R for Introductory Statistics

Bret Larget

September 25, 2002

The aim of this document is to help you, an undergraduate student in an introductory statistics course, learn to use the software R as part of your learning of statistics. If you find that it reads like the rough draft of something that could be more useful and better written, that is because it is a rough draft that could be more useful and better written. This document will evolve on a weekly basis as the semester progresses. I will add new material as we cover it in class and edit old material based on feedback from you to make it clearer for you and future students. I suggest that you do not print new versions but merely replace your electronic copy from time to time. Good luck as you begin your quest to master introductory statistical concepts and their application!

1 What is R?

R is powerful software for interacting with data. With R you can create sophisticated graphs, you can carryout statistical analyses, and you can create and run simulations. R is also a programming language with an extensive set of built-in functions, so you can, with some experience, extend the language and write your own code to build your own statistical tools. Advanced users can even incorporate functions written in other languages, such as C, C++, and Fortran.

The S language has been around for more than twenty years and has been the most widely-used statistical software in departments of statistics for most of that time, first as S and then as the commercially available S-PLUS. R is an open source implementation of the S language that is now a viable alternative to S-PLUS, and in fact, has many advantages. A core team of statisticians and many other contributors work to update and improve R and to make versions that run well under all of the most popular operating systems. Most importantly to you, R is free, high-quality statistical software that will be useful as you learn statistics even though it is also a first-rate tool for professional statisticians.

Why use R for introductory statistics?

There are several reasons that make R an excellent choice of statistical software for an introductory statistical course. First, R is free and available on the Web. You can use it on your home computers and are not tied to campus labs. Second, R is powerful, widely-used software. The knowledge of R you gain during the course potentially translates to a marketable skill. You will learn to use a tool that has many practical uses outside the classroom. Third, even though it is not the simplest statistical software, the basics are easy enough to master that learning to use R need not interfere overly much with learning the statistical concepts encountered in an introductory course. Fourth, did I mention that it is free and you can use it at home?

The primary drawback to using R in an introductory course is that most existing documentation for R

is written for an audience that is knowledgable about statistics and has experience with other statistical computing programs. In contrast, this document intends to make R accessible to the typical student in an introductory statistics course who is new to both statistical concepts and statistical computing. The aim is to teach you how to install R on your home computer and to teach you to use R to learn the statistical concepts usually included in an introductory course with explanations and examples aimed at the appropriate level. This document purposely does not attempt to teach you about R's advanced features. The intention is to teach you enough R to enhance your learning of introductory statistics and to point you in the direction of more information should you find a desire to learn more.

2 Installing R

Installing R on your computer is simple if you have clear directions you can find that tell you exactly what to do in a way that is easy to understand. Directions exist at the R website (`http://cran.r-project.org/`) for installing R, but many students may have difficulty determining which files they need to download and then how to install them. Here are more explicit instructions that tell you what to do.

Obtaining the software

There are two options for installing the software: downloading it from the Web or installing from a prepared CD. If you have a fast Internet connection (a direct campus connection, cable modem, or DSL), I recommend that you download the software. If you have no Internet connection or are limited to a regular modem I recommend that you borrow a CD from me. In either case, there is only one file that you need to obtain (different depending on the operating system). Running this file begins the installation process which is straight-forward.

Downloading R from the Web

Go the R homepage at `http://cran.us.r-project.org/`.

Windows (95 or later) Click on the link `Windows (95 and later)`, then click on the link `base/`, and finally click on `SetupR.exe` which begins the download. After the download is complete, double click on the downloaded file and follow the on screen installation instructions.

Macintosh Click on the link `MacOS (System 8.6 to 9.1 and MacOS X)`, then click on the link `base/`, and finally click on `rm151.sit` which begins the download. After the download is complete, double click on the downloaded file and follow the on screen installation instructions.

Loading R from a CD

Insert the CD into the drive, open the CD (from My Computer in Windows) and double click on the `SetupR.exe` icon to begin installation. Follow the on screen installation instructions.

3 A First Session with R

Starting and Quitting

Because most students in the course are running R under Windows, these instructions will assume that you are using the Windows version. (Apologies to the few Mac users.) I actually run R most often under Linux. If you notice differences in what I write and how R actually performs under Windows, please let me know.

Begin R by double clicking on the shortcut (if you added a shortcut to your Desktop) or from the Start button followed by the Program menu. R will open with a command window with a prompt `>` that awaits your first command. R is a command line program. You interact with the software by typing in commands which the program then interprets and acts on.

When you are done with your R session, you can quit from the `File` menu or by typing `q()` in the command window at the prompt `>`.

Several Examples

Here is a demonstration of several functions you will use frequently. A later section will provide more details. In these examples I will look at a data set from the textbook *Statistics for the Life Sciences*, second edition, by Samuels and Witmer. The data set consists of measurements of glucose concentration in the anterior chamber of the right eye of 31 dogs, measured for each dog as a percentage of the serum glucose concentration (page 31, problem 2.9). At this point we are not doing any statistical thinking, but are merely learning the nuts and bolts of using R.

Entering data as a vector The easiest way to enter small data sets is with the function `c` that *concatenates* numbers (or vectors) together. For example, we could create an object named 'glucose' containing the 31 measures as follows.

```
R> glucose <- c(81, 85, 93, 93, 99, 76, 75, 84, 78, 84, 81, 82,
+      89, 81, 96, 82, 74, 70, 84, 86, 80, 70, 131, 75, 88, 102,
+      115, 89, 82, 79, 106)
```

The symbol '`<-`', a less than sign followed immediately without a space by a hyphen, looks like an arrow. It is the assignment operator and creates an object with the name on the left of the `<-` whose value is the evaluation of the remainder of the command.

You do not need to type in the '+' symbols, which are prompts that indicates R is waiting for a command to be completed. This command was so long it did not fit onto a single line, so R wrapped to the following line. You may also press [Enter] to continue a command on the next line.

Warning—if you type a '(' and then do not complete the command by typing a ')', R will continue to wait for the command to be completed and show a string of '+' prompts even if you continue to press [Enter]. If you get in trouble, you can press [Esc], the Escape key to break back to a regular prompt.

Alternatively, you can enter data into R by first creating a text file and then reading in the data. This is especially useful for larger data sets or when there are more than one variable. A later section will explain how to do this.

Let's use R to create a histogram, using the command `hist`.

```
R> hist(glucose)
```

Here is a stem-and-leaf diagram.

```
R> stem(glucose)
```

```
  The decimal point is 1 digit(s) to the right of the |

   7 | 00455689
   8 | 011122244456899
   9 | 3369
  10 | 26
  11 | 5
  12 |
  13 | 1
```

To calculate measures of center, use the functions `mean` and `median`.

```
R> mean(glucose)
```

```
[1] 86.7742
```

```
R> median(glucose)
```

```
[1] 84
```

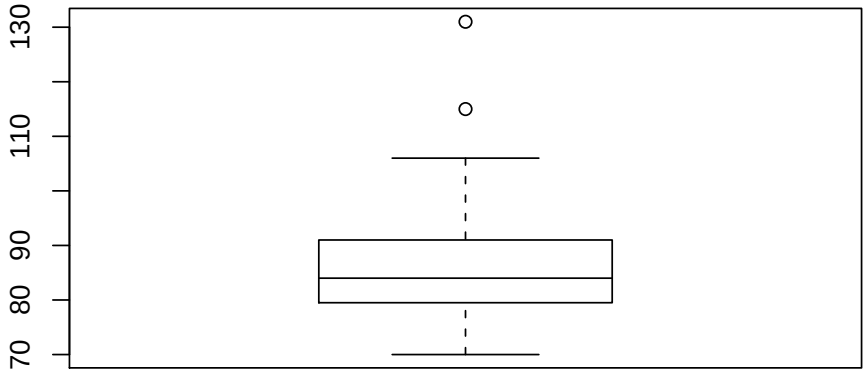
The `quantile` function computes the five number summary, minimum, first quartile, median, third quartile, maximum, by default.

```
R> quantile(glucose)
```

```
   0%   25%   50%   75%  100%
 70.0  79.5  84.0  91.0 131.0
```

The five number summary may be shown graphically as a modified boxplot.

```
R> boxplot(glucose)
```



Boxplots show less information than histograms. Their true utility is for making comparisons between different distributions. Here is an example from Samuels and Witmer, pages 22–24.

```
R> growthDark <- c(15, 20, 11, 30, 33, 22, 37, 20, 29, 35, 8, 10,
+      15, 25)
R> growthLight <- c(10, 15, 22, 25, 9, 15, 4, 11, 20, 21, 27, 20,
+      10, 20)
R> boxplot(list(dark = growthDark, light = growthLight))
```

4 Some R Nuts and Bolts

New users of complex software typically learn what they need to know to do what they need to do and try not to learn anything extra until they need to. A complex program like R is like a big black box with many complicated buttons and dials to control its operation. For new users especially, most of the details of the box can (and should) remain hidden. The program can work well enough for simple tasks with just a few learned commands More advanced users will invariably need to learn and understand more of what happens in the box and will want to master more ways to control its operation, but even highly advanced users are often content to leave details unexamined. (Do I need to know the algorithm for finding the median? Probably not. I am content to trust that it works.)

Getting Help

The first place to look for more information than this document provides is the documentation distributed with the software. You can access the help by typing the command `help.start()`. In addition, the Windows version has a Help menu. From this you can find several choices of manuals, all of which the typical introductory statistics student may find intimidating, and a mechanism to search for help on a specific topic. As you become a more experienced user, this source of help will become more accessible and useful. You can get help on a specific function by typing `?function-name` at the prompt. For example, `?hist` provides many more details on how to use `hist` using all of its available options.

Generally speaking, the guiding principle in creating this introduction to R is to show you only what you need to know to use R effectively as part of an introductory statistics course and no more. For example, the documentation on using `hist` contained here is far less complete than the distributed documentation but should be highly accessible to introductory statistics students.

However, at times, I will violate to this principle and include material that is extraneous, with the judgment that the material might be helpful to some students.

Objects in R

In R, everything is an *object*. For your purposes, you can classify the objects in R as *functions* and *non-functions*.

There are two types of commands that R understands. If you type in the name of an object, R will print (display) the object. If you type in a function call, R will execute the function and (depending on the function), print results. (The third type of command you will type in is an expression with poor syntax that R cannot parse and understand. When you do this, R will print an error message.)

What can be confusing to beginners is that functions themseleves are objects. Function calls are indicated with parentheses that can contain input. Typing a function name without parentheses displays the function, but does not execute it. For example the function `q` will quit the program if executed. To actually quit, you would type `q()`. Typing `q` shows the function that quits the program, but does not quit.

```
R> q
```

```
function (save = "default", status = 0, runLast = TRUE)
.Internal(quit(save, status, runLast))
```

The `q` function does little more than call an internal function. Other functions are small programs written in R (more technically, the S language). For example, `median` is a small R program. When you become a more advanced user of R you will want to learn to write your own small programs.

5 Getting Help

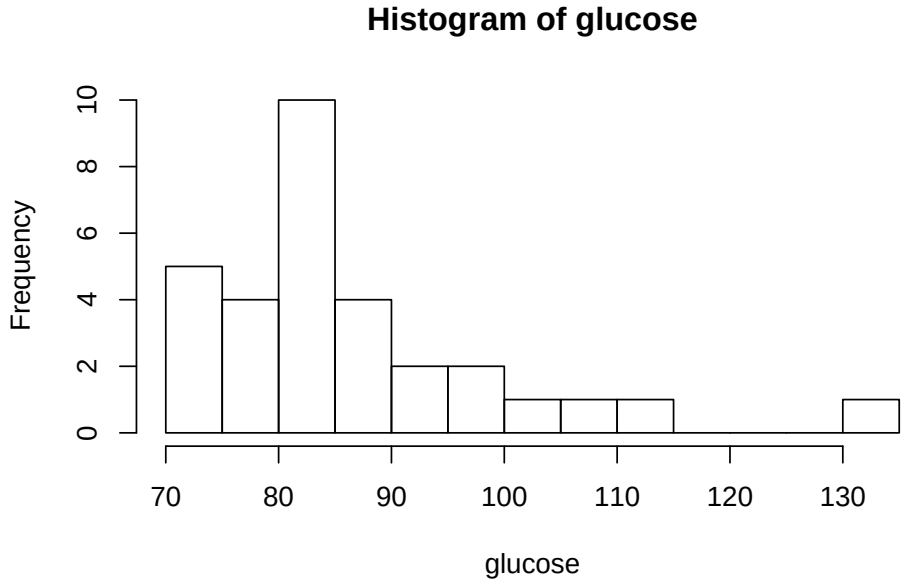
There is complete on-line documentation of R that this section does not attempt to reproduce. To open the help page for a particular function, type a '?' before the name of the function name. For example, to get help about the function `hist`, you would typing `?hist` after the prompt. You may also use the `Help` menu and search for keywords.

6 Elementary Exploratory Data Analysis of One Variable

This section provides several examples of using R functions to graph a single variable and explains how to use some of the parameters associated with these functions to modify the results.

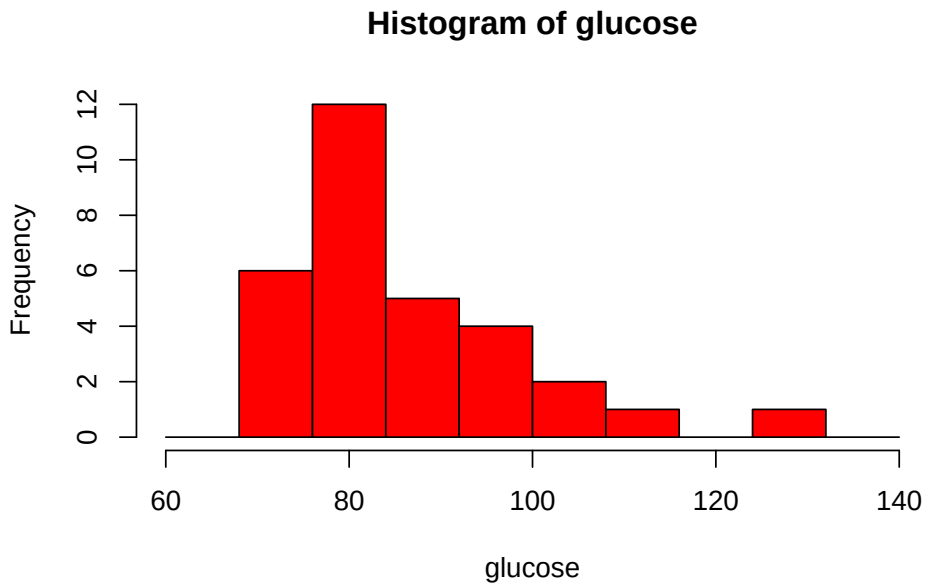
Histograms The `hist` function produces histograms. The example we looked at previously plotted the variable glucose. By default, there were seven classes, each of width ten, ranging from 70 to 140. If we wanted to specify a different number of classes, say 14, we could have typed this.

```
R> hist(glucose, breaks = 14)
```



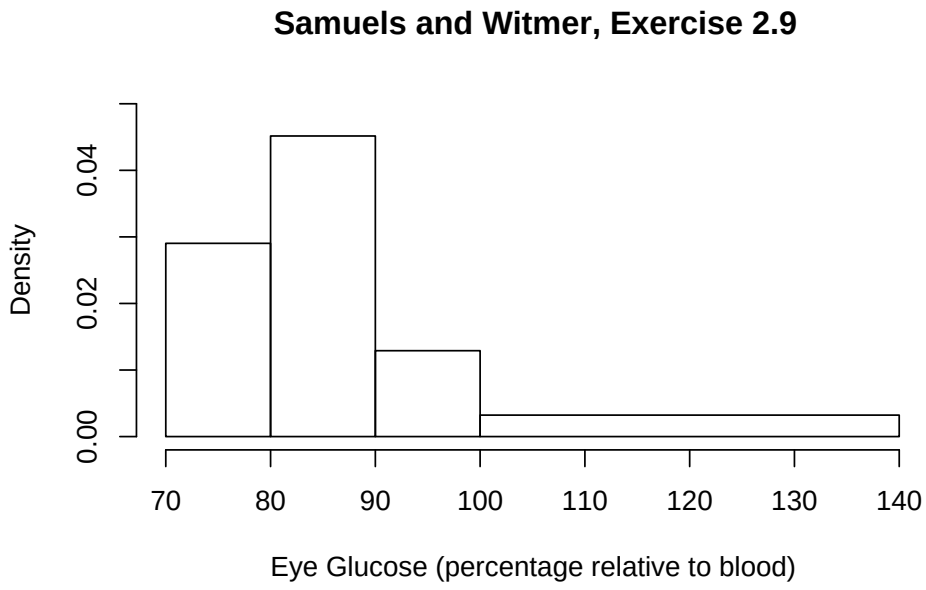
If we wished to manually set breaks to begin at 60, end at 140, and have widths of 8, while shading in the bars red (color 2), we could have done this.

```
R> hist(glucose, breaks = seq(60, 140, by = 8), col = 2)
```



The values of `main`, `xlab`, and `ylab` may be set to change the main title or the labels on the x and y axes. You may also set unequal class widths. R will correctly scale the heights so the areas are proportional and relative frequencies are equal to density times class width. You may set the range of the axes with `xlim` and `ylim`.

```
R> hist(glucose, breaks = c(70, 80, 90, 100, 140), xlab = "Eye Glucose (percentage relative to blood)",
+      main = "Samuels and Witmer, Exercise 2.9", ylim = c(0, 0.05))
```



Stem-and-Leaf Diagrams The function `stem` produces stem-and-leaf diagrams. By default, the function may not round or split stems as you might like. In fact, sometimes the default behavior is to combine two stems to one and to place the leaves of both stems on the same row. Consider this example, the total amount of time each of twenty fruit flies spent preening (in seconds) during a six-minute of observation period.

```
R> preen <- c(34, 24, 10, 16, 52, 76, 33, 31, 46, 24, 18, 26, 57,
+      32, 25, 48, 22, 48, 29, 19)
R> stem(preen)
```

```
  The decimal point is 1 digit(s) to the right of the |

  0 | 0689
  2 | 2445691234
  4 | 68827
  6 | 6
```

The stem-and-leaf diagram is misleading. The middle two stems are clear enough; for example, values range from 22 to 29 and then 31 to 34 in the '2' stem. However, it appears that the maximum value is 66, when, in fact, it is 76. Similarly, it is unclear as to whether the 0 stem shos values in the single digits, teens, or both. We can get around this behavior by using the `scale` parameter. Setting `scale=2` should double the number of stems.

```
R> stem(preen, scale = 2)
```

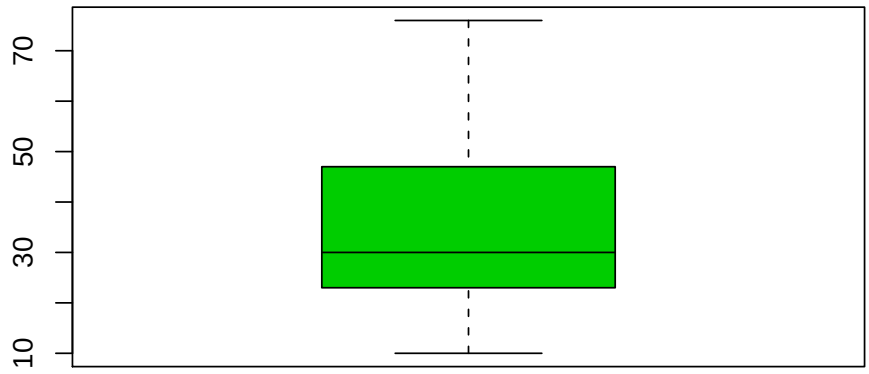
```
  The decimal point is 1 digit(s) to the right of the |

  1 | 0689
```

```
2 | 244569
3 | 1234
4 | 688
5 | 27
6 |
7 | 6
```

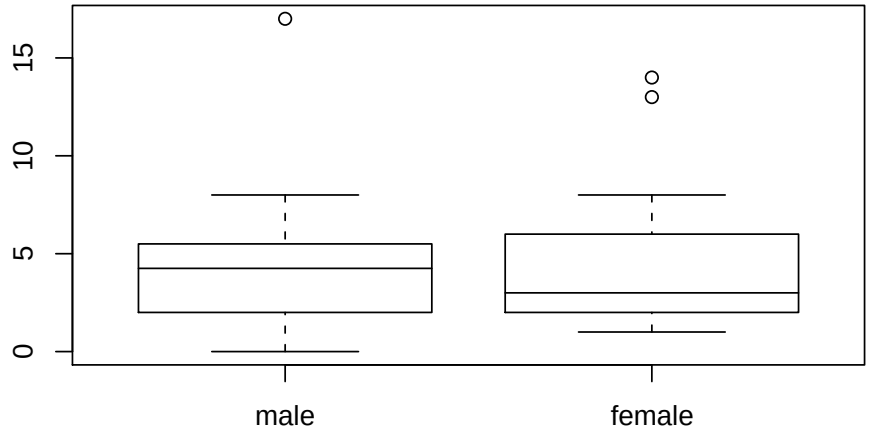
Boxplots Boxplots are constructed using the `boxplot` function. If the argument is one vector, a single boxplot will be drawn. Parallel boxplots may be drawn by providing a list of variables, either directly using the `list` function or as the output of the `split` function which partitions one variable according to the categories of a second (categorical) variable. Here is a boxplot of the preening times from the previous example with the box shaded green.

```
R> boxplot(preen, col = 3)
```



Samuels and Witmer Exercise 2.26 presents self-reported numbers of hours of exercise per week given by 25 college students, 12 men and 13 women. Here is one way to read in the data and make parallel boxplots.

```
R> male <- c(6, 0, 2, 1, 2, 4.5, 8, 3, 17, 4.5, 4, 5)
R> female <- c(5, 13, 3, 2, 6, 14, 3, 1, 1.5, 1.5, 3, 8, 4)
R> boxplot(list(male = male, female = female))
```



The functions `read.table` and `split` would be useful to create parallel boxplots if the data were read in from a file. The function `data.frame` could also be used to create a "data frame" or matrix of the two variables.

For example, you could create a text file `ex2-26.txt` with the values of the two variables like this.

```
hours sex
6     male
0     male
2     male

.
.
.

5     male
5     female
13    female
3     female

.
.
.

4     female
```

Here is how to read it and create a data frame x.

```
R> x <- read.table("ex2-26.txt", header = TRUE)
```

Alternatively, you could create the data frame using the variables entered before. The function `rep` repeats a value a specified number of times.

```
R> hours <- c(male, female)
R> sex <- c(rep("male", 12), rep("female", 13))
R> x <- data.frame(hours = hours, sex = sex)
```

Finally, make the parallel boxplots. The `attach` function adds the names of the variables to the search path.

```
R> attach(x)
R> boxplot(split(hours, sex))
```

Quantitative Summaries R is also useful for numerical summaries of variables. The functions `mean` and `median` compute the mean and median, respectively. The function `fivenum` may be used to find the five-number summary, the minimum, first quartile, median, third quartile, and maximum. The standard deviation may be computed with `sd`. Here are examples of their use with the preening time data.

```
R> mean(preen)

[1] 33.5

R> sd(preen)

[1] 16.31435

R> median(preen)

[1] 30

R> max(preen)

[1] 76

R> min(preen)

[1] 10

R> fivenum(preen)

[1] 10 23 30 47 76
```

Using R to make Simple Calculations Here are several examples of using R to do simple calculations. The colon operator ':' creates a sequence from one integer to another. The bracket operators '[' and ']' are used to take a subset of a variable. Functions such as `sum` and `length` do obvious things.

Sum the numbers from 1 to 10.

```
R> sum(1:10)

[1] 55
```

Find the interquartile range of the preening-time data set. Determine the lower and upper fence values.

```
R> fnum <- fivenum(preen)
R> iqr <- fnum[4] - fnum[2]
R> iqr

[1] 24

R> fnum[2] - 1.5 * iqr

[1] -13

R> fnum[4] + 1.5 * iqr

[1] 83
```

Count the number of observations within one, two, and three standard deviations of the mean for the preening-time data and then report these as percentages. A statement with '<' or '>' returns true or false for each position of an array while '&' means 'and' applied at each position of a vector. A sum of a vector of 'T' and 'F' counts the number that are true.

```
R> m <- mean(preen)
R> s <- sd(preen)
R> sum((m - s < preen) & (preen < m + s))

[1] 15

R> round((sum((m - s < preen) & (preen < m + s))/length(preen) *
+      100))

[1] 75

R> sum((m - 2 * s < preen) & (preen < m + 2 * s))
```

```
[1] 19
```

```
R> round((sum((m - 2 * s < preen) & (preen < m + 2 * s))/length(preen) *
+      100))
```

```
[1] 95
```

```
R> sum((m - 3 * s < preen) & (preen < m + 3 * s))
```

```
[1] 20
```

```
R> round((sum((m - 3 * s < preen) & (preen < m + 3 * s))/length(preen) *
+      100))
```

```
[1] 100
```

7 Probability Distributions

R has a number of built in functions for calculations involving probability distributions, both discrete and continuous. In introductory statistics courses, the binomial and normal distributions are normally introduced early in the semester. Occasionally the Poisson distribution makes an appearance. When the topic changes to statistical inference, the t, chi-square, and F distributions become important.

For each of these distributions (and others), R has four primary functions. Each function has a one letter prefix followed by the root name of the function. The names make mnemonic sense for continuous random variables but are used in both cases. For example `dnorm` is the height of the density of a normal curve while `dbinom` returns the probability of an outcome of a binomial distribution. Here is the complete list: 'd' represents 'density' for continuous random varaibles or 'probability mass function' for discrete random variables; 'p' represents 'probability' and returns the cumulative distribution function value in each case; 'q' represents 'quantile' and is the inverse of the corresponding 'p' function; while 'r' can be used to generate a 'random' sample from a distribution. Below, I will include home-brewed functions with prefix 'g' that are useful for graphing the distributions.

Binomial Distribution The binomial distribution is applicable for counting the number of outcomes of a given type from a prespecified number n independent trials, each with two possible outcomes, and the same probability of the outcome of interest, p. The distribution is completely determined by n and p. The probability mass function is defined as:

$$\Pr\{Y = j\} = \binom{n, j}{p}^{j} (1-p)^{n-j}$$

where

$$\binom{n, j}{=} \frac{n!}{j!(n-j)!}$$

is called a binomial coefficient. (Some textbooks use the notation ${}_nC_j$ instead.) In R, the function `dbinom` returns this probability. There are three required arguments: the value(s) for which to compute the probability (j), the number of trials (n), and the success probability for each trial (p).

For example, here we find the complete distribution when $n = 5$ and $p = 0.1$.

```
R> dbinom(0:5, 5, 0.1)
```

```
[1] 0.59049 0.32805 0.07290 0.00810 0.00045 0.00001
```

If we want to find the single probability of exactly 10 successes in 100 trials with $p = 0.1$, we do this.

```
R> dbinom(10, 100, 0.1)
```

```
[1] 0.1318653
```

The function `pbinom` is useful for summing consecutive binomial probabilities. With $n = 5$ and $p = 0.1$, here are some example calcuations.

$$\begin{aligned} \Pr\{Y \leq 2\} &= \text{pbinom}(2,5,0.1) \doteq 0.99144 \\ \Pr\{Y \geq 3\} &= 1 - \Pr\{Y \leq 2\} = 1 - \text{pbinom}(2,5,0.1) \doteq 0.00856 \\ \Pr\{1 \leq Y \leq 3\} &= \Pr\{Y \leq 3\} - \Pr\{Y \leq 0\} = \text{pbinom}(3,5,0.1) - \text{pbinom}(0,5,0.1) \doteq 0.40905 \end{aligned}$$

Normal Distribution Normal distributions have symmetric, bell-shaped density curves that are described by two parameters: the mean μ and the standard deviation σ. The two points of a normal density curve that are the steepest—at the "shoulders" of the curve— are precisely one standard deviation above and below the mean.

Heights of individual corn plants may be modeled as normally distributed with a mean of 145 cm and a standard deviation of 22 cm (Samuels and Witmer, page 148, exercise 4.24). Here are several example normal calculations using R.

Find the proportion of plants:
... larger than 100cm;

```
R> 1 - pnorm(100, 145, 22)
```

```
[1] 0.979595
```

... between 120cm and 150cm:

```
R> pnorm(150, 145, 22) - pnorm(120, 145, 22)
```

```
[1] 0.461992
```

... 150cm or less:

```
R> pnorm(150, 145, 22)
```

```
[1] 0.5898942
```

Find the 75th percentile.

```
R> qnorm(0.75, 145, 22)
```

```
[1] 159.8388
```

Find the 99th percentile.

```
R> qnorm(0.99, 145, 22)
```

```
[1] 196.1797
```

Find the endpoints of middle 95% of the distribution.

```
R> qnorm(c(0.025, 0.975), 145, 22)
```

```
[1] 101.8808 188.1192
```