

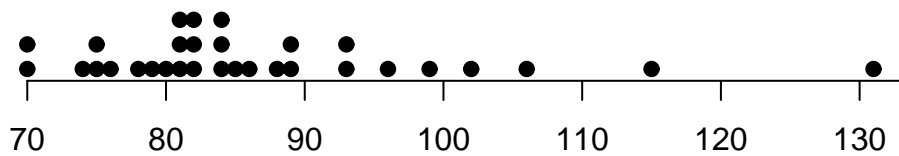
This document describes how to use a number of R commands for plotting one variable and for calculating one variable summary statistics. Specifically, it describes how to use R to create *dotplots*, *histograms*, *stemplots*, and *boxplots*, and to compute the *mean*, *median*, first and third *quartiles*, *standard deviation*, and *variance* of a variable.

Let's begin with the same `glucose` data set from the previous document (Exercise 2.10 from the textbook).

```
> 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)
```

Dotplots. R does not have a built-in dotplot function. A function in R code is available on the R Help web page, in a file named `dotplot.r`. Save this file to your computer and put it in your working directory. You can source this code using the File menu, selecting **Source R code...**, and following the directions. Alternatively, *if the file `dotplot.r` is in your working directory* this command will source the code and then create the dotplot.

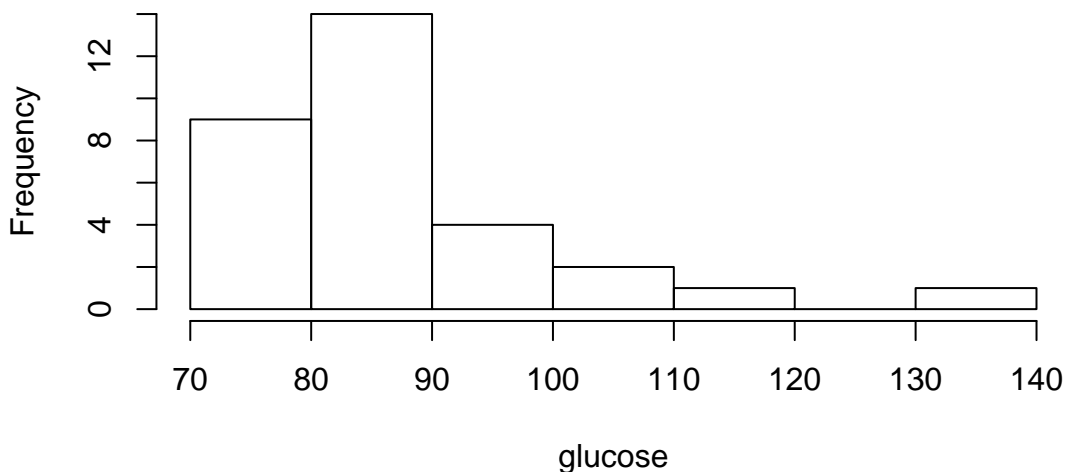
```
> source("dotplot.R")  
> dotplot(glucose)
```



Histograms. The `hist` function produces histograms.

```
> hist(glucose)
```

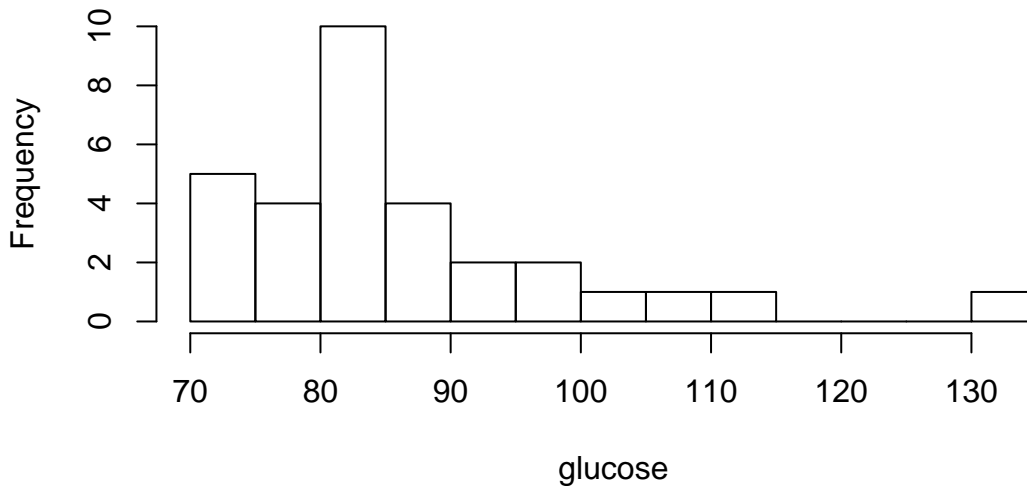
Histogram of glucose



By default, for this data set, there are seven classes, each of width ten, ranging from 70 to 140. If we want to specify a different number of classes, say 14, we can type this.

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

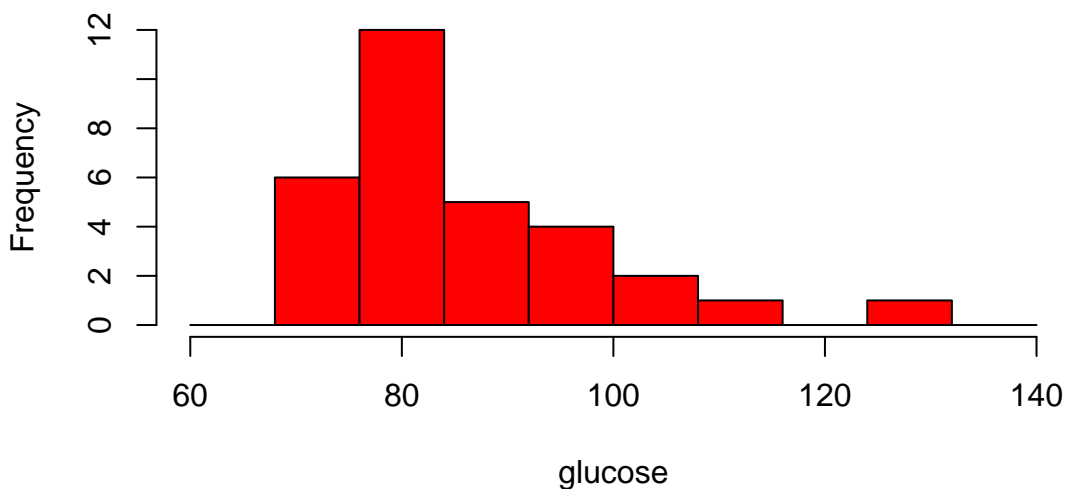
Histogram of glucose



If we wish 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 can do this.

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

Histogram of glucose

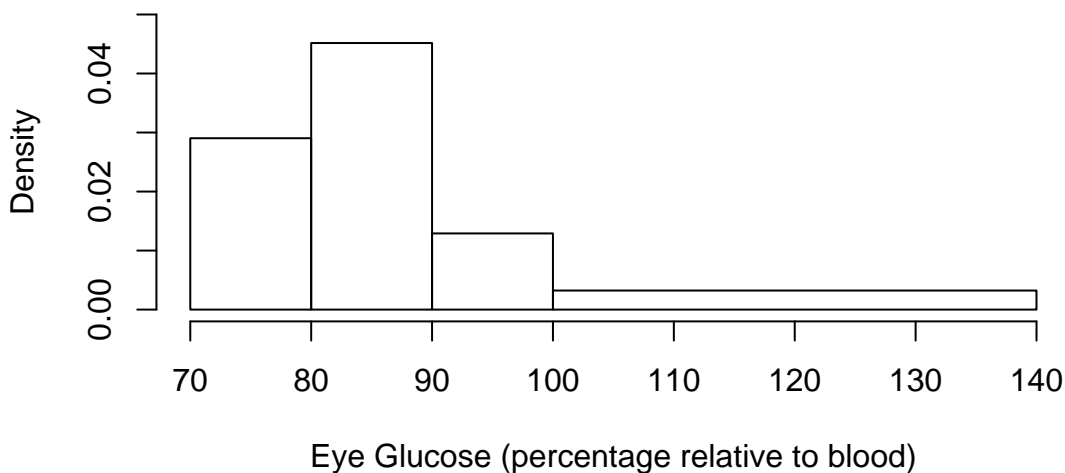


The values of `main`, `xlab`, and `ylab` may be set to change the main title or the labels on the x and y axes. We may also set unequal class widths. R will correctly scale the heights so the areas are proportional

and relative frequencies are equal to density times class width. You may set the range of the axes with `xlim` and `ylim`.

```
> hist(glucose, breaks = c(70, 80, 90, 100, 140),
+      xlab = "Eye Glucose (percentage relative to blood)",
+      main = "Samuels and Witmer, Exercise 2.10", ylim = c(0, 0.05))
```

Samuels and Witmer, Exercise 2.10



Typing `?hist` opens up a help window with more options to modify histograms. The command `?par` will show you how to modify general plotting parameters (Warning: changing graphing parameters is an advanced topic.)

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.

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

The stem-and-leaf display we would be waiting for is the following:

1	0689
2	244569
3	1234
4	688
5	27
6	
7	6

Now look at what R does with this data set, by default:

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

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

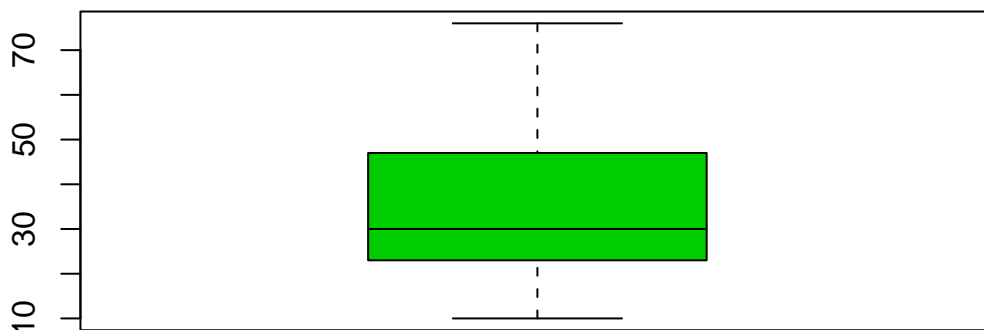
Our previous stems '2' and '3' have been merged together into a new stem '2', containing all leaves from both previous stems: 244569 and then 1234 next to it. In the same way, stems '4' and '5' have been merged together, as well as stems '6' and '7'. Former stem '0' was empty and merged with former stem '1'. The default stem-and-leaf diagram R produced is not retaining all information and thus misleading. For example, it appears that the maximum value is 66, when, in fact, it is 76. Similarly, it is unclear as to whether the '0' stem shows 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.

```
> 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.

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

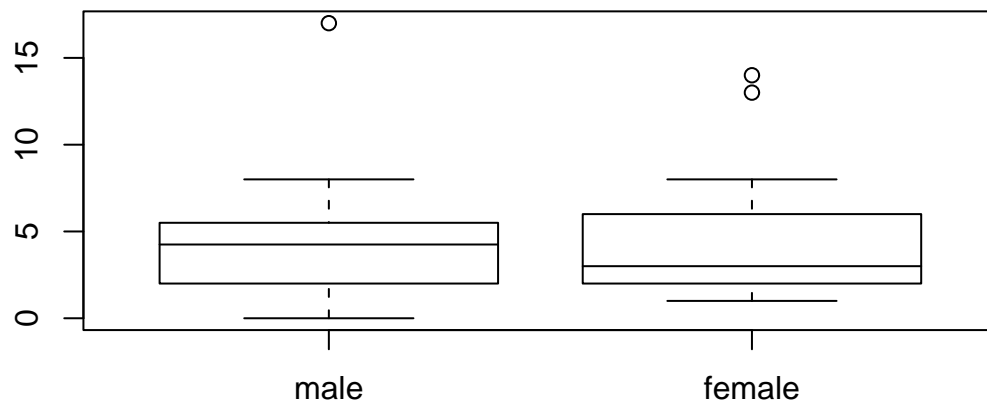


If you like horizontal boxplots better, just tell R. I let you try this.

```
> boxplot(preen, horizontal=TRUE)
```

Exercise 2.33 from the textbook 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.

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



Boxplots show less information than histograms. Their true utility is for making comparisons between different distributions. Here is an example from the text, page 20, data on radish growth.

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

The functions `read.table` and `split` are useful to create parallel boxplots when the data were read in from a file. The function `data.frame` could also be used to create a *data frame* 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, create a data frame `x` from what was read, and verifying `x`'s structure.

```
> x = read.table("ex2-33.txt", header = TRUE)
> str(x)
'data.frame': 25 obs. of 2 variables:
 $ hours: num 6 0 2 1 2 4.5 8 3 17 4.5 ...
 $ sex : Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 ...
> attach(x)
```

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

```
> h = c(maleHours, femaleHours)
> s = c(rep("male", 12), rep("female", 13))
> x = data.frame(hours = h, sex = s)
```

Finally, make the parallel boxplots, with either one of the two commands below.

```
> boxplot(split(hours, sex))
> boxplot(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.

```
> mean(preen)
[1] 33.5
> sd(preen)
[1] 16.31435
> var(preen)
[1] 266.1579
> median(preen)
[1] 30
> max(preen)
[1] 76
> min(preen)
[1] 10
> fivenum(preen)
[1] 10 23 30 47 76
> summary(preen)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 10.0   23.5   30.0   33.5   46.5   76.0
```

We experience here that there is not one single definition for the first and third quartiles ...

```
> IQR(preen)
[1] 23
```

Here is a sample calculation to find the fences for a modified boxplot.

```
> fnum = fivenum(preen)
> iqr = IQR(preen)
> fnum[2] - 1.5 * iqr
[1] -11.5
> fnum[4] + 1.5 * iqr
[1] 81.5
```

Counting observations close to the mean. This code counts the number of observations within one, two, and three standard deviations of the mean for the preening-time data and then reports these as percentages. The function `abs` finds the absolute value.

```
> m = mean(preen)
> s = sd(preen)
> n = length(preen)
> abs(preen - m) < s
[1] TRUE TRUE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE
[13] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

A statement with '`<`' returns true or false for each position of an array. This previous command asks to each fruit flies whether its observation is within one standard deviation from the mean. When a numerical operation is needed, `TRUE` is interpreted as 1 and `FALSE` as 0.

```
> (abs(preen - m) < s) + 0
[1] 1 1 0 0 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1
> sum(abs(preen - m) < s)
[1] 15
```

A sum of `TRUE`s and `FALSE`s is then the number of `TRUE`s. We get here the number of fruit flies within one standard deviation from the mean. To get their percentage, we do the following.

```
> round(sum(abs(preen - m) < s)/n * 100)
[1] 75
```

Now we find the number of fruit flies within two standard deviation from the mean, and get their percentage.

```
> sum(abs(preen - m) < 2 * s)
[1] 19
> round(sum(abs(preen - m) < 2 * s)/n * 100)
[1] 95
> sum(abs(preen - m) < 3 * s)
[1] 20
> round(sum(abs(preen - m) < 3 * s)/n * 100)
[1] 100
```