Introduction to the Practice of Statistics using R: Chapter 1

Nicholas J. Horton[*] Ben Baumer

March 10, 2013

Contents

Introduction

This document is intended to help describe how to undertake analyses introduced as examples in the Sixth Edition of *Introduction to the Practice of Statistics* (2009) by David Moore, George McCabe and Bruce Craig. More information about the book can be found at `http://bcs.whfreeman.com/ips6e/`. This file as well as the associated knitr reproducible analysis source file can be found at `http://www.math.smith.edu/~nhorton/ips6e`.

[*]Department of Mathematics and Statistics, Smith College, nhorton@smith.edu

This work leverages initiatives undertaken by Project MOSAIC (http://www.mosaic-web.org), an NSF-funded effort to improve the teaching of statistics, calculus, science and computing in the undergraduate curriculum. In particular, we utilize the mosaic package, which was written to simplify the use of R for introductory statistics courses. A short summary of the R needed to teach introductory statistics can be found in the mosaic package vignette (http://cran.r-project.org/web/packages/mosaic/vignettes/MinimalR.pdf).

To use a package within R, it must be installed (one time), and loaded (each session). The package can be installed using the following command:

```
> install.packages('mosaic')                # note the quotation marks
```

The # character is a comment in R, and all text after that on the current line is ignored.

Once the package is installed (one time only), it can be loaded by running the command:

```
> require(mosaic)
```

This needs to be done once per session.

We also set some options to improve legibility of graphs and output.

```
> trellis.par.set(theme=col.mosaic())  # get a better color scheme for lattice
> options(digits=3)
```

The specific goal of this document is to demonstrate how to replicate the analysis described in Chapter 1: Looking at Data (Distributions).

1 Displaying distributions with graphs

1.1 Histograms

Table 1.1 (page 8) displays service times (in seconds) for calls to a customer service center.

We begin by reading the data and summarizing the variable.

```
> calltimes = read.csv("http://www.math.smith.edu/ips6eR/ch01/eg01_004.csv")
> summary(calltimes)

     length
 Min.   :    1
 1st Qu.:   57
 Median :  115
 Mean   :  189
 3rd Qu.:  225
 Max.   :28739

> head(calltimes)
```

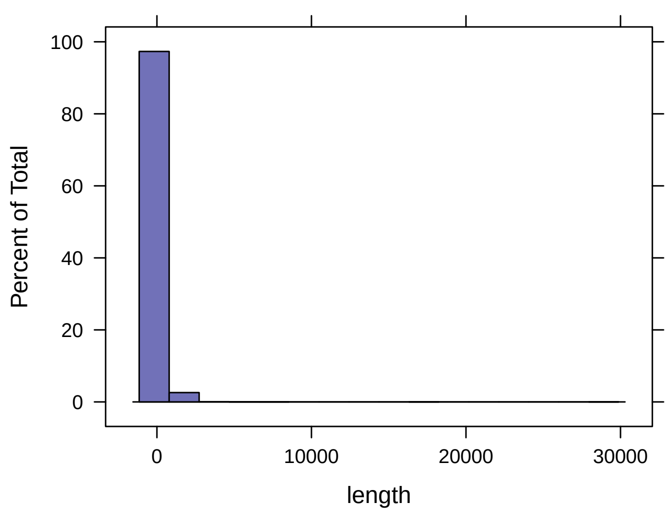
```
  length
1     77
2    289
3    128
4     59
5     19
6    148

> nrow(calltimes)

[1] 31492

> favstats(~ length, data=calltimes)

 min Q1 median  Q3   max mean  sd     n missing
   1 57    115 225 28739  189 313 31492       0
```

The `=` sign is one of the assignment operators in R (the other common one is `<-`). We use this to create a dataframe read from the internet using the `read.csv()` function to read a Comma-Separated Value file.

A total of 31492 service times are reported in the dataframe (or dataset) called `calltimes`. The `head()` function displays the first rows of the dataframe, which has a single variable called `length` (length of the service times, in seconds).

Creating a histogram using the defaults is straightforward, and requires specification of the variable and the dataset:

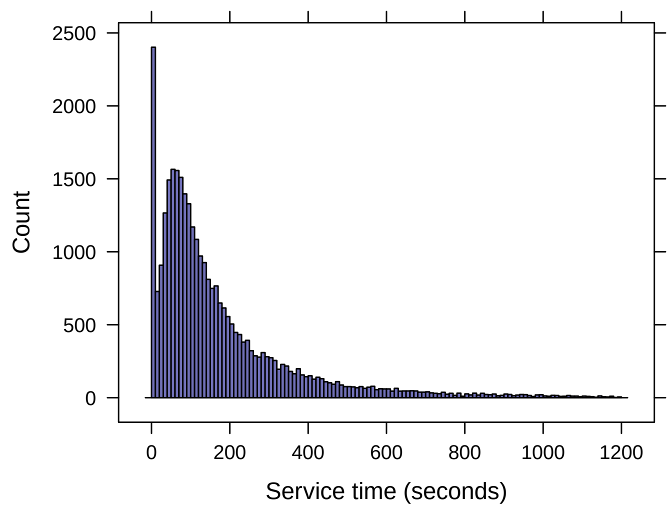
```
> histogram(~ length, data=calltimes)
```



To match the output in Figure 1.4 (page 8), we can add some additional options that display counts rather than density, add more bins, restrict the x-axis limits, and improve the axis labels.

We begin by creating a new dataframe called `shortercalls` which matches the condition within the `subset()` function.

```
> shortercalls = subset(calltimes, length <= 1200)
> histogram(~ length, type="count", breaks=121,
   xlab="Service time (seconds)", shortercalls)
```



We can calculate the proportion less than or equal to ten seconds (to replicate the text in Figure 1.4, on page 8).

```
> tally(~ length <= 10, format="percent", data=calltimes)


  TRUE  FALSE  Total
  7.63  92.37 100.00
```

1.2 Stem (and leaf) plots

Figure 1.6 (page 12) displays the stem and leaf plot in Minitab for a sample of n=80 observations from the call lengths dataset.

```
> eightytimes = read.csv("http://www.math.smith.edu/ips6eR/ch01/ta01_001.csv")
> favstats(~ length, data=eightytimes)

 min   Q1 median  Q3  max mean  sd  n missing
   1 54.8    104 200 2631  197 342 80       0

> with(eightytimes, stem(length))
```

```
  The decimal point is 3 digit(s) to the right of the |

  0 | 00000000000000000111111111111111111111111111111111111112222222223333
  0 | 5577
  1 | 01
  1 |
  2 |
  2 | 6
```

Many common functions (e.g. `mean()`, `median()`, `favstats()`) support a `data=` option to specify the dataframe on which to operate. For functions (such as `stem()` which do not, the `with()` function can achieve the same result, and avoids the use of the $ operator to reference a variable within a dataframe.

We can approximate the Minitab output by adding the `scale=` option:

```
> with(eightytimes, stem(length, scale=2))


  The decimal point is 2 digit(s) to the right of the |

   0 | 00000111122334445555666667777888888999000122223344444556888
   2 | 0001378993779
   4 | 478
   6 | 00
   8 | 5
  10 | 5
  12 |
  14 |
  16 |
  18 |
  20 |
  22 |
  24 |
  26 | 3
```

As always, it is critical to include a legend along with a stem and leaf plot. For the latter figure, this would be of the form:

```
Legend:  26 | 3  corresponds to a call of 2,630 seconds.
```

1.3 Creating classes from quantitative variables

```
> iqscores = read.csv("http://www.math.smith.edu/ips6eR/ch01/ta01_003.csv")
> head(iqscores)
```

```
   iq
1 145
2 139
3 126
4 122
5 125
6 130

> names(iqscores)

[1] "iq"

> favstats(~ iq, data=iqscores)

 min  Q1 median  Q3 max mean   sd  n missing
  81 104    114 125 145  115 14.8 60       0
```

We can create classes using the rules defined on page 13 using the `cut()` command:

```
> iqscores = transform(iqscores, iqcat=cut(iq, right=FALSE,
   breaks=c(75, 85, 95, 105, 115, 125, 135, 145, 155)))
> tally(~ iqcat, data=iqscores)


  [75,85)   [85,95)  [95,105) [105,115) [115,125) [125,135) [135,145)
        2         3        10        16        13        10         5
[145,155)     Total
        1        60
```

Here we demonstrate use of the `c()` function to glue together a vector (a one-dimensional array) with the breakpoints).

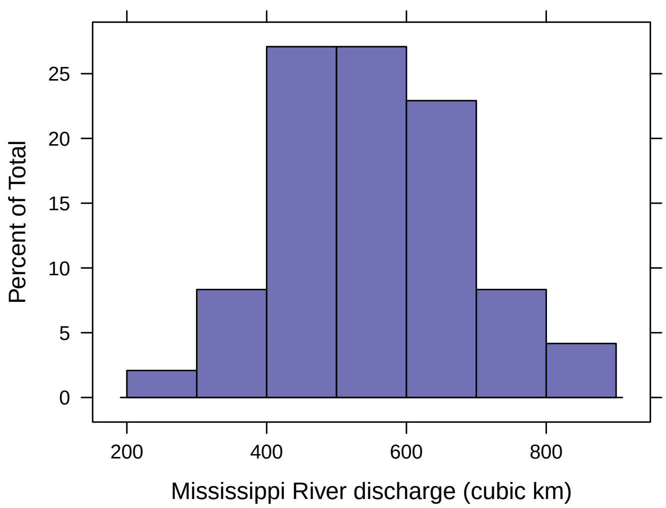
1.4 Time plots

```
> mississippi = read.csv("http://www.math.smith.edu/ips6eR/ch01/ta01_004.csv")
> head(mississippi)

  year discharge
1 1954       290
2 1955       420
3 1956       390
4 1957       610
5 1958       550
6 1959       440

> summary(mississippi)
```

```
      year        discharge
 Min.   :1954   Min.   :290
 1st Qu.:1966   1st Qu.:448
 Median :1978   Median :560
 Mean   :1978   Mean   :563
 3rd Qu.:1989   3rd Qu.:670
 Max.   :2001   Max.   :900
```

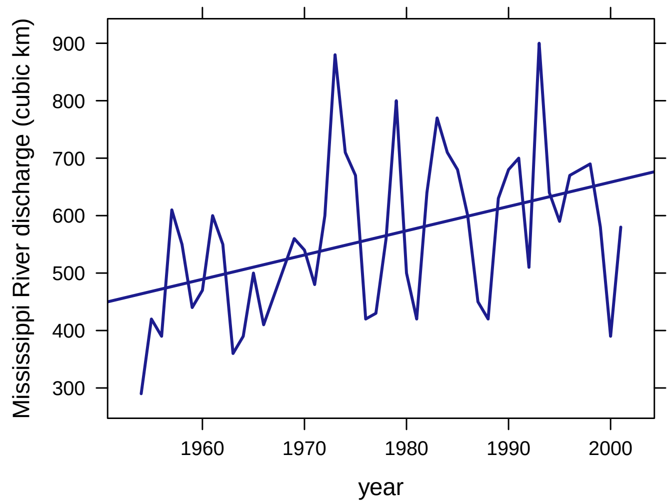
We can replicate Figure 1.10 (a) on page 19 using the **histogram()** command with specification of the breaks.

```
> histogram(~ discharge, breaks=seq(200, 900, by=100),
   xlab="Mississippi River discharge (cubic km)", data=mississippi)
```



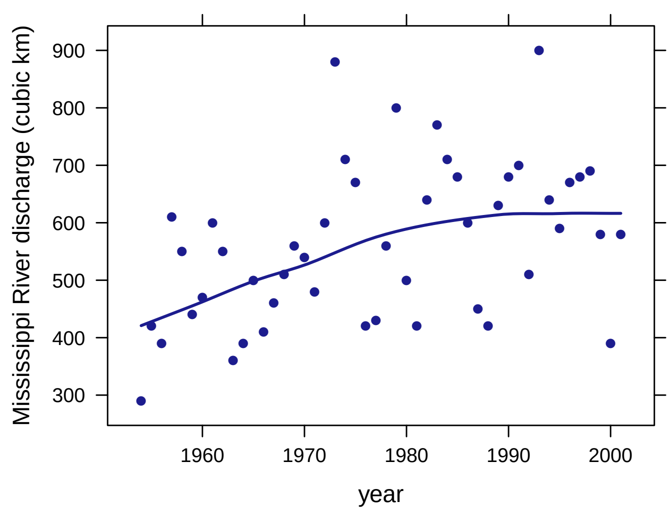
We can replicated Figure 1.10 (b) on page 19 using the **xyplot()** command with specification of the Line and Regression type.

```
> xyplot(discharge ~ year, type=c("l", "r"),
   ylab="Mississippi River discharge (cubic km)", data=mississippi)
```



Other options for `type=` include Points and Smooth:

```
> xyplot(discharge ~ year, type=c("p", "smooth"),
   ylab="Mississippi River discharge (cubic km)", data=mississippi)
```



2 Displaying distributions with numbers

2.1 Mean

We begin by reading in the dataset, and calculating the mean highway mileage of the two seaters:

```
> origmileage = read.csv("http://www.math.smith.edu/ips6eR/ch01/ta01_010.csv",
   stringsAsFactor=FALSE)
> mean(~ Hwy, data=subset(origmileage, Type=="T"))

[1] 24.7

> favstats(~ Hwy, data=subset(origmileage, Type=="T"))

 min Q1 median Q3 max mean   sd  n missing
  13 19     23 28  66 24.7 10.8 21       0
```

The use of `stringsAsFactors` ensures that the `Type` variable can be referenced as a character string.

As described on page 30, we drop the outlier as the authors suggest, with the justification that it appears to be completely different from the other cars.

```
> mileage = subset(origmileage, Hwy < 60)
> twoseat = subset(mileage, Type=="T")
> mean(~ Hwy, data=twoseat)

[1] 22.6

> favstats(~ Hwy, data=twoseat)

 min   Q1 median   Q3 max mean   sd  n missing
  13 18.5     23 26.5  32 22.6 5.29 20       0
```

The dataset with the outlier dropped will be used for all further analyses.

2.2 Median and quantiles

The `favstats()` function displays a variety of useful quantities, though other functions are also available to calculate specific statistics.

```
> favstats(~ Hwy, data=twoseat)

 min   Q1 median   Q3 max mean   sd  n missing
  13 18.5     23 26.5  32 22.6 5.29 20       0

> median(~ Hwy, data=twoseat)

[1] 23

> with(twoseat, quantile(Hwy, probs=c(0.5)))

50%
 23
```

This is an example of the use of `with()` to make a variable within a dataframe accessible to the `quantile()` function.

The output matches the description in Example 1.16 (page 35).

The default behavior in R for the calculation of quantiles does not match that of SPSS and Minitab. For those with a fetish for accuracy, the results displayed in part (b) of Figure 1.18 (page 36) can be replicated using the `type=6` option to `quantile()`:

```
> with(twoseat, quantile(Hwy, probs=c(0.25, 0.75), type=6))

 25%  75%
17.5 27.5
```

2.3 Five number summary

```
> favstats(~ Hwy, data=twoseat)

 min   Q1 median   Q3 max mean   sd  n missing
  13 18.5     23 26.5  32 22.6 5.29 20       0

> min(~ Hwy, data=twoseat)

[1] 13

> max(~ Hwy, data=twoseat)

[1] 32

> with(twoseat, fivenum(Hwy))

[1] 13 18 23 27 32
```

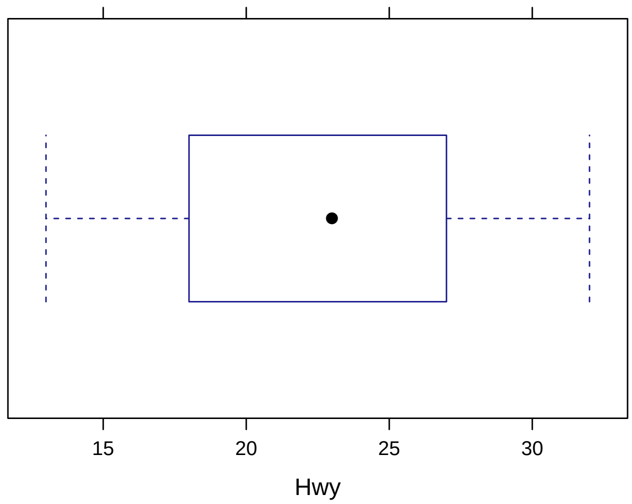
Note that the five number summary is calculating the lower and upper hinges, rather than Q1 and Q3.

For pedagogical purposes, we often find it simpler to just introduce `favstats()` for calculations of this sort.

2.4 Interquartile range and outliers

We can calculate the IQR, as well as display boxplots.

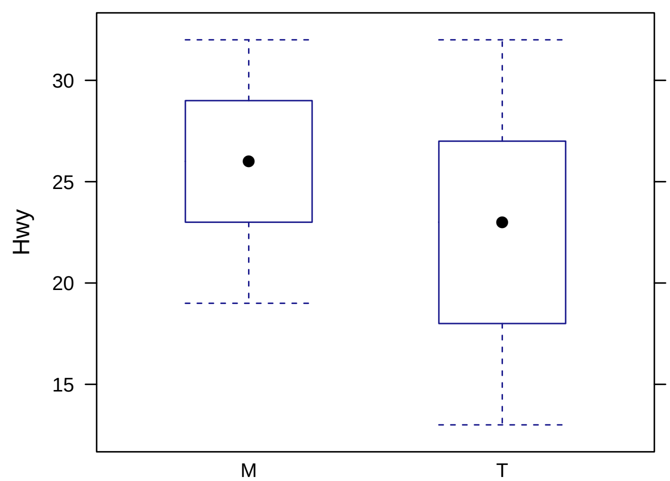
```
> with(twoseat, IQR(Hwy))

[1] 8

> bwplot(~ Hwy, data=twoseat)
```



This matches the display for two seater cars on page 37.

We generally encourage students to use boxplots when comparing two of more groups, as it's not a particularly compelling display for a single population.

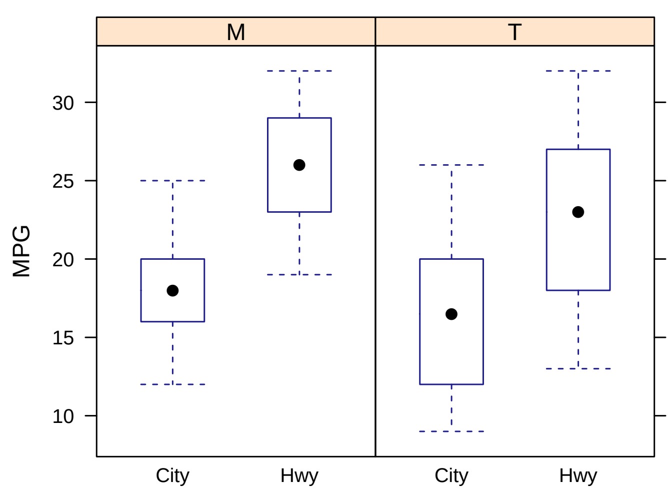
```
> bwplot(Hwy ~ Type, data=mileage)
```



To generate all four groups from Figure 1.19 (page 37), we need to transform the dataset into *tall* format. This is a somewhat pesky data management task that is best done by instructors (rather than students) early on in a course.

```
> head(mileage)

  Type City Hwy
1    T   17  24
```

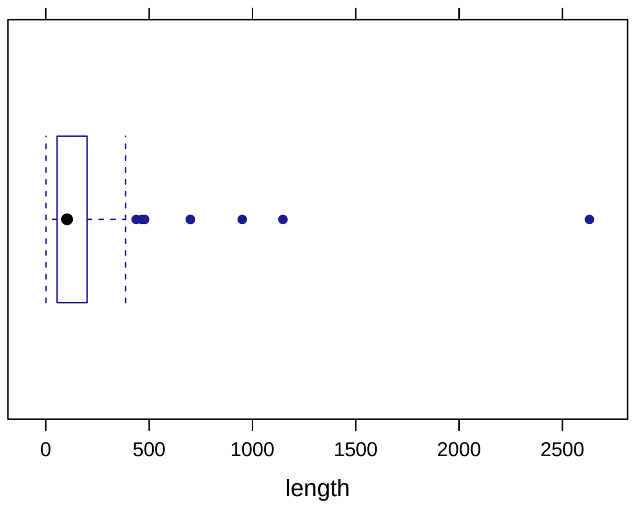
```
2    T   20  28
3    T   20  28
4    T   17  25
5    T   18  25
6    T   12  20

> # create a vector of locations
> Location = c(rep("Hwy", nrow(mileage)), rep("City", nrow(mileage)))
> # create a vector of car types
> CarType = with(mileage, c(Type, Type))
> # create a vector of miles per gallon
> MPG = with(mileage, c(Hwy, City))
> # glue them all together
> figure1.19 = with(mileage, data.frame(CarType, MPG, Location))
> head(figure1.19)

  CarType MPG Location
1       T  24      Hwy
2       T  28      Hwy
3       T  28      Hwy
4       T  25      Hwy
5       T  25      Hwy
6       T  20      Hwy

> # cleanup
> rm(Location, CarType, MPG)
> bwplot(MPG ~ Location | CarType, data=figure1.19)
```



2.5 IQR rule and outliers

We can flag outliers using the 1.5 IQR rule, for the call times dataset (as displayed in Figure 1.20):

```
> bwplot(~ length, data=eightytimes)
```



We can also display information regarding the outliers:

```
> threshold = 1.5 * with(eightytimes, IQR(length))
> threshold

[1] 217

> q1 = with(eightytimes, quantile(length, probs=0.25))
> q1

 25%
54.8

> q3 = with(eightytimes, quantile(length, probs=0.75))
> q3

75%
200

> # outlier if either condition matches
> eightytimes = transform(eightytimes,
   outliers = (length < q1 - threshold) | (length > q3 + threshold))
```

```
> tally(~ outliers, data=eightytimes)


 TRUE FALSE Total
    8    72    80

> favstats(~ length, data=subset(eightytimes, outliers==TRUE))

 min  Q1 median   Q3  max mean  sd n missing
 438 476    700 1000 2631  939 728 8       0

> subset(eightytimes, outliers==TRUE)

   length outliers
19    438     TRUE
23    479     TRUE
29   2631     TRUE
36    700     TRUE
54    951     TRUE
65    700     TRUE
77   1148     TRUE
79    465     TRUE
```

2.6 Standard deviation and variance

It's straightforward to calculate the variance and standard deviation directly within R.

```
> x = c(1792, 1666, 1362, 1614, 1460, 1867, 1439)
> n = length(x)
> n

[1] 7

> mean(x)

[1] 1600

> myvar = sum((x - mean(x))^2) / (n - 1)
> myvar

[1] 35812

> sqrt(myvar)

[1] 189
```

But it's simpler to use the built-in commands:

```
> var(x)

[1] 35812

> sd(x)

[1] 189
```

These match the values calculated on page 41.

Normally, we'll access variables in a dataframe, which requires use of the operator and the `data=` statement (or use of `with()`).

2.7 Linear transformations

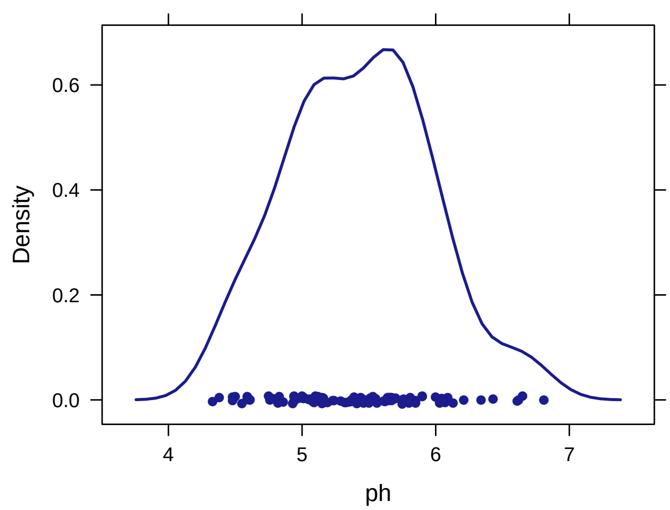
We replicate the analyses from example 1.22 (page 46). Instead of operating directly on the vector, we'll create a simple dataframe.

```
> score = c(1056, 1080, 900, 1164, 1020)
> grades = data.frame(score)
> mean(~ score, data=grades)

[1] 1044

> sd(~ score, data=grades)

[1] 96.4

> grades = transform(grades, points = score / 4)
> grades

  score points
1  1056    264
2  1080    270
3   900    225
4  1164    291
5  1020    255

> mean(~ points, data=grades)

[1] 261

> sd(~ points, data=grades)

[1] 24.1
```

3 Density curves and normal distributions

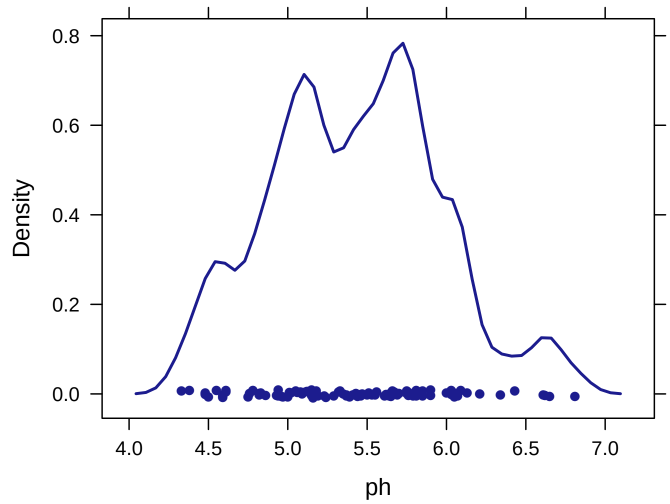
3.1 Density curves

```
> rainwater = read.csv("http://www.math.smith.edu/ips6eR/ch01/ex01_036.csv")
> names(rainwater)

[1] "ph"

> densityplot(~ ph, data=rainwater)
```



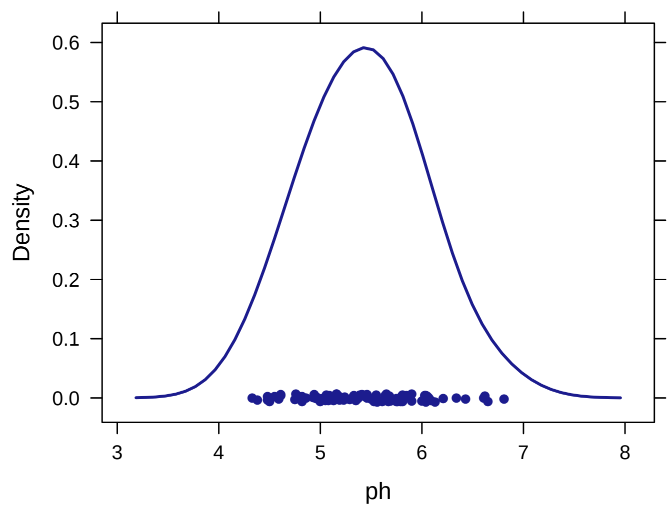
We can adjust how "smooth" the curve will be. Here we make the bandwidth (see page 71) narrower, which will make the curve less smooth.

```
> densityplot(~ ph, adjust=0.5, data=rainwater)
```



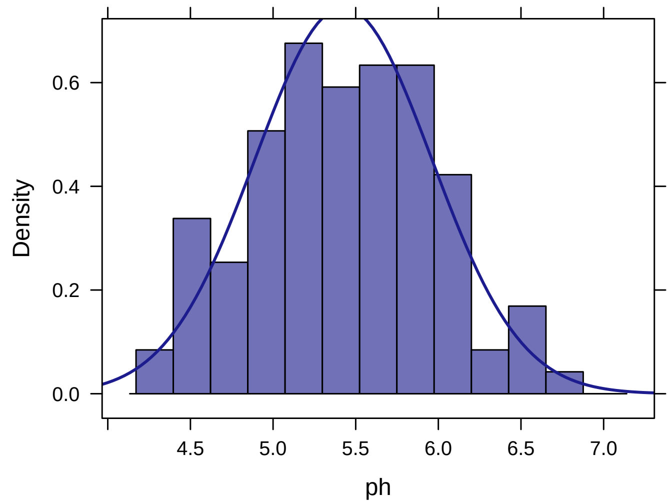
Here we make the bandwidth wider, which will make the curve smoother.

```
> densityplot(~ ph, adjust=2, data=rainwater)
```



The defaults are generally satisfactory.
We can also overlay a normal distribution on top of a histogram.

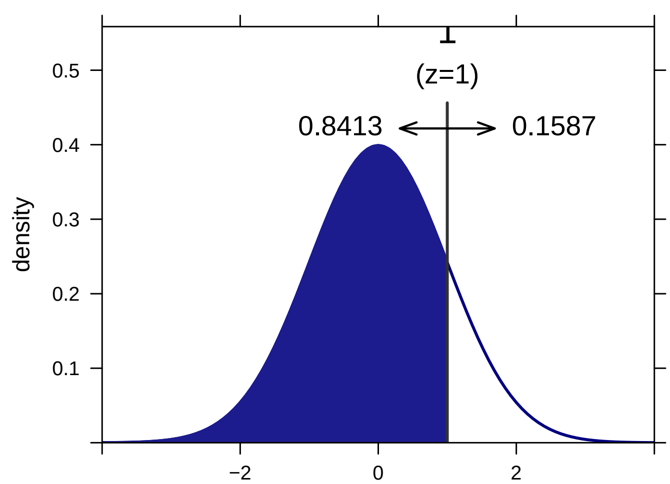
```
> xhistogram(~ ph, fit='normal', data=rainwater)

Loading required package:  MASS
```



3.2 Empirical (68/95/99.7) rule

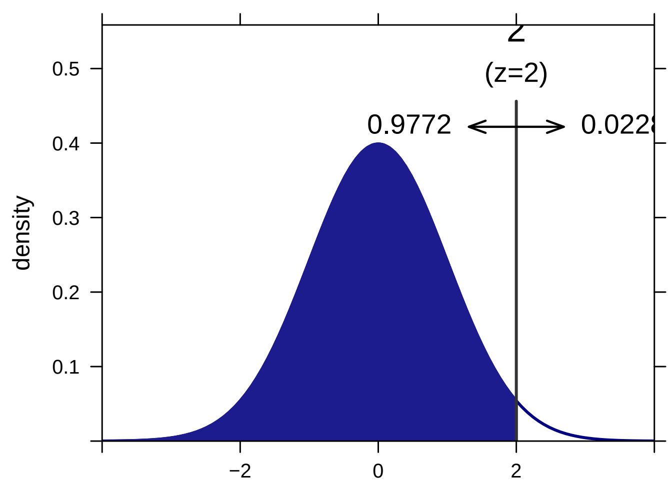
While it's straightforward to use R to calculate the probabilities for any distribution, many times the empirical (or 68/95/99.7) rule can be used to get a rough sense of probabilities.

```
> xpnorm(1, mean=0, sd=1)


If X ~ N(0,1), then

P(X <= 1) = P(Z <= 1) = 0.8413
P(X >  1) = P(Z >  1) = 0.1587
[1] 0.841
```



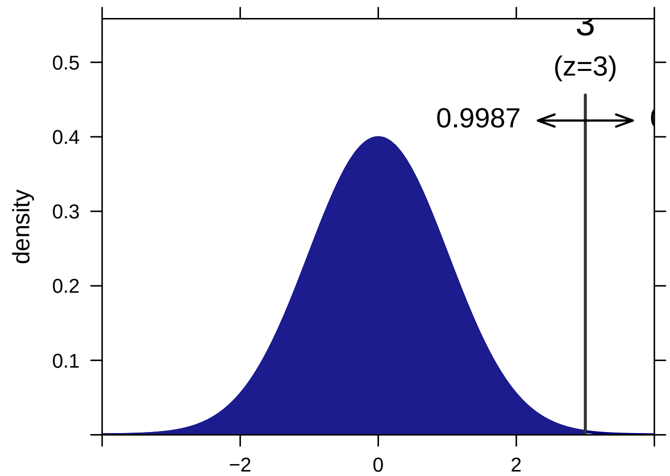
Because it is symmetric, we observe that approximately $2 * .1587 = 0.317$ (or a little less than 1/3) of the density for a normal distribution is more than 1 standard deviation from the mean.

```
> xpnorm(2, mean=0, sd=1)


If X ~ N(0,1), then

P(X <= 2) = P(Z <= 2) = 0.9772
P(X >  2) = P(Z >  2) = 0.0228
[1] 0.977
```



Similarly, we observe that approximately $2 * .0228 = 0.046$ (or a little less than 5%) of the density for a normal distribution is more than 2 standard deviations from the mean.

```
> xpnorm(3, mean=0, sd=1)


If X ~ N(0,1), then

P(X <= 3) = P(Z <= 3) = 0.9987
P(X >  3) = P(Z >  3) = 0.0013
[1] 0.999
```



Only a small proportion ($2 * .0013 = 0.003$) of the density of a normal distribution is more than 3 standard deviations from the mean.

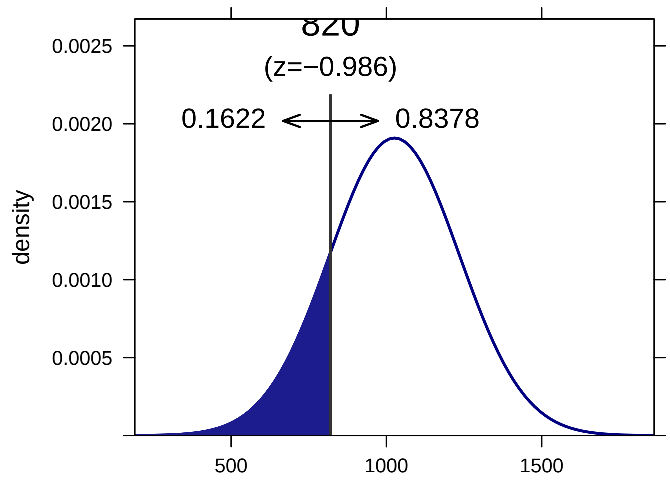
We also know that the probability of a value above the mean is 0.5, since the distribution is symmetric.

3.3 Normal distribution calculations

The `xpnorm()` function can be used to calculate normal probabilities (look ma: no Table!). More formally, it calculates the probability that a random variable X takes on probability of x or less given a distribution with mean μ and standard deviation σ.

Example 1.27 (page 63) calculates the probability that a student had a score of 820 on the SAT, given that SAT scores are approximately normal with mean $\mu = 1026$ and standard deviation $\sigma = 209$:

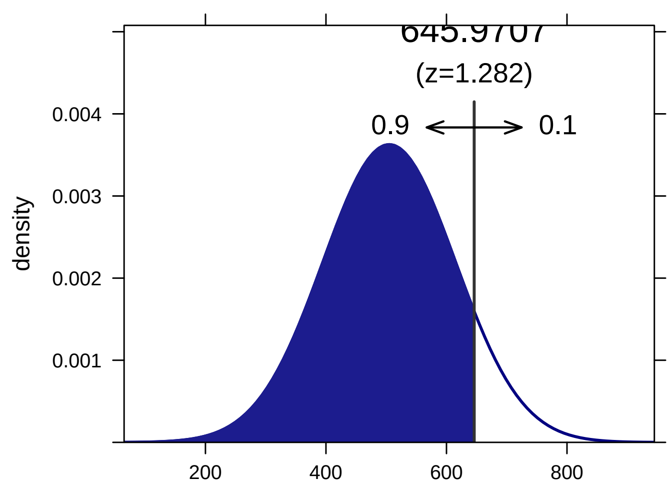
```
> xpnorm(820, mean=1026, sd=209)


If X ~ N(1026,209), then

P(X <= 820) = P(Z <= -0.986) = 0.1622
P(X >  820) = P(Z >  -0.986) = 0.8378
[1] 0.162
```



This matches the value of 0.8379 at the bottom of the page.

Other functions can be used to work backwards to find a quantile in terms of a probability. Example 1.32 (page 67) asks to find the quantile of the distribution which corresponds to the top 10%:

```
> xqnorm(.90, mean=505, sd=110)

P(X <= 645.970672209906) = 0.9
P(X >  645.970672209906) = 0.1
[1] 646
```



The value of 646 matches the value from the calculations from the Table.

3.4 Normal quantile plots

We can replicate Figure 1.34 (normal quantile plot of the breaking strengths of wires, page 69) using the qqnorm() command:

```
> wires = read.csv("http://www.math.smith.edu/ips6eR/ch01/eg01_011.csv")
> names(wires)

[1] "strength"

> with(wires, qqnorm(strength))
```