to get particular entries. In the first mean()  
# we only want the Score[] which have the corresponding Group equal to  
# "A". (Notice the double equal sign. That's important!) The second  
# mean() does the same thing, but uses Group "B".  
print("Difference in means between groups A & B:")

## [1] "Difference in means between groups A & B:"

mean(anova\_data$Score[which(anova\_data$Group == "A")]) -  
 mean(anova\_data$Score[which(anova\_data$Group == "B")])

## [1] -0.2911227

# ANOVA results:  
summary(  
 aov(Score ~ Group, # Standard ANOVA. The Score depends  
 # on the Group.  
 data = anova\_data)) # Print the ANOVA results

## Df Sum Sq Mean Sq F value Pr(>F)  
## Group 1 0.888 0.8879 2.151 0.15  
## Residuals 40 16.511 0.4128

# Linear model results:  
summary(  
 lm(Score ~ Group, # Do an ANOVA via a linear model!  
 data = anova\_data)) # Use the same data as above.

##   
## Call:  
## lm(formula = Score ~ Group, data = anova\_data)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1.93027 -0.35274 0.06629 0.40782 1.43473   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 5.5432 0.1437 38.585 <2e-16  
## GroupB 0.2911 0.1985 1.467 0.15  
##   
## Residual standard error: 0.6425 on 40 degrees of freedom  
## Multiple R-squared: 0.05103, Adjusted R-squared: 0.02731   
## F-statistic: 2.151 on 1 and 40 DF, p-value: 0.1503

# Hey, look at that! The slope is equal to the difference in group means  
# and the p-value is the same as noted in the aov().

Code Snippet 4.12

set.seed(412)  
sample(LETTERS[1:4], # From the first four letters,  
 size = 2, # pick two,  
 replace = FALSE, # do NOT replace between picks,  
 prob = c(1/4, 1/4, 1/4, 1/4)) # use equal probabilities.

## [1] "C" "A"

## [1] "C" "A"  
  
set.seed(412)  
sample(LETTERS[1:4], # replace = FALSE and equal  
 size = 2) # probabilities are the defaults

## [1] "A" "D"

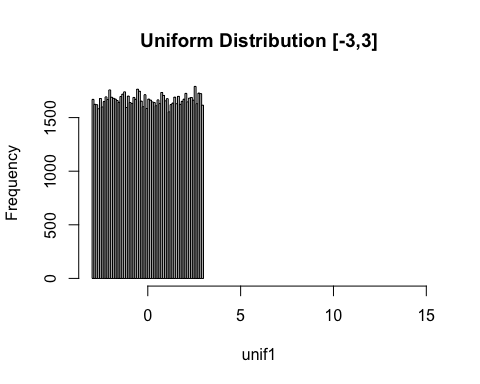
## [1] "A" "D" # But the results are different!  
#  
#  
# A note about the different results: Really, the default value in sample  
# is "prob = NULL" which is implemented in the underlying code slightly   
# differently than listing the four equal probabilities. Look at this:  
#  
set.seed(412)  
sample(LETTERS[1:4], # replace = FALSE and equal  
 size = 2, # probabilities are the defaults  
 prob = NULL) # The default is "NULL."

## [1] "A" "D"

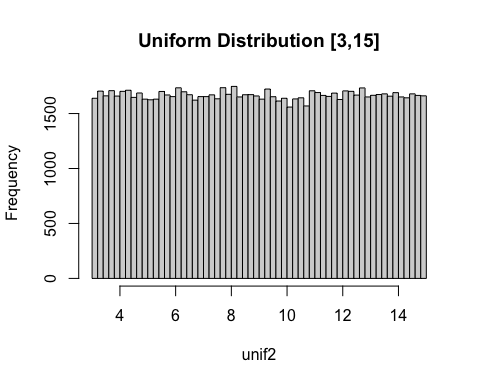
## [1] "A" "D" # Now the results are the same.

Code Snippet 4.13 and Figures 4.15 and 4.16

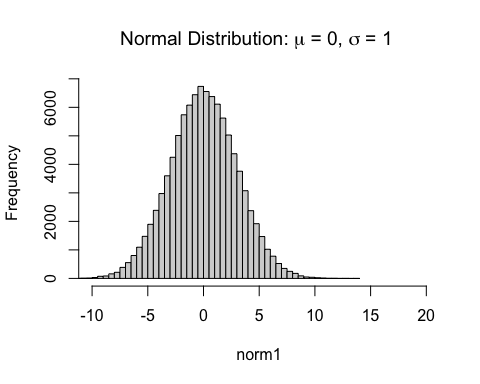
set.seed(seed = 413)  
runif(n = 1e5, # Draw 100,000 samples uniformly  
 min = -3, # between -3...  
 max = 3) -> # ...and 3. Store the results...  
 unif1 # ...in variable unif1.  
hist(unif1, # Make a histogram of the data in variable unif1.  
 breaks = 50, # Divide the data into 50 bins  
 xlim = c(-3, 15), # Draw graph from x = -3 to x = 15. (See next example.)  
 main = "Uniform Distribution [-3,3]") # Put a title on the graph



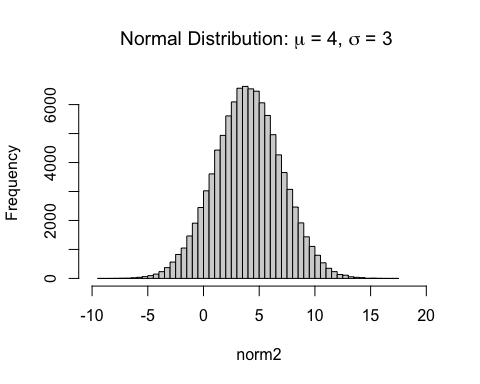
runif(n = 1e5,  
 min = 3,  
 max = 15) -> # Use a different maximum value  
 unif2  
hist(unif2,  
 breaks = 50,  
 main = "Uniform Distribution [3,15]")



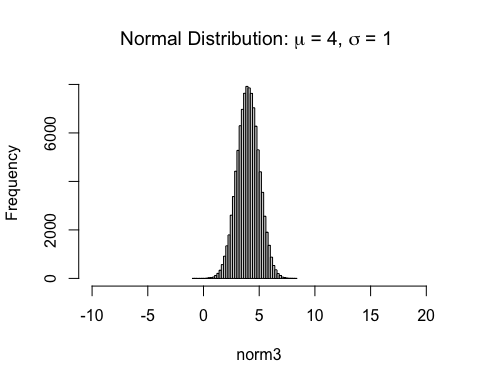
rnorm(n = 1e5, # Draw 100,000 samples from a normal distribution...  
 mean = 0, # ...with a mean of 0...  
 sd = 3) -> # ...and standard deviation of 3. Store the results...  
 norm1 # ...in variable norm1.  
hist(norm1, # Make a histogram.  
 breaks = 50, #   
 main = expression( # "expression" allows for math and Greek characters.  
 paste( # "paste" all this stuff together:  
 "Normal Distribution: ", # String  
 mu, # Greek letter mu (because of expression)  
 " = 0, ", # Another string  
 sigma, # Greek letter sigma  
 " = 1 ")), # Another string  
 xlim = c(-10, 20)) # Set the limits (see the next two)



rnorm(n = 1e5,  
 mean = 4,  
 sd = 3) ->   
 norm2  
hist(norm2,  
 breaks = 50,  
 main = expression(paste("Normal Distribution: ",  
 mu,  
 " = 4, ",  
 sigma,  
 " = 3 ")),  
 xlim = c(-10, 20))



rnorm(n = 1e5,  
 mean = 4,  
 sd = 1) ->   
 norm3  
hist(norm3,  
 breaks = 50,  
 main = expression(paste("Normal Distribution: ",  
 mu,  
 " = 4, ",  
 sigma,  
 " = 1 ")),  
 xlim = c(-10, 20))



Code Snippet 4.14: Fisher’s experiment

set.seed(414)  
8 -> cups # Fisher proposed 8 cups  
cups/2 -> cd2 # Half the cups will be tea and half milk  
# First, randomly set up the cups, "M" for milk-first, "T" for  
# tea-first.  
sample( # Create a random sample...  
 rep(c("M","T"), # ...from a collection of Milk and Tea...  
 each = cd2), # ...using 4 of each.  
 cups, # Get a total of 8 cups.  
 replace = FALSE) -> # Don't reuse the cups.   
 treatment # Put that into the variable treatment  
  
1e6 -> N # Repeat the experiment 1 million times  
  
# First, it will make life easier to create a "function." The next command  
# allows us to treat "guess1" just like any other built-in R function.  
# (In other words, we are expanding R's capabilities.)  
guess1 <- function(treat) { # Define a function.  
 # The function "guess1" takes the arrangement of cups (the variable treat),  
 # makes a random guess and returns how many cups were correctly identified  
 # by the guess.  
  
 length(treat) -> cups # How many cups are in the treatment?  
 cups / 2 -> cd2 # Half the cups are milk, half are tea.  
  
 sample( # This sample caommand should look familiar!  
 rep(c("M","T"), # Hint: Look up about a dozen lines  
 each = cd2),  
 cups,   
 replace = FALSE) ->   
 attempt # The guess is stored in the variable attemp  
  
 # The next takes a bit of unpacking. The which() command returns the  
 # positions of all those entries that meet some criteria. In this case,  
 # the criteria is "treat == attempt"; NOTICE THE DOUBLE EQUAL SIGN!  
 # The == means "if the first thing is equal to the second" then the  
 # criteria is met. (In computer science speak, we'll say "return TRUE.")  
 # length() counts how many things met which()'s criteria.  
 # Whew!  
 return( # Return the results to whatever called the function  
 length(  
 which(treat == attempt)))  
}  
  
# The next is part 1, the large number of guesses. (See the text.)  
replicate(N, # Do the next line N times  
 guess1(treatment)) -> # Make the guess!  
 results # Store in a variable called results.  
  
table(results) -> results\_tab # Create a table of the results  
 # These will be in order from 0 correct  
 # up to all correct.  
  
# Now, for the answer: What fraction of results guessed all the cups  
# correctly?   
results\_tab[length(results\_tab)] / N

## 8   
## 0.014181

Figure 4.16. Indicator of an error. This next chunk intentionally has an error in it, so you need to correct the error (or remove the line) before knitting.

# To knit this file, you would need to use the correct version of the next  
# command. Otherwise, the error will cause the knitting to stop.  
# Hence, before knitting, put a hashtag in front of the next command.  
  
lm(y ~ x, data = sample\_data\_df)

##   
## Call:  
## lm(formula = y ~ x, data = sample\_data\_df)  
##   
## Coefficients:  
## (Intercept) x   
## 0.1857 2.0000