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)
```

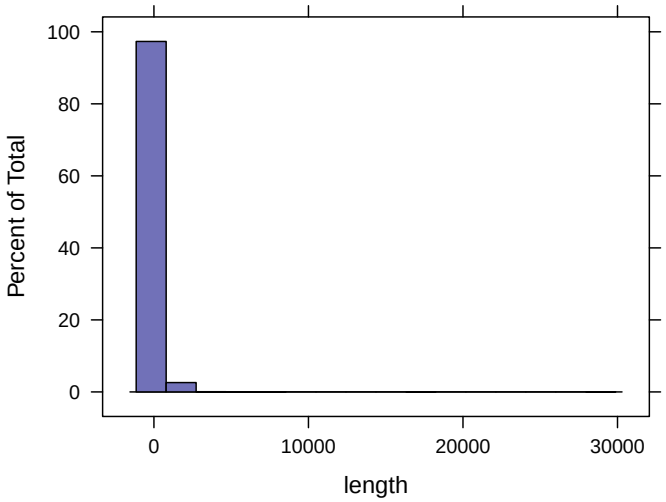
```
  length
1     77
2    289
3    128
4     59
5     19
6    148

> 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:

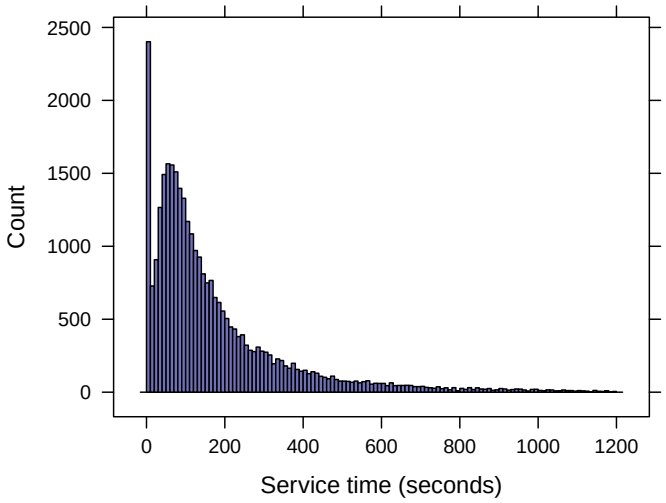
```
> histogram(~ length, data=calltimes)
```



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 | 000000000000000001111111111111111111111111111111111111112222222223333
  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 | 000001111223344455556666677778888889990001222233444445568888
   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).

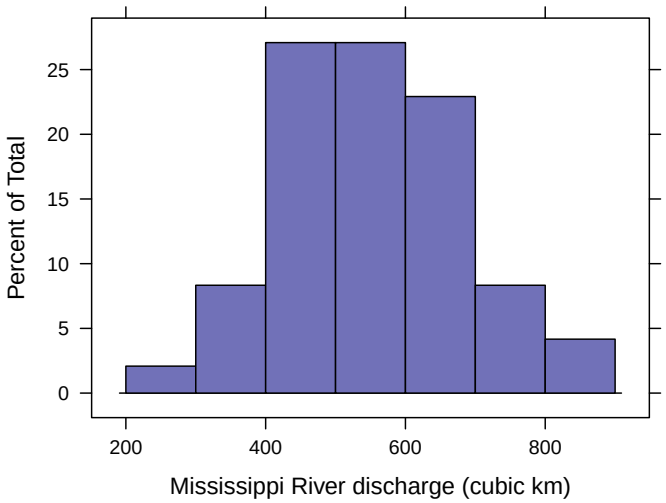
1.4 Time plots

```
> 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
```

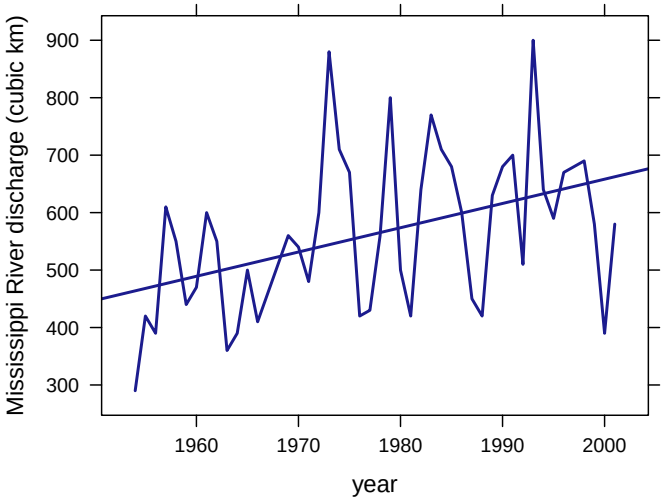
We can replicate Figure 1.10 (a) on page 19 using the `histogram()` command with specification of the breaks.

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



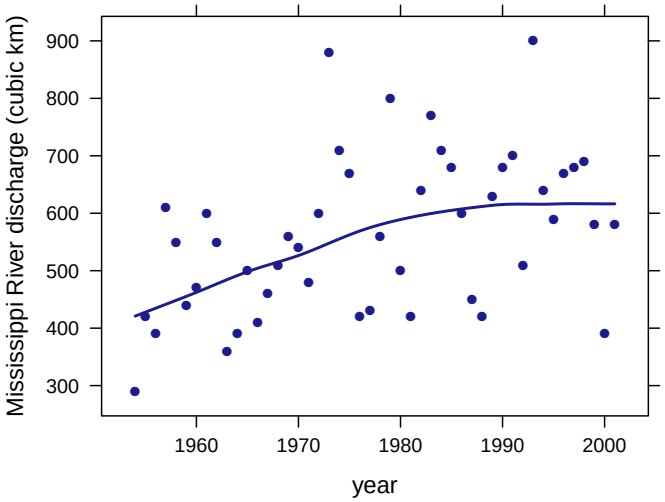
We can replicated Figure 1.10 (b) on page 19 using the `xyplot()` command with specification of the Line and Regression type.

```
> 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
```

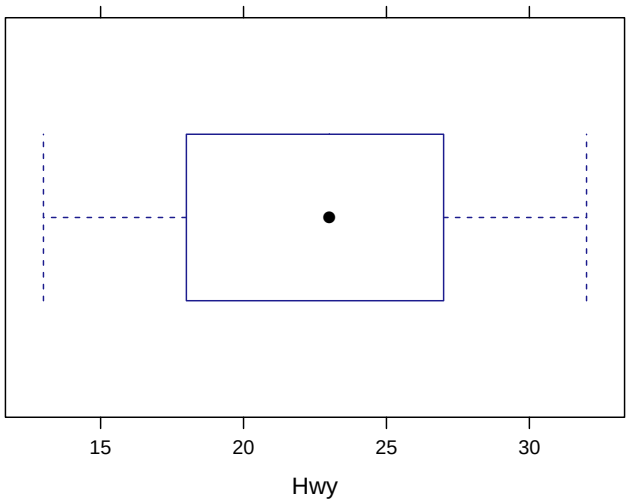
Note that the five number summary is calculating the lower and upper hinges, rather than Q1 and Q3.

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.

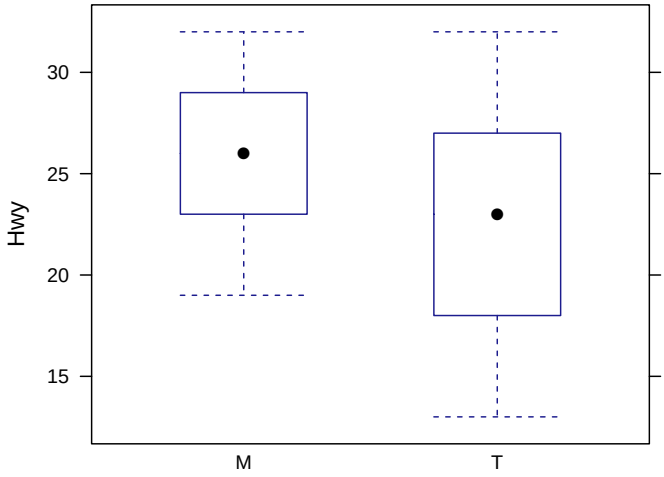
```
> with(twoseat, IQR(Hwy))

[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.

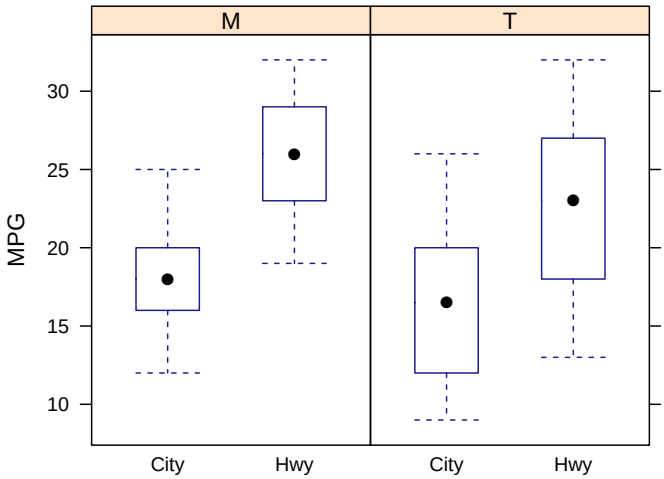
```
> bwplot(Hwy ~ Type, data=mileage)
```



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
```

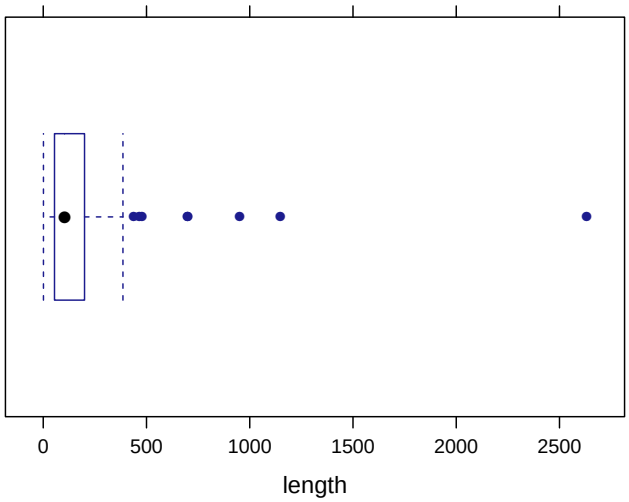
```
2    T   20  28
3    T   20  28
4    T   17  25
5    T   18  25
6    T   12  20

> # 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

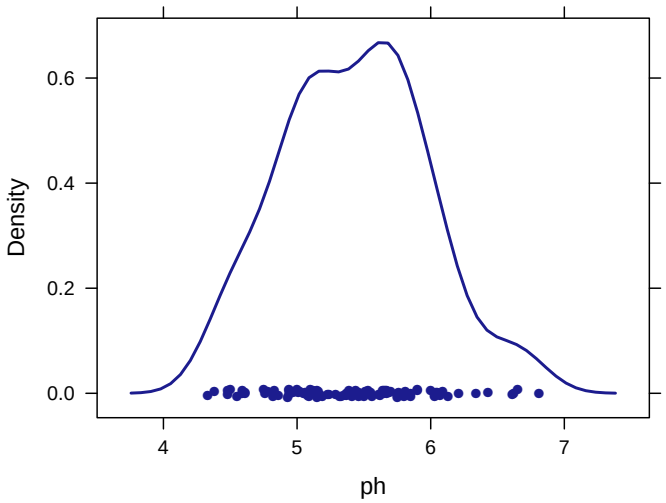
We replicate the analyses from example 1.22 (page 46). Instead of operating directly on the vector, we'll create a simple dataframe.

```
> 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

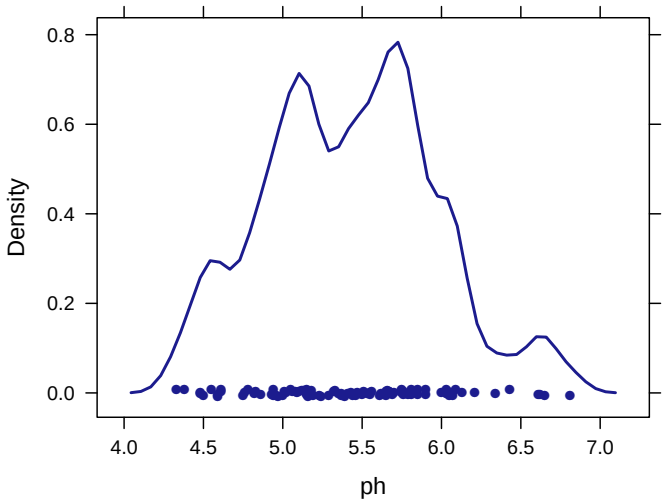
3.1 Density curves

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

[1] "ph"

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



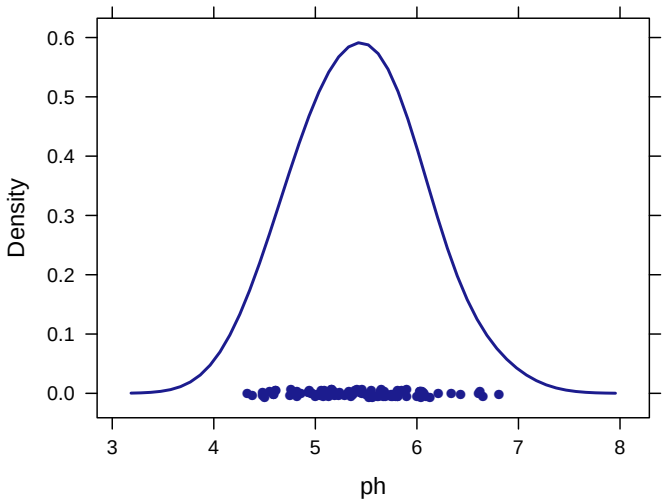
We can adjust how "smooth" the curve will be. Here we make the bandwidth (see page 71) narrower, which will make the curve less smooth.

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



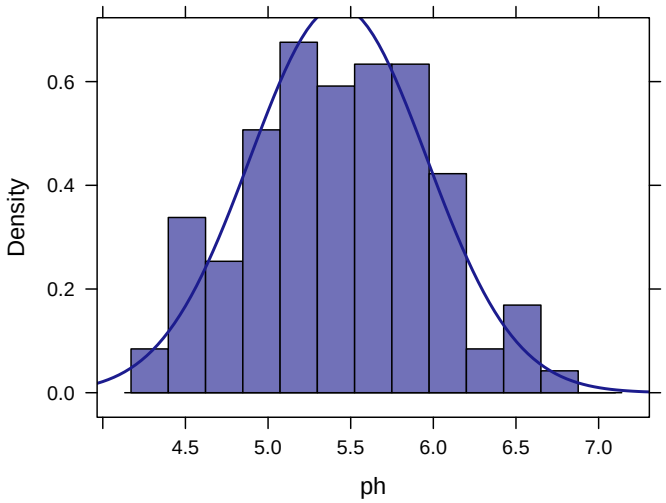
Here we make the bandwidth wider, which will make the curve smoother.

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



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

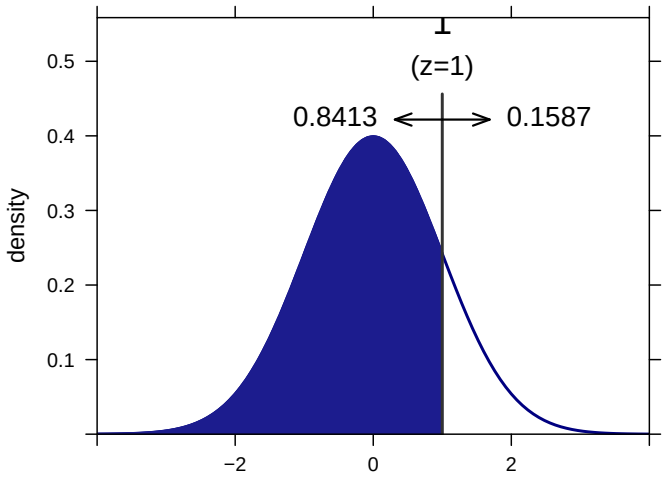
```
> xhistogram(~ ph, fit='normal', data=rainwater)

Loading required package:  MASS
```



3.2 Empirical (68/95/99.7) rule

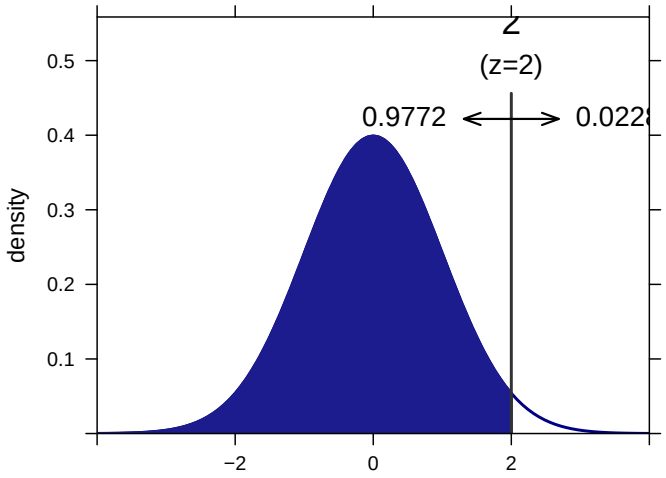
While it's straightforward to use R to calculate the probabilities for any distribution, many times the empirical (or 68/95/99.7) rule can be used to get a rough sense of probabilities.

```
> 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
```



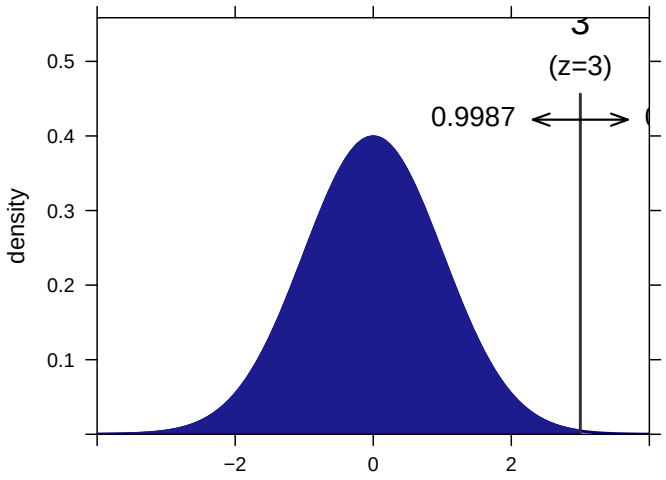
Because it is symmetric, we observe that approximately $2 * .1587 = 0.317$ (or a little less than 1/3) of the density for a normal distribution is more than 1 standard deviation from the mean.

```
> 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.

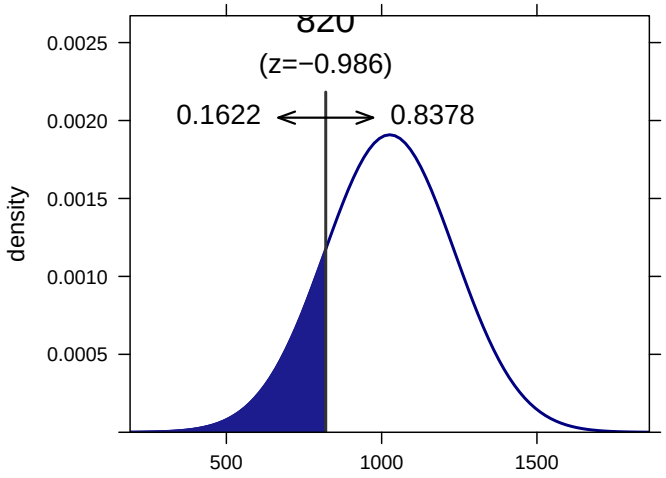
We also know that the probability of a value above the mean is 0.5, since the distribution is symmetric.

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 $\sigma = 209$:

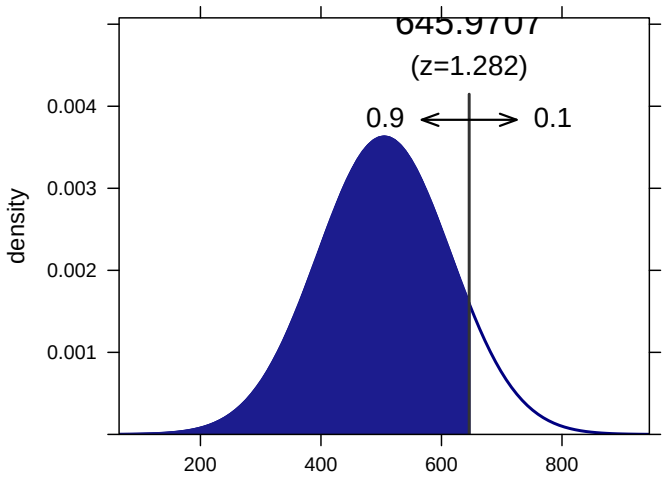
```
> xpnorm(820, mean=1026, sd=209)


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
```

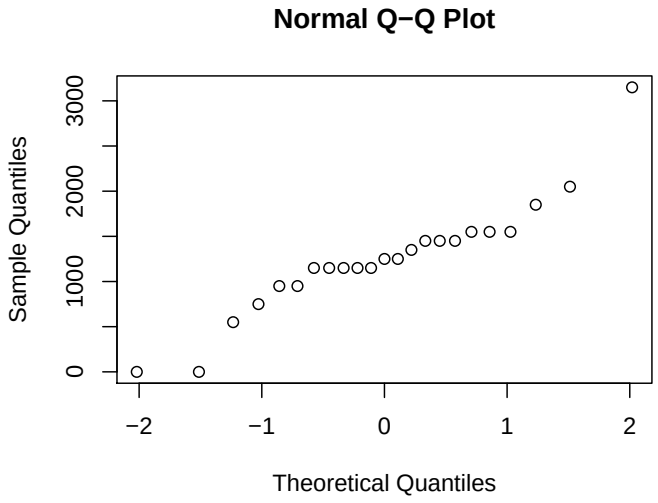


The value of 646 matches the value from the calculations from the Table.

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))
```



Introduction to the Practice of Statistics using R: Chapter 2

Ben Baumer Nicholas J. Horton*

July 31, 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`.

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

*Department of Mathematics, Amherst College, nhorton@amherst.edu

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.

This chapter also references a dataset from the second edition of *The Statistical Sleuth*, so this must be installed as well.

```
> install.packages('Sleuth2')                # note the quotation marks
```

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

```
> require(mosaic)
> require(Sleuth2)
```

This needs to be done once per session.

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

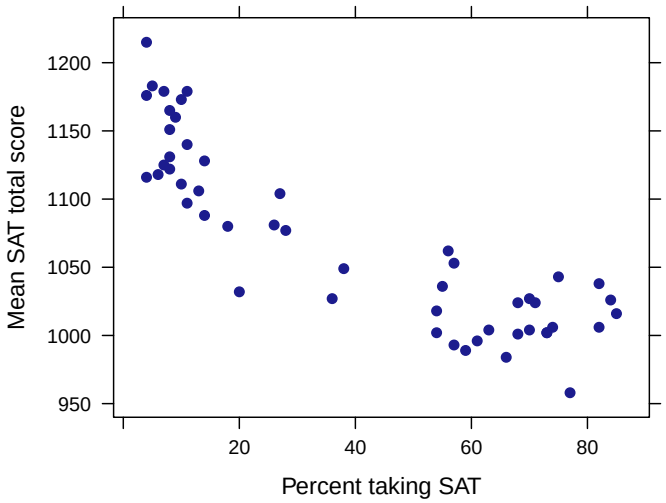
```
> trellis.par.set(theme=col.mosaic())  # get a better color scheme for lattice
> options(digits=3)
```

The specific goal of this document is to demonstrate how to replicate the analysis described in Chapter 2: Looking at Data (Relationships).

1 Scatterplots

Example 2.6 (page 87) shows a scatterplot of the average SAT scores for all 50 states and the District of Columbia.

```
> SAT = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_006.csv")
> head(SAT)

       state percent satv satm total
1    Alabama      10  559  552  1111
2     Alaska      55  518  518  1036
3    Arizona      38  524  525  1049
4   Arkansas       6  564  554  1118
5 California      54  499  519  1018
6   Colorado      27  551  553  1104

> plotPoints(total ~ percent, data=SAT, xlab="Percent taking SAT",
   ylab="Mean SAT total score", pch=19)
```



This can also be generated using the `xyplot()` function.

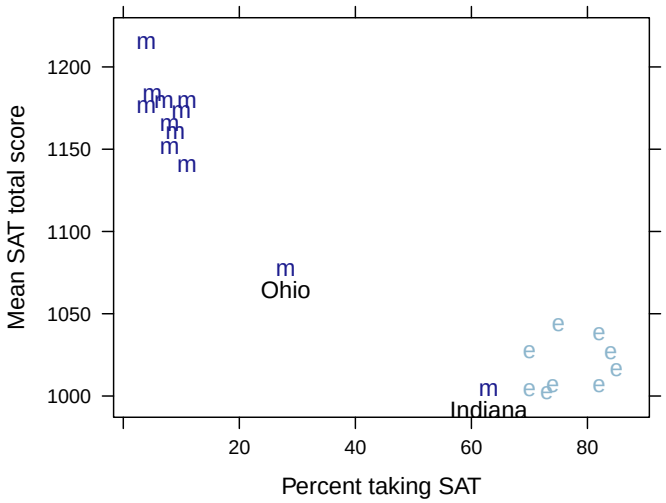
1.1 Adding categorical variables to scatterplots

Figure 2.2 (page 89) illustrates the use of different plotting symbols as a means of distinguishing levels of a categorical variable in a scatterplot. This can be achieved just as easily using color and the `groups` attribute. First, we assign a region to each state. This can be done using the `transform` and `ifelse` commands.

```
> midwest = c("Ohio", "Michigan", "Wisconsin", "Indiana", "Illinois", "Minnesota"
              , "Iowa", "NorthDakota", "SouthDakota", "Nebraska"
              , "Kansas", "Missouri")
> northeast = c("Maine", "NewHampshire", "Vermont", "Massachusetts", "Connecticut"
                , "RhodeIsland", "NewYork", "NewJersey", "Pennsylvania")
> SAT = transform(SAT, region = ifelse(state %in% midwest, "midwest",
   ifelse(state %in% northeast, "northeast", NA)))
```

Now that the `region` variable is set, we can use the `groups` argument to separate the states by color. Adding specific labels can be achieved by using `ladd` and `panel.text` to add things to an existing plot. Note the use of the `subset()` function to restrict the states which are being plotted and labelled.

```
> plotPoints(total ~ percent, groups=region, data=subset(SAT, !is.na(region)),
   xlab="Percent taking SAT", ylab="Mean SAT total score", pch=c("m", "e"),
   cex=1.5)
> SAT.labels = subset(SAT, state %in% c("Ohio", "Indiana"))
> with(SAT.labels, ladd(panel.text(percent, total, state, pos=1)))
```



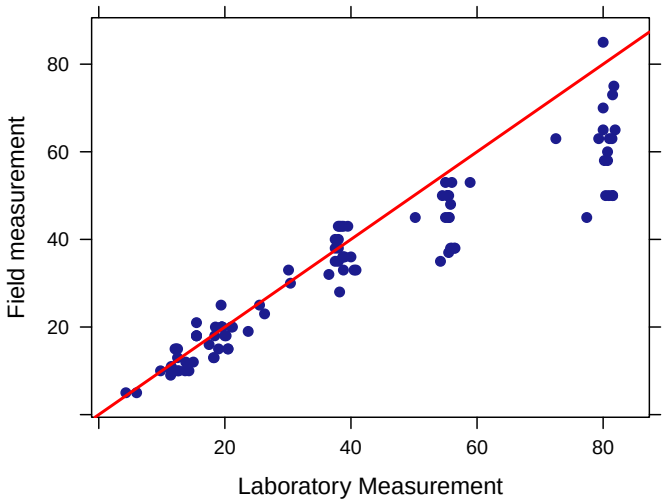
1.2 More examples of scatterplots

Example 2.7 (page 90) concerns the Trans-Alaska Pipeline. The dataset is in the following format:

```
> Oil = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_007.csv")
> head(Oil)

  field  lab
1    18 20.2
2    38 56.0
3    15 12.5
4    20 21.2
5    18 15.5
6    36 39.0
```

We can add a straight line to a plot using the `panel.abline` function. In this we specify the line $y = x$ by giving the first argument (the intercept) as 0 and the second argument (the slope) as 1. Thus, the `abline` is $y = a + bx$.

```
> plotPoints(field ~ lab, data=Oil, xlab="Laboratory Measurement",
   ylab="Field measurement", pch=19)
> ladd(panel.abline(a=0, b=1, col="red"))
```

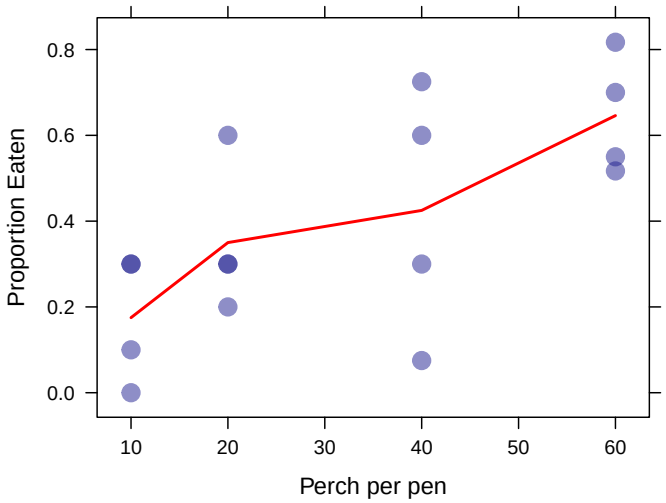


Example 2.8 (page 91) relates the density of perch to the proportion that are killed by predators.

```
> Perch = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_008.csv")
> head(Perch)

  perch killed
1    10    0.0
2    10    0.1
3    10    0.3
4    10    0.3
5    20    0.2
6    20    0.3
```

Figure 2.4 (pagee 92) shows the relationship between perch density and the proportion killed. Note that some data points occur multiple times. Although the textbook uses different plot marks to distinguish single instances from those with multiplicity, we can use transparency to achieve the same effect.

```
> plotPoints(killed ~ perch, data=Perch, xlab="Perch per pen",
   ylab="Proportion Eaten", pch=19, cex=1.5, alpha=0.5)
> perch.means = mean(~killed | perch, data=Perch)
> perch.means

   10    20    40    60
0.175 0.350 0.425 0.646

> ladd(panel.lines(names(perch.means), perch.means, col="red"))
```

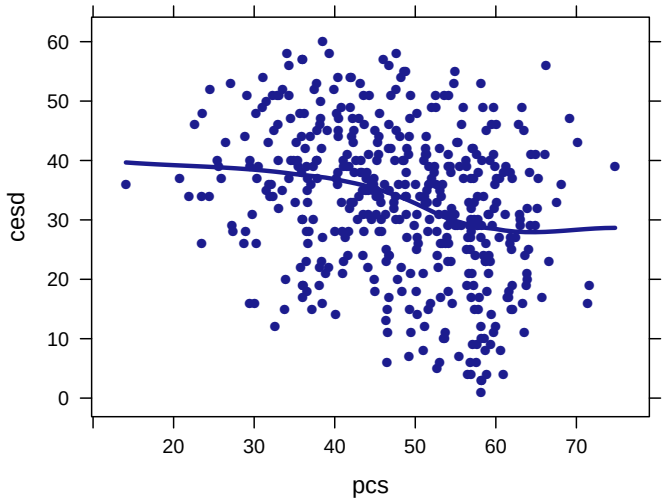


Note the use of the bar notation for computing the groupwise means.

1.3 Smoothing

Because the data for Figure 2.5 (page 93), we display how to add a scatterplot smoother to a figure using data from the HELP (Health Evaluation and Linkage to Primary Care) study.

```
> xyplot(cesd ~ pcs, type=c("p", "smooth"), lwd=3, data=HELPrct)
```



2 Correlation

Correlation can be computed using the `cor()` function. For example, we can create two random vector and compute their correlation:

```
> x = runif(100)
> y = runif(100)
> cor(x,y)

[1] -0.0873
```

Of course, because the random variables are generated independently of each other, the sample correlation should be close to zero. Moreover, you can compute the pairwise correlation coefficients for more than two vectors at one time.

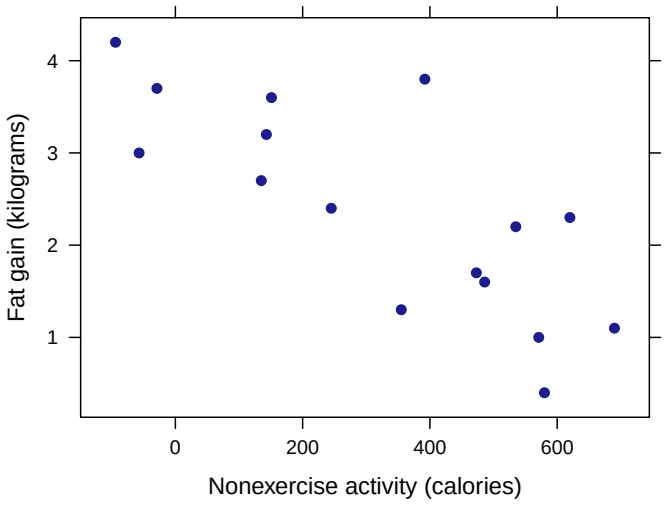
```
> z = runif(100)
> cor(data.frame(x,y,z))

        x       y       z
x  1.0000 -0.0873 -0.0683
y -0.0873  1.0000 -0.1044
z -0.0683 -0.1044  1.0000
```

The 1's along the diagonal indicate the correlation of a vector with itself.

3 Least-Squares Regression

Figure 2.11 (page 109) shows the relationship between non-exercise activity (*nea*) and fat gain after 8 weeks of overeating.

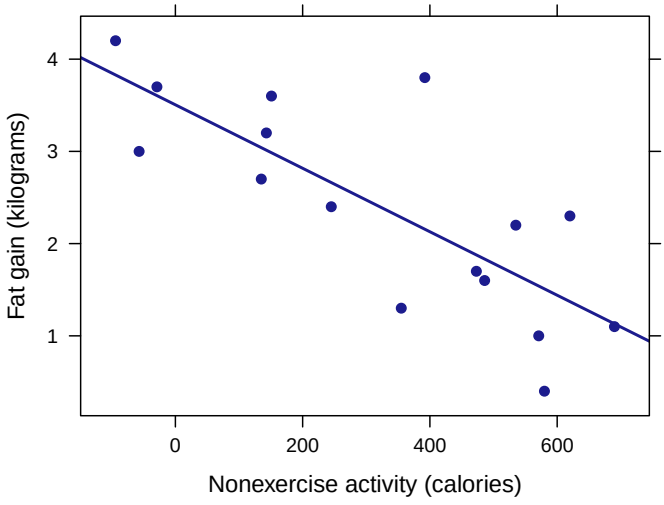
```
> Fat = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_012.csv")
> head(Fat)

  nea fat
1 -94 4.2
2 -57 3.0
3 -29 3.7
4 135 2.7
5 143 3.2
6 151 3.6

> xyplot(fat ~ nea, data=Fat, xlab="Nonexercise activity (calories)",
   ylab="Fat gain (kilograms)", pch=19)
```



3.1 Fitting a line to data

If we just want to add the least squares regression line to the plot, we can specificy the "r" value for the `type` argument to the plotting function.

```
> plotPoints(fat ~ nea, xlab="Nonexercise activity (calories)",
   ylab="Fat gain (kilograms)", pch=19, type=c("p", "r"), data=Fat)
```



However, in addition to plotting the line, we often will want to extract further information about the regression model that we built. To build the model, we use the `lm` command, and to see the coefficients from the model, we use `coef`.

```
> fm = lm(fat ~ nea, data=Fat)
> coef(fm)

(Intercept)         nea
    3.50512    -0.00344
```

3.2 Prediction

There are a few different ways of using our model, now that we've built it. A nice way is to run `makeFun()` on the model to convert it into a function. Then, we can simply ask the resulting function to compute estimate based on our model.

```
> fit.fat = makeFun(fm)
```

Note that the arguments to this function are named.

In Example 2.14 (page 111), we use the model for fat gain to estimate the fat gain for an individual whose NEA increases by 400 calories.

```
> fit.fat(nea = 400)

   1
2.13
```

This is certainly easier than calculating it manually using R as a calculator:

```
> 3.50512 - 0.00344*400

[1] 2.13
```

In Example 2.15 (page 112), we use the model for fat gain to esimate the fat gain for an individual whose NEA increases by 1500 calories.

```
> fit.fat(nea = 1500)

    1
-1.66
```

This information can also be extracted using the `predict` function. In this case, we can ask the model to return values for many different inputs at once.

```
> nea.new = data.frame(nea = c(400, 1500))
> predict(fm, newdata = nea.new)

    1     2
 2.13 -1.66
```

3.3 Least-squares regression

In Example 2.16, we verify that the coefficients returned by the regression model can be computed directly from the correlation coefficient, along with the means and standard deviations of the two variables. The correlation coefficient is:

```
> cor(fat, nea, data=Fat)

[1] -0.779
```

The means and standard deviation are given below.

```
> favstats(~fat, data=Fat)

 min   Q1 median  Q3 max mean   sd  n missing
 0.4 1.53   2.35 3.3 4.2 2.39 1.14 16       0

> favstats(~nea, data=Fat)

 min  Q1 median  Q3 max mean  sd  n missing
 -94 141    374 544 690  325 258 16       0
```

The slope of the regression line is the product of the correlation coefficient and the ratio of the standard deviations. We can verify this:

```
> coef(fm)["nea"]

     nea
-0.00344

> slope = with(Fat, cor(fat, nea) * (sd(fat) / sd(nea)))
> slope

[1] -0.00344
```

With the slope in hand, we can then compute the intercept, since we know that the point $(\bar{x}, \bar{y})$ is on the regression line.

```
> coef(fm)["(Intercept)"]

(Intercept)
       3.51

> intercept = with(Fat, mean(fat) - slope * mean(nea))
> intercept

[1] 3.51
```

3.4 Correlation and Regression

The `summary` command will show a table containing information about a regression model that is similar to the information shown in Figure 2.14 (page 116).

```
> summary(fm)


Call:
lm(formula = fat ~ nea, data = Fat)

Residuals:
   Min     1Q Median     3Q    Max
-1.109 -0.390 -0.104  0.413  1.644

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.505123   0.303616   11.54  1.5e-08 ***
nea         -0.003441   0.000741   -4.64  0.00038 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.74 on 14 degrees of freedom
Multiple R-squared:  0.606,Adjusted R-squared:  0.578
F-statistic: 21.5 on 1 and 14 DF,  p-value: 0.000381
```

For the purposes of Chapter 2 of IPS, regression is being used to describe relationships, so the primary focus is the regression parameter estimates in the first numeric column.

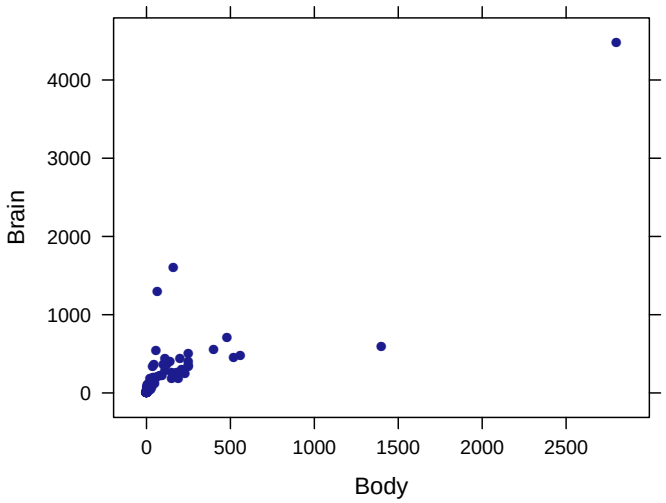
For least squares regression of one variable, the square of the correlation coefficient is equal to the coefficient of determination (or R-squared aka R^2).

```
> cor(fat, nea, data=Fat)^2

[1] 0.606

> r.squared(fm)

[1] 0.606
```

3.5 Transforming Relationships

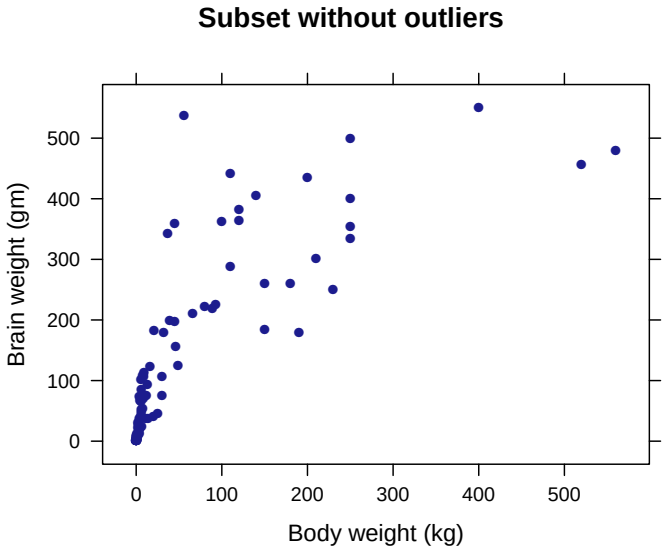
The dataset from Example 2.17 (page 119), Example 2.18 and Figure 2.18 resemble data from Chapter 9 of the Statistical Sleuth.

```
> xyplot(Brain ~ Body, data=case0902)
```



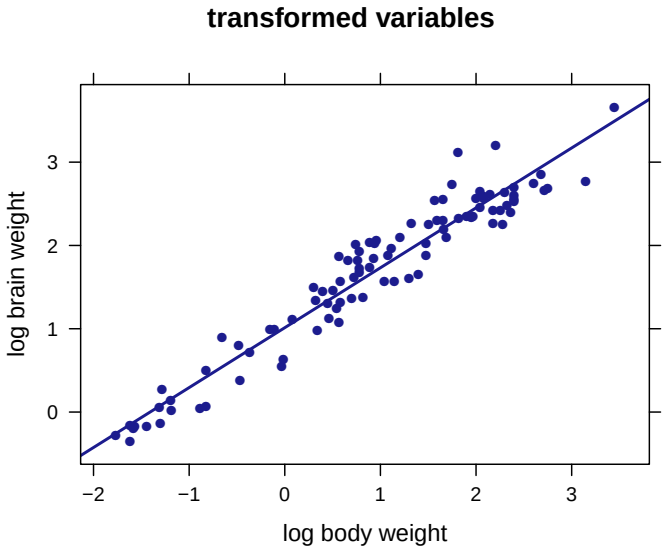
We can remove outliers, which yields a plot similar to that in Figure 2.18 (page 120).

```
> smaller = subset(case0902, Brain < 700 & Body < 1000)
> xyplot(Brain ~ Body, xlab="Body weight (kg)", ylab="Brain weight (gm)",
   main="Subset without outliers", data=smaller)
```



We can also transform the two variables, as seen in Figure 2.19 (page 121).

```
> case0902 = transform(case0902, logbrain = log10(Brain))
> case0902 = transform(case0902, logbody = log10(Body))
> xyplot(logbrain ~ logbody, xlab="log body weight", ylab="log brain weight",
   main="transformed variables", type=c("p", "r"), data=case0902)
```



4 Cautions about Correlation and Regression

The residuals can be extracted easily from a linear regression model, as displayed in Example 2.19 (page 126).

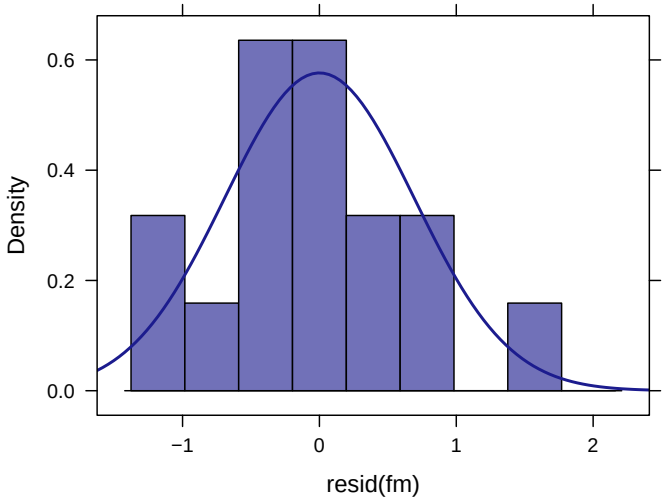
```
> resid(fm)

      1       2       3       4       5       6       7       8       9
 0.3714 -0.7013  0.0951 -0.3405  0.1870  0.6145 -0.2620 -0.9834  1.6439
     10      11      12      13      14      15      16
-0.1773 -0.2326  0.5361 -0.5400 -1.1091  0.9286 -0.0305
```

Note that the mean of the residuals is always (by mathematical construction) equal to zero.

```
> mean(resid(fm))

[1] 0
```

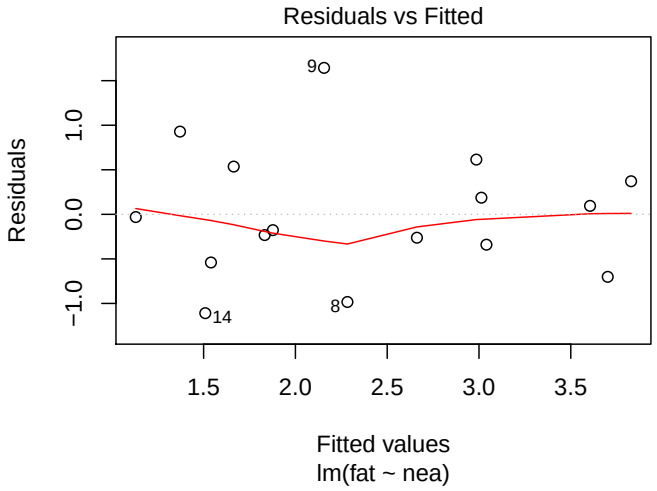
It's straightforward to display the univariate distribution of the residuals, to help assess the normality assumption.

```
> histogram(~ resid(fm), fit="normal")

Loading required package:  MASS
```



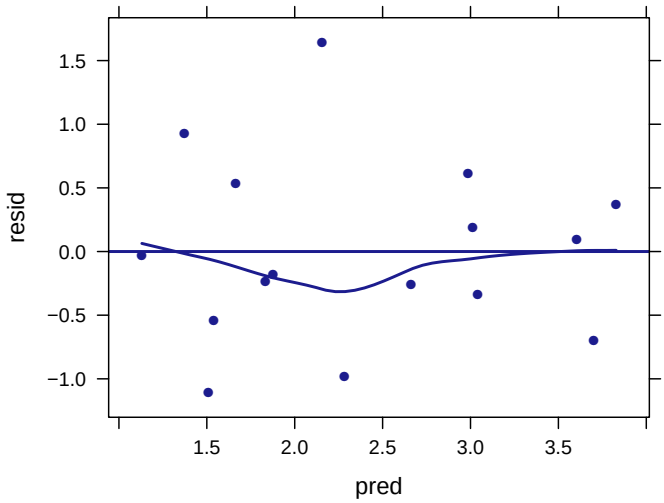
R has a built-in function for showing residual plots. You can simply `plot` the model object, and ask for only the first (of four) diagnostic plots.

```
> plot(fm, which=1)
```



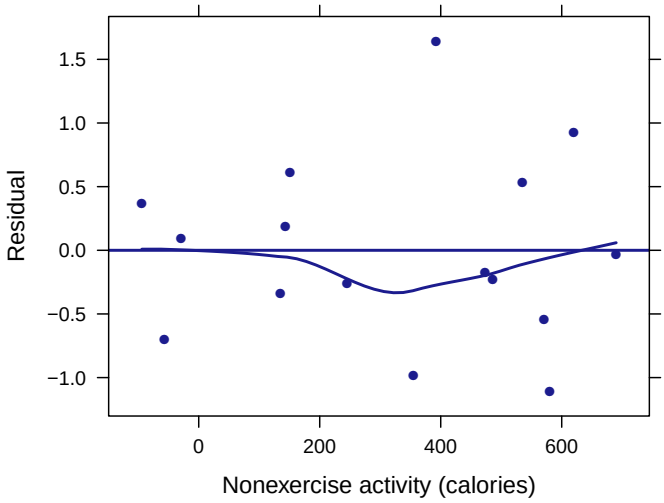
Alternatively this can be created in parts, using `plotPoints()`.

```
> Fat = transform(Fat, pred = fitted(fm))
> Fat = transform(Fat, resid = residuals(fm))
> plotPoints(resid ~ pred, type=c("p", "r", "smooth"), data=Fat)
```



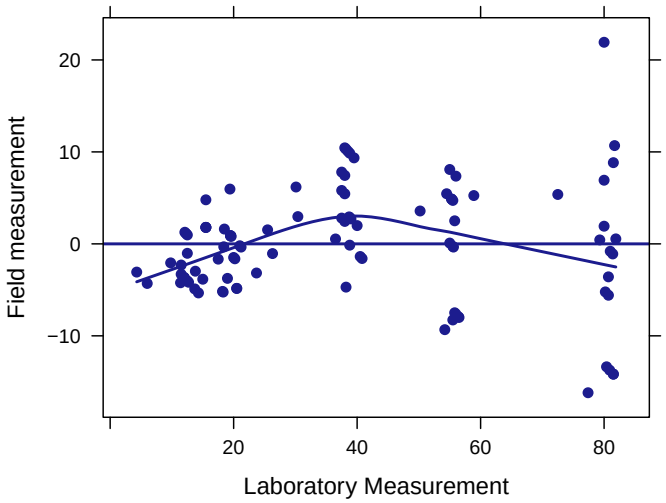
For both of these R plots the residuals against the fitted values. However, in Figure 2.20 (page 127), the residuals are plotted against the values of the explanatory variable.

```
> plotPoints(resid(fm) ~ nea, ylab="Residual",
   xlab="Nonexercise activity (calories)", type=c("p", "r", "smooth"), data=Fat)
```



A similar plot is shown in Figure 2.21 (page 128).

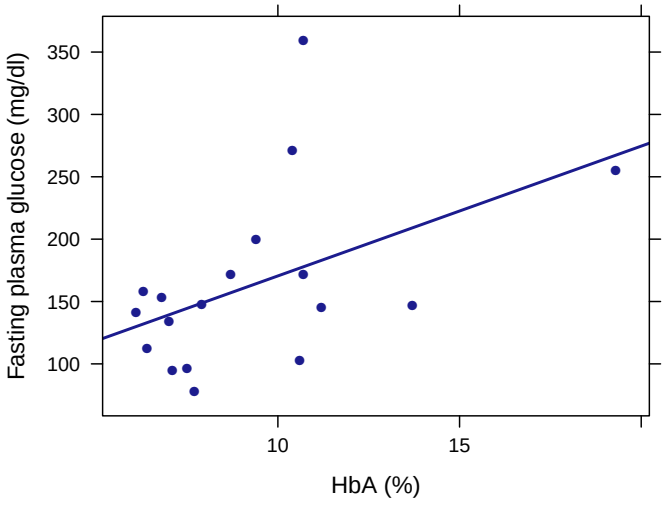
```
> fm.oil = lm(field ~ lab, data=Oil)
> plotPoints(resid(fm.oil) ~ lab, xlab="Laboratory Measurement",
   ylab="Field measurement", pch=19, type=c("p", "r", "smooth"),
   data=Oil)
```



4.1 Outliers and influential observations

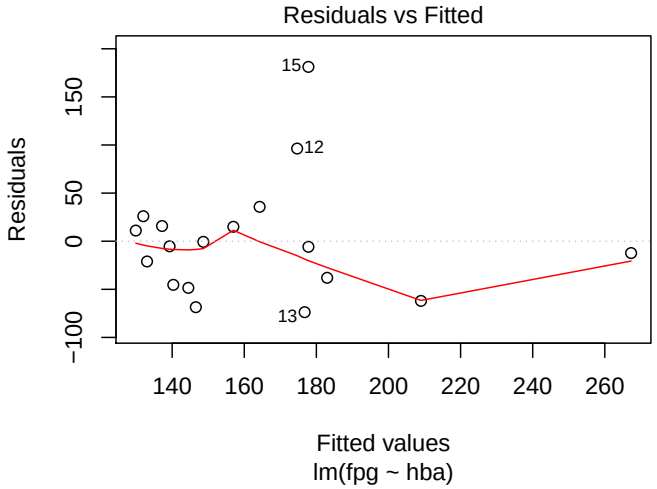
The analyses displayed in Example 2.21 (page 129) and Figures 2.22 and 2.23 (page 130) can be generated in a straightforward manner.

```
> diabetes = read.csv("http://www.math.smith.edu/ips6eR/ch02/ta02_005.csv")
> xyplot(fpg ~ hba, xlab="HbA (%)", ylab="Fasting plasma glucose (mg/dl)",
       type=c("p", "r"), data=diabetes)
```



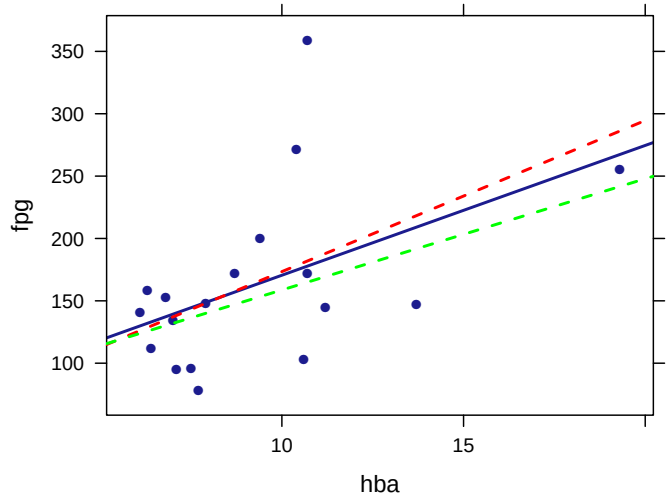

```
> outliermodel = lm(fpg ~ hba, data=diabetes)
> coef(outliermodel)
```

```
(Intercept)         hba
       66.4        10.4

> plot(outliermodel, which=1)
```



To replicate the display in Figure 2.24 (page 131), we need to fit the model twice more dropping either subject 15 or subject 18, and generating predicted lines for each of these models.

```
> no18 = lm(fpg ~ hba, data=subset(diabetes, obs != 18))
> no15 = lm(fpg ~ hba, data=subset(diabetes, obs != 15))
> plotPoints(fpg ~ hba, type=c("p", "r"), data=diabetes)
> ladd(panel.abline(no18, col="red", lty=2))
> ladd(panel.abline(no15, col="green", lty=2))
```



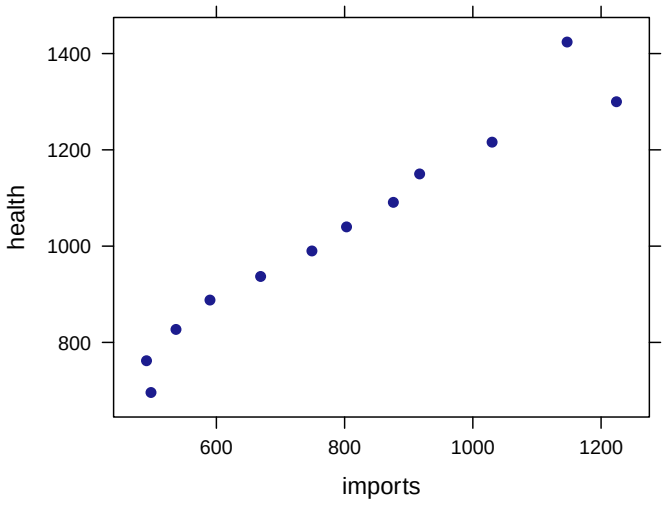
4.2 Beware the lurking variable

In Example 2.24 (page 133), the association between private health care spending and goods imported to the United States is examined each year between 1990 and 2001.

```
> Imports = read.csv("http://www.math.smith.edu/ips6eR/ch02/eg02_024.csv")
> head(Imports)

  imports health
1     498    696
2     491    762
3     537    827
4     590    888
5     669    937
6     749    990
```

Figure 2.25 (page 133) illustrates the relationship.

```
> plotPoints(health ~ imports, data=Imports, pch=19)
```



Introduction to the Practice of Statistics using R: Chapter 3

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`.

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
```

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

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 3: Producing Data.

1 Design of experiments

1.1 Randomizing subjects

It's straightforward to randomly divide 40 students into two groups of 20 students each (as described in Example 3.11 on page 185).

```
> students = 1:40    # equivalent to seq(from=1, to=40, by=1)
> group1 = sample(students, size=20)
> sort(group1)

 [1]  1  2  3  6  8 12 14 15 16 18 19 21 23 29 30 31 33 34 36 38

> group2 = students[-group1]    # all but those values are included
> sort(group2)

 [1]  4  5  7  9 10 11 13 17 20 22 24 25 26 27 28 32 35 37 39 40
```

2 Sampling design

2.1 Simple random samples

We reproduce a random sampling of resorts (from Figure 3.8, page 202).

```
> resorts = c("Aloha Kai", "Captiva", "Palm Tree", "Sea Shell", "Anchor Down",
   "Casa del Mar", "Radisson", "Silver Beach")
> # generate a SRS of size 3
> sampled = sample(resorts, size=3)
> sampled

[1] "Silver Beach" "Anchor Down"  "Sea Shell"
```

3 Toward statistical inference

3.1 Simulate a random sample

It's straightforward to use R to generate simple random samples. Example 3.32 (page 214) describes how this is done by using a table of random digits. It's more generalizable to do this with a set of possible options each with specified probabilities (probability frustrated=0.6, probability not-frustrated=0.4):

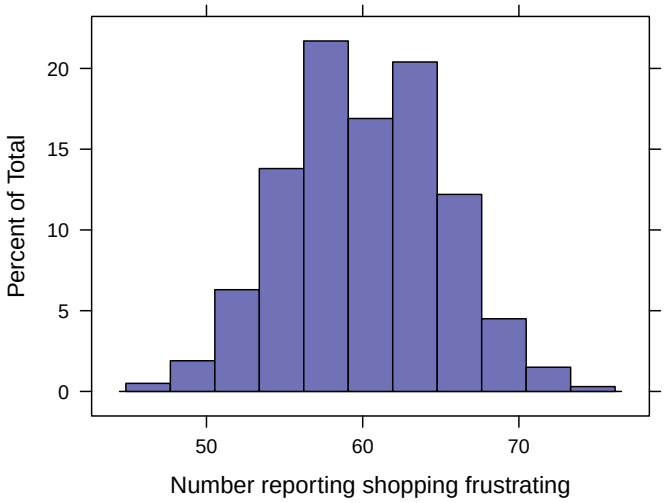
```
> srs1 = sample(c("Frustrating", "Not-frustrating"), size=100, prob=c(0.6, 0.4),
   replace=TRUE)
> tally(srs1)


    Frustrating Not-frustrating           Total
             62              38             100
```

We can repeat the process, which will (generally) give different answers.

```
> n = 100
> n

[1] 100

> tally(sample(c("Frustrating", "Not-frustrating"), size=n, prob=c(0.6, 0.4),
   replace=TRUE))


    Frustrating Not-frustrating           Total
             62              38             100

> tally(sample(c("Frustrating", "Not-frustrating"), size=n, prob=c(0.6, 0.4),
   replace=TRUE))


    Frustrating Not-frustrating           Total
             61              39             100

> tally(sample(c("Frustrating", "Not-frustrating"), size=n, prob=c(0.6, 0.4),
   replace=TRUE))


    Frustrating Not-frustrating           Total
             63              37             100
```

We can repeat the process many times using the `do()` function, which saves the results.

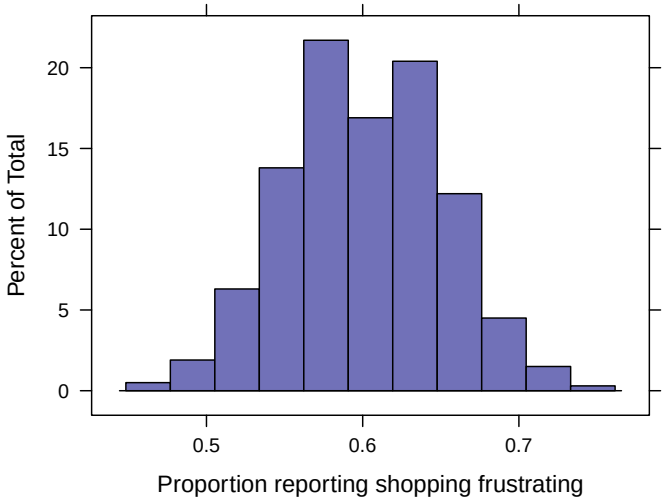
```
> res = do(1000) * tally(sample(c("Frustrating", "Not-frustrating"), size=n,
   prob=c(0.6, 0.4), replace=TRUE))
> histogram(~ Frustrating, xlab="Number reporting shopping frustrating", data=res)
```



We see that the sampling distribution for the number reporting *Frustrating* in m=1000 simple random samples each of size n=100 is centered at the value of around 60, which we would expect since the true probability of being Frustrating is in fact $p = 0.60$.

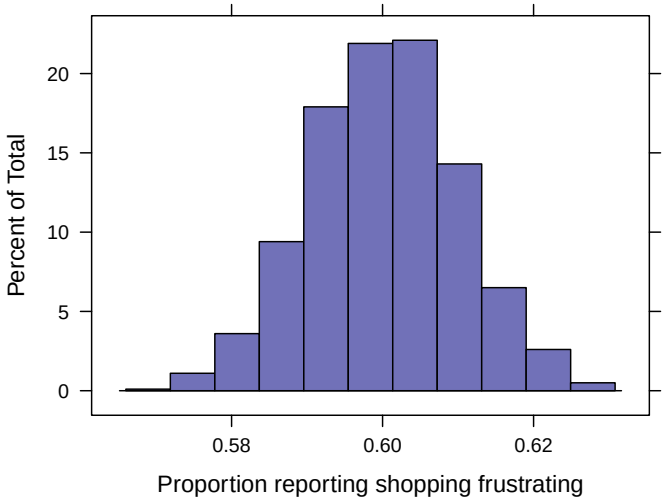
The results are equivalent if rescaled as a proportion (by dividing by the sample size).

```
> sd(~ Frustrating/n, data=res)

[1] 0.0493

> histogram(~ Frustrating/n, xlab="Proportion reporting shopping frustrating", data=res)
```



What happens if we take samples of size n=2500 (as displayed in Example 3.33, on pages 214–215).

```
> n=2500
> n

[1] 2500

> res = do(1000) * tally(sample(c("Frustrating", "Not-frustrating"), size=n,
   prob=c(0.6, 0.4), replace=TRUE))
> sd(~ Frustrating/n, data=res)

[1] 0.00985

> histogram(~ Frustrating/n, xlab="Proportion reporting shopping frustrating",
   data=res)
```



The sampling distribution is much narrower, given the much larger sample size.

3.2 Capture-recapture sampling

R can be used as a calculator, as for the calculations in Example 3.34 (page 220).

```
> 200*120/12

[1] 2000
```

Introduction to the Practice of Statistics using R: Chapter 4

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`.

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:

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

```
> require(mosaic)
```

This needs to be done once per session.
We also set some options to improve legibility of graphs and output.

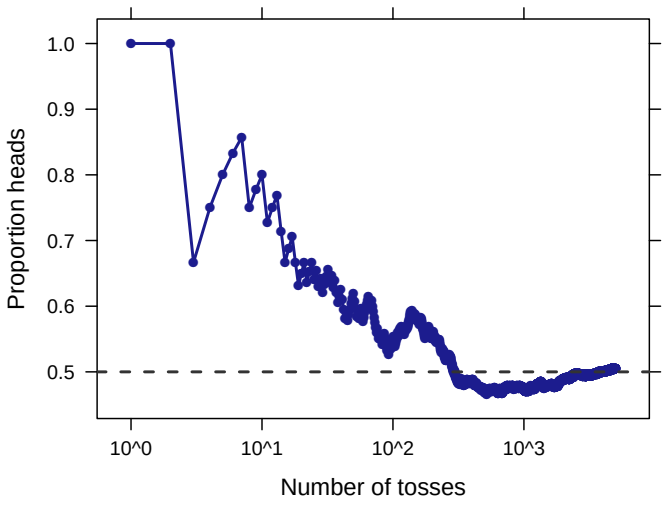
```
> trellis.par.set(theme=col.mosaic())  # get a better color scheme for lattice
> options(digits=3)
```

The specific goal of this document is to demonstrate how to replicate the analysis described in Chapter 4: Probability (The Study of Randomness).

1 Randomness

It's straightforward to replicate displays such as Figure 4.1 (page 240) using R. We begin by specifying the random number seed (this is set arbitrarily if `set.seed()` is not run), then generating a thousand coin flips (using `rbinom()`) then calculating the running average for each of the tosses. To match the Figure, we use a log scale for the x-axis.

```
> set.seed(42)
> numtosses = 5000
> runave = numeric(numtosses)
> toss = rbinom(numtosses, size=1, prob=0.50)
> for (i in 1:numtosses) {
  runave[i] = mean(toss[1:i])
 }
> xyplot(runave ~ 1:numtosses, type=c("p", "l"), scales=list(x=list(log=T)),
   ylab="Proportion heads", xlab="Number of tosses", lwd=2)
> ladd(panel.abline(h=0.50, lty=2))
```



Random digits can be sampled using the `sample()` command, as described using Table B at the bottom of page 239 (0 through 4 called *tails* or false and 5 through 9 *heads* or true:

```
> x = sample(0:9, size=10, replace=TRUE)
> x

 [1] 7 5 4 8 4 1 6 9 7 0

> x > 4

 [1]  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE
```

Alternatively, heads and tails can be generated directly.

```
> rbinom(10, size=1, prob=0.50)

 [1] 0 0 1 0 1 0 1 0 1 0
```

2 Probability models

The `mosaic` package includes support for samples of cards.

```
> Cards

 [1] "2C"  "3C"  "4C"  "5C"  "6C"  "7C"  "8C"  "9C"  "10C" "JC"  "QC"
[12] "KC"  "AC"  "2D"  "3D"  "4D"  "5D"  "6D"  "7D"  "8D"  "9D"  "10D"
[23] "JD"  "QD"  "KD"  "AD"  "2H"  "3H"  "4H"  "5H"  "6H"  "7H"  "8H"
[34] "9H"  "10H" "JH"  "QH"  "KH"  "AH"  "2S"  "3S"  "4S"  "5S"  "6S"
[45] "7S"  "8S"  "9S"  "10S" "JS"  "QS"  "KS"  "AS"
```

Let's create a deck which is missing an ace (to verify the calculation on page 252):

```
> noacespades = subset(Cards, Cards != "AS")
> noacespades

 [1] "2C"  "3C"  "4C"  "5C"  "6C"  "7C"  "8C"  "9C"  "10C" "JC"  "QC"
[12] "KC"  "AC"  "2D"  "3D"  "4D"  "5D"  "6D"  "7D"  "8D"  "9D"  "10D"
[23] "JD"  "QD"  "KD"  "AD"  "2H"  "3H"  "4H"  "5H"  "6H"  "7H"  "8H"
[34] "9H"  "10H" "JH"  "QH"  "KH"  "AH"  "2S"  "3S"  "4S"  "5S"  "6S"
[45] "7S"  "8S"  "9S"  "10S" "JS"  "QS"  "KS"
```

How often is the next card also an ace? We know that the true answer is 3/51 (or 0.059), and we can estimate this through sampling.

```
> res = do(10000) * sample(noacespades, size=1, replace=TRUE)
> head(res)

  result
1    10C
2     8H
3     KS
4     4H
5     9D
6     KC

> tally(~ (result %in% c("AD", "AC", "AH")), format="percent", data=res)


  TRUE  FALSE  Total
  5.42  94.58 100.00
```

3 Random variables

Example 4.23 (page 261) derives the Binomial distribution when $n = 4$ and $p = 0.50$.

```
> dbinom(0:4, size=4, prob=0.50)   # probability mass function

[1] 0.0625 0.2500 0.3750 0.2500 0.0625

> pbinom(0:4, size=4, prob=0.50)   # cumulative probability

[1] 0.0625 0.3125 0.6875 0.9375 1.0000
```

Example 4.24 (page 262) asks about the probability of at least two heads, which is equivalent to one minus the probability of no more than one head, or $P(X = 2) + P(X = 3) + P(X = 4)$.

```
> dbinom(2:4, size=4, prob=0.50)

[1] 0.3750 0.2500 0.0625

> sum(dbinom(2:4, size=4, prob=0.50))

[1] 0.688

> 1 - pbinom(1, size=4, prob=0.50)

[1] 0.688
```

Calculations for Uniform random variables can be undertaken as easily (as seen in Example 4.25, page 263):

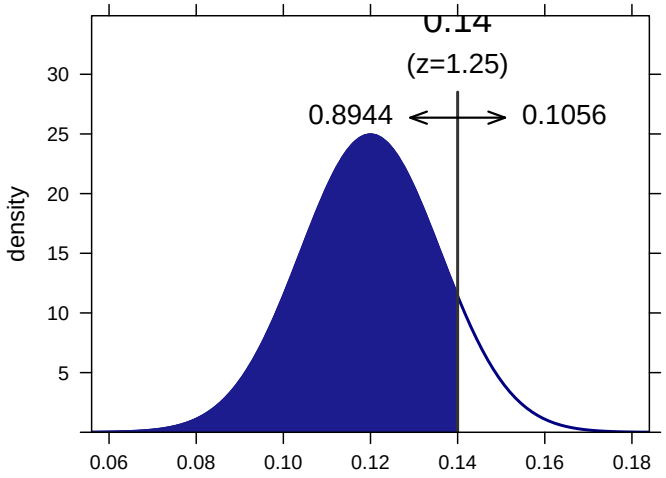
```
> punif(0.7, min=0, max=1)

[1] 0.7

> punif(0.3, min=0, max=1)

[1] 0.3

> punif(0.7, min=0, max=1) - punif(0.3, min=0, max=1)

[1] 0.4
```

Simulation studies are also easy to carry out:

```
> randnums = runif(10000, min=0, max=1)
> head(randnums)

[1] 0.0416 0.0387 0.3144 0.5852 0.4764 0.9085

> tally(~ (randnums > 0.3 & randnums < 0.7), format="percent")


 TRUE FALSE Total
 40.1  59.9 100.0
```

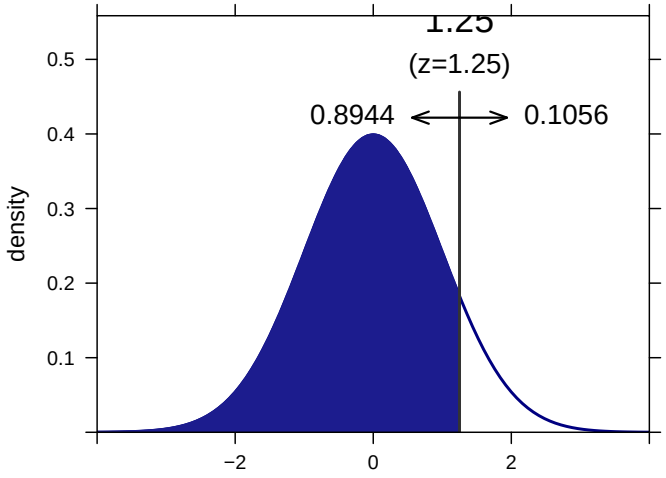
Example 4.26 (pages 265–266) displays the same type of calculation for a normal random variable:

```
> xpnorm(0.14, mean=0.12, sd=0.016)


If X ~ N(0.12,0.016), then

P(X <= 0.14) = P(Z <= 1.25) = 0.8944
P(X >  0.14) = P(Z >  1.25) = 0.1056
[1] 0.894

> pnorm(0.10, mean=0.12, sd=0.016)

[1] 0.106

> pnorm(0.14, mean=0.12, sd=0.016) - pnorm(0.10, mean=0.12, sd=0.016)

[1] 0.789
```



or on the normalized scale:

```
> xpnorm(1.25, mean=0, sd=1)


If X ~ N(0,1), then

P(X <= 1.25) = P(Z <= 1.25) = 0.8944
P(X >  1.25) = P(Z >  1.25) = 0.1056
[1] 0.894

> pnorm(-1.25, mean=0, sd=1)

[1] 0.106

> pnorm(1.25, mean=0, sd=1) - pnorm(-1.25, mean=0, sd=1)

[1] 0.789
```



4 Means and variances of random variables

Example 4.29 (page 272) calculates the mean of the first digits following Benford's law:

```
> V = 1:9
> probV = c(0.301, 0.176, 0.125, 0.097, 0.079, 0.067, 0.058, 0.051, 0.046)
> sum(probV)

[1] 1

> xyplot(probV ~ V, xlab="Outcomes", ylab="Probability")
> V*probV

[1] 0.301 0.352 0.375 0.388 0.395 0.402 0.406 0.408 0.414

> benfordmean = sum(V*probV)
> benfordmean

[1] 3.44
```

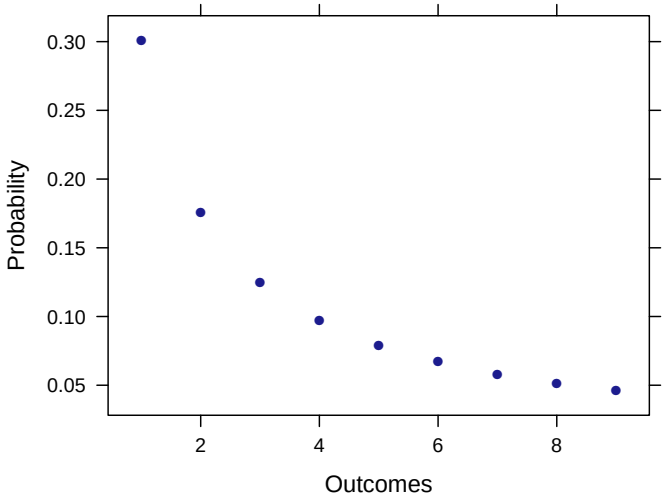
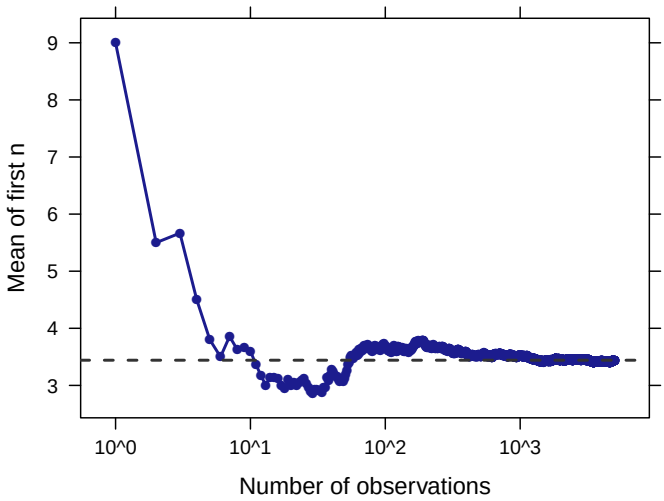


Figure 4.14 (page 275) describes the law of large numbers in action. We can display this for samples from the Benford distribution:

```
> runave = numeric(numtosses)
> benford = sample(V, size=numtosses, prob=probV, replace=TRUE)
> for (i in 1:numtosses) {
  runave[i] = mean(benford[1:i])
 }
> xyplot(runave ~ 1:numtosses, type=c("p", "l"), scales=list(x=list(log=T)),
   ylab="Mean of first n", xlab="Number of observations", lwd=2)
> ladd(panel.abline(h=3.441, lty=2))
```



The variance (introduced on page 280) can be carried out in a similar fashion:

```
> sum((V - benfordmean)^2 * probV)

[1] 6.06
```

Note that we can estimate this value from the variance of the simulated samples above:

```
> var(benford)

[1] 6.11
```

Similar calculations can be undertaken on either the original or linearly transformed scale for the Tri-State pick 3 lottery example (4.34) on page 282:

```
> X = c(0, 500)
> probX = c(0.999, 0.001)
> xmean = sum(X*probX)
> xmean

[1] 0.5

> sum((X - xmean)^2 * probX)

[1] 250
```

For Example 4.35 (page 283), since $W = X - 1$ we know that $\mu_w = \mu_x - 1$:

```
> W = X - 1
> wmean = sum(W*probX)
> wmean

[1] -0.5

> sum((W - wmean)^2 * probX)

[1] 250
```

Introduction to the Practice of Statistics using R: Chapter 5

Nicholas J. Horton* Ben Baumer

April 8, 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`.

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.

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

```
> 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 5: Sampling Distributions.

1 Sampling distributions for counts and proportions

Calculations with the binomial distribution can be undertaken using the `pbinom()` and `dbinom()` functions. For example, the results from Figure 5.1 (page 317) can be reproduced.

```
> dbinom(10, size=150, prob=0.08)

[1] 0.107

> pbinom(10, size=150, prob=0.08)

[1] 0.338
```

The table Figure 5.2 (page 318) can be reproduced using the following command:

```
> cbind(0:9, dbinom(0:9, size=15, p=0.08))

      [,1]     [,2]
 [1,]    0 2.86e-01
 [2,]    1 3.73e-01
 [3,]    2 2.27e-01
 [4,]    3 8.57e-02
 [5,]    4 2.23e-02
 [6,]    5 4.27e-03
 [7,]    6 6.19e-04
 [8,]    7 6.93e-05
 [9,]    8 6.02e-06
[10,]    9 4.07e-07
```

And the calculation on page 318 using the command:

```
> sum(dbinom(0:1, size=15, prob=0.08))

[1] 0.66
```

or using `pbinom()`:

```
> pbinom(1, size=15, prob=0.08)

[1] 0.66
```

Example 5.9 (pages 318-319) can be calculated:

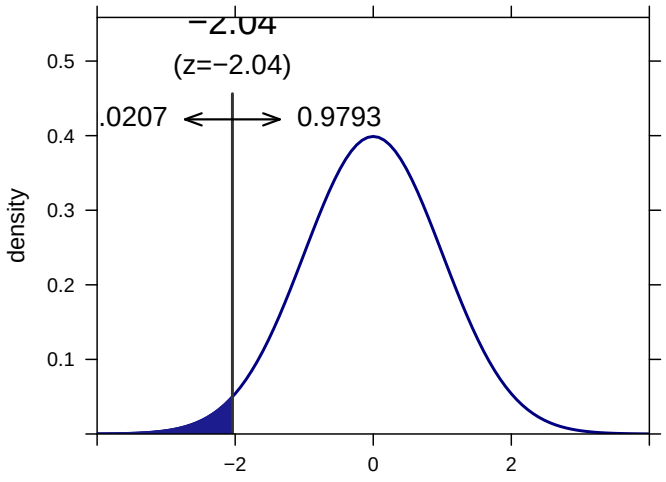
```
> 1 - pbinom(4, size=12, prob=0.25)

[1] 0.158
```

Using R, there is little need for the normal approximation to the binomial. As an example, the probability of interest in Example 5.11 (page 321) can be calculated using the command:

```
> 1 - pbinom(1449, size=2500, prob=0.6)

[1] 0.98
```

Example 5.13 (page 324) uses the normal approximation nonetheless:

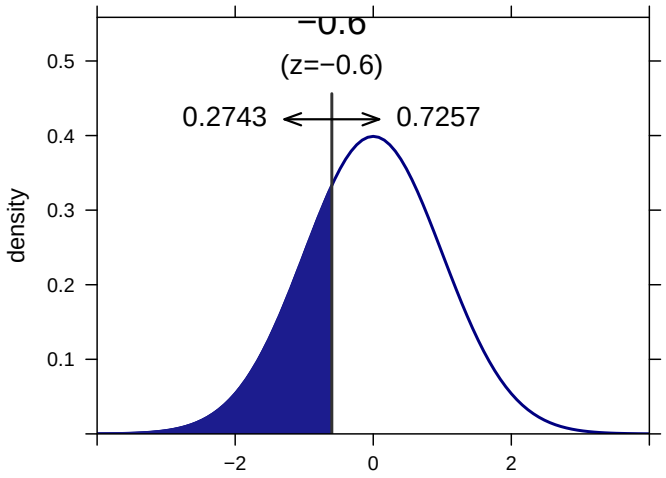
```
> xpnorm(-2.04, mean=0, sd=1)


If X ~ N(0,1), then

P(X <= -2.04) = P(Z <= -2.04) = 0.0207
P(X >  -2.04) = P(Z >  -2.04) = 0.9793
[1] 0.0207
```



A similar calculation is done in Example 5.14 (page 325):

```
> xpnorm(-0.60, mean=0, sd=1)


If X ~ N(0,1), then
```

```
P(X <= -0.6) = P(Z <= -0.6) = 0.2743
P(X >  -0.6) = P(Z >  -0.6) = 0.7257
[1] 0.274
```



We can compare this to the exact calculation:

```
> pbinom(10, size=150, prob=0.08)

[1] 0.338
```

The approximation isn't great (which is a good reason not to use it).

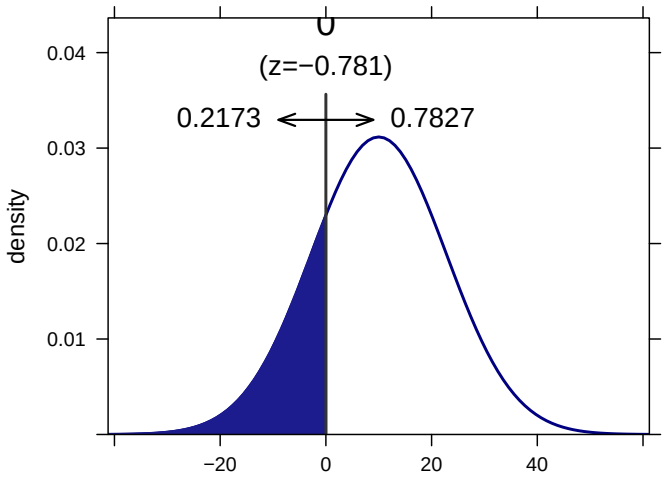
The binomial probability formula (Example 5.16, page 329) can be used to calculate probabilities, or `dbinom()` and `pbinom()` can do the trick:

```
> dbinom(0, size=15, prob=0.08)

[1] 0.286

> dbinom(1, size=15, prob=0.08)

[1] 0.373

> dbinom(0, size=15, prob=0.08) + dbinom(1, size=15, prob=0.08)

[1] 0.66

> pbinom(1, size=15, prob=0.08)

[1] 0.66
```

2 Sampling distributions for a sample mean

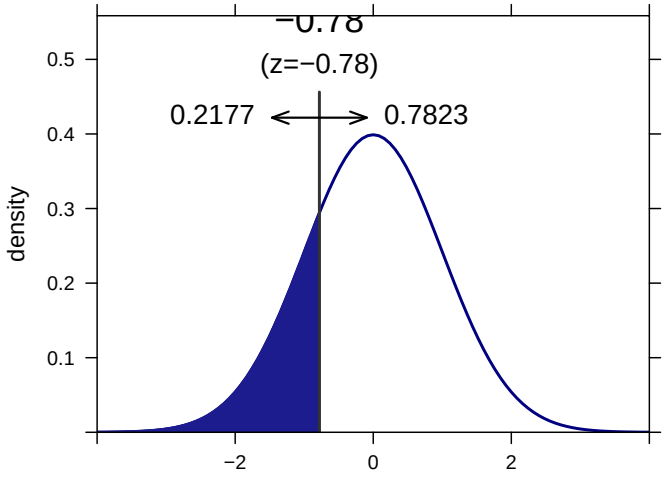
Similar approaches are used for sampling distributions for a sample mean, using `xpnorm()`. For instance, Example 5.24 (page 343) can be found using:

```
> xpnorm(0, mean=10, sd=12.8)


If X ~ N(10,12.8), then

P(X <= 0) = P(Z <= -0.781) = 0.2173
P(X >  0) = P(Z >  -0.781) = 0.7827
[1] 0.217
```



or

```
> xpnorm(-0.78, mean=0, sd=1)


If X ~ N(0,1), then

P(X <= -0.78) = P(Z <= -0.78) = 0.2177
P(X >  -0.78) = P(Z >  -0.78) = 0.7823
[1] 0.218
```


−0.78
(z=−0.78)
0.2177
0.7823
density
0.5
0.4
0.3
0.2
0.1
−2
0
2

Introduction to the Practice of Statistics using R: Chapter 6

Ben Baumer Nicholas J. Horton*

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`.

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:

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

```
> 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 6: Introduction to Inference.

1 Estimating with Confidence

First, let's generate a random sample of 500 SAT scores drawn from a normal distribution with mean 500 and standard deviation 100. To do this we use the `rnorm()` function, which draws from a normal distribution.

```
> mu = 500
> sigma = 100
> x = rnorm(500, mean=mu, sd=sigma)
> favstats(x)

 min  Q1 median  Q3 max mean   sd   n missing
 195 430    500 566 773  500 98.6 500       0
```

To compute a confidence interval for the mean, we'll use a simple function that finds a confidence interval for the mean of any vector of data x, given a specified significance level and the true (assumed known) population standard deviation. Note that 95% is the default level of confidence.

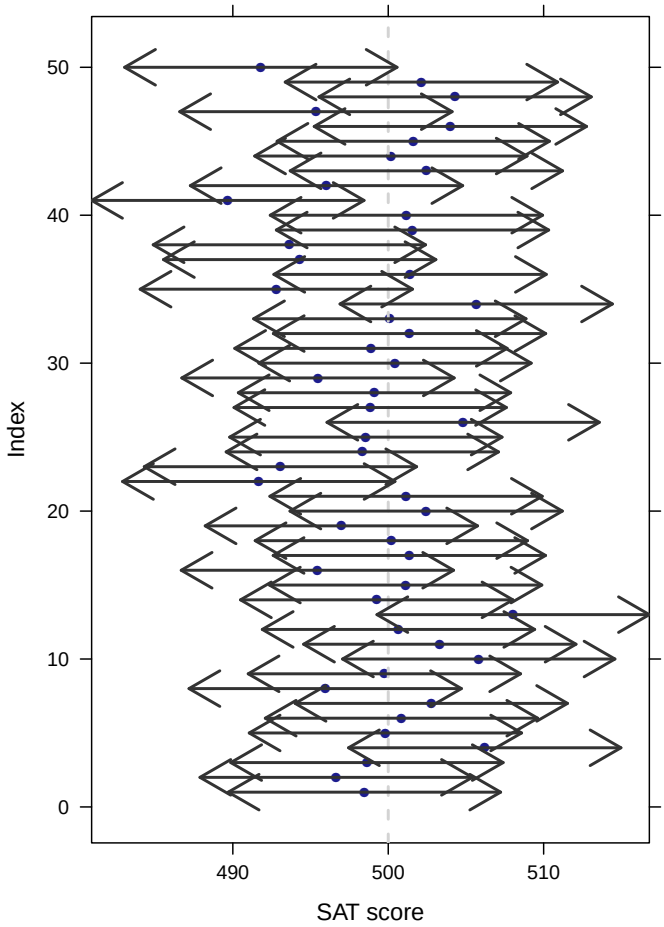
```
> meanconfint = function (x, sigma, level = 0.95, ...) {
   se = sigma / sqrt(length(x))
   mu = mean(x)
   z = qnorm(1 - (1 - level)/2)
   out = c(mu, mu - z * se, mu + z * se)
   names(out) = c("mean", "lower", "upper")
   return(out)
 }
> meanconfint(x, sigma = sigma)

 mean lower upper
  500   492   509
```

At the bottom of page 358, many such confidence intervals are calculated. We can simulate this using our function. The `do()` function will repeat any operation a specified number of times, and return a data frame of the results. The `apply()` family of functions provide a powerful way to apply an operation to the rows or columns of a data frame. Here it lets us repeat an operation for each of the 50 sets of 500 random numbers.

```
> randomx = do(50) * rnorm(500, mean=mu, sd=sigma)
> ci = data.frame(t(apply(randomx, 1, meanconfint, sigma=sigma)))
> head(ci, 3)

  mean lower upper
1  498   490   507
2  497   488   505
3  499   490   507
```

Let's try to visualize these intervals in a manner analogous to the plot on the bottom of page 358.

```
> xyplot(1:nrow(ci) ~ mean, data=ci, xlim=range(ci), xlab="SAT score", ylab="Index")
> ladd(panel.abline(v=500, col="lightgray", lty=2))
> ladd(with(ci, panel.arrows(x0 = lower, y0=1:nrow(ci), y1=1:nrow(ci), cex=0.5,
   x1=upper, code=3)))
```



We see that sometimes (e.g. simulation 41) the confidence interval does *not* cover the true population mean (500 points).

Note that we can consider confidence levels other than 0.95 by specifying the `level` argument. Here's how we compute a 90% confidence interval.

```
> head(t(apply(randomx, 1, meanconfint, sigma=sigma, level=0.9)), 3)

     mean lower upper
[1,]  498   491   506
[2,]  497   489   504
[3,]  499   491   506
```

The 90% confidence intervals are narrower than the 95% confidence intervals, since we sacrifice some accuracy in exchange for increased confidence that the interval will contain the true mean.

In Example 6.4 (page 361), we are asked to compute a 95% confidence interval for a sample mean of \$18,900 in undergraduate debt, computed from a sample of 1280 borrowers. The standard deviation of the population is known to be \$49,000. Since we want a 95% confidence interval, we need to find the z-score that corresponds to 0.025 (or equivalently 0.0975), since 95% of the standard normal distribution lies between these two values.

```
> z.star = qnorm(0.975)
> z.star

[1] 1.96
```

Then we compute the margin or error and the confidence interval by:

```
> se = z.star * (49000) / sqrt(1280)
> se

[1] 2684

> 18900 + c(-se, se)

[1] 16216 21584
```

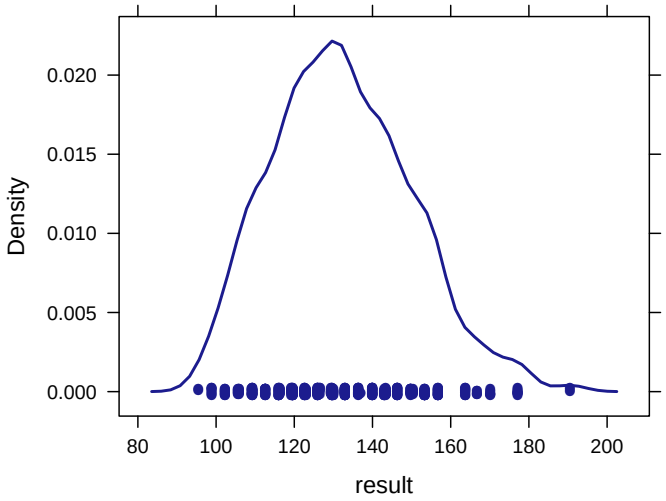
In Example 6.6 (page 364), we change the confidence level to 99%. Thus, we need to compute a different value of z^*.

```
> z.star2 = qnorm(0.995)
> z.star2

[1] 2.58

> se2 = z.star2 * (49000) / sqrt(1280)
> se2

[1] 3528

> 18900 + c(-se2, se2)

[1] 15372 22428
```

1.1 Beyond the Basics

We'll discuss the bootstrap in much greater detail in Chapter 16. Here, we can use the `resample()` function from `mosaic` to quickly compute a bootstrap sample.

```
> time = c(190.5, 109, 95.5, 137)
> resample(time)

[1] 190.5  95.5 190.5 109.0

> bootstrap = do(1000) * mean(resample(time))
> densityplot(~result, data=bootstrap)
```



2 Tests of Significance

In Example 6.12 (page 378), we compute a p-value for the observed difference of \$4,100. Note that we need to multiply the cumulative probability in the right-hand tail by 2 for a two-sided test.

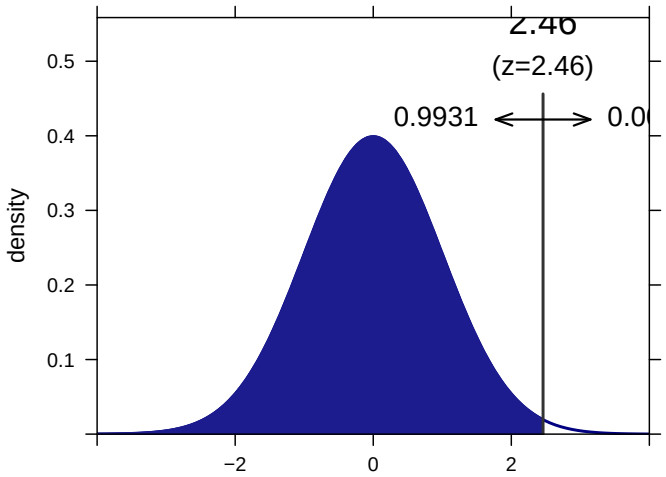
```
> z = (4100 - 0) / 3000
> z

[1] 1.37

> 2 * (1 - pnorm(z))

[1] 0.172
```

The z-test for a population mean on page 383 can be computed using the `pbinom()`.

```
> # one-sided test for right tail probability
> pnorm(2, lower.tail=FALSE)

[1] 0.0228

> # one-sided test for left tail probability
> pnorm(-2)

[1] 0.0228

> # two-sided test
> 2 * pnorm(2, lower.tail=FALSE)

[1] 0.0455
```

In Example 6.16 (page 385), we find the right-hand tail probability.

```
> pnorm(461, mean=450, sd=100 / sqrt(500), lower.tail=FALSE)

[1] 0.00695

> xpnorm(2.46)


If X ~ N(0,1), then

P(X <= 2.46) = P(Z <= 2.46) = 0.9931
P(X >  2.46) = P(Z >  2.46) = 0.0069
[1] 0.993
```



3 Use and Abuse of Tests

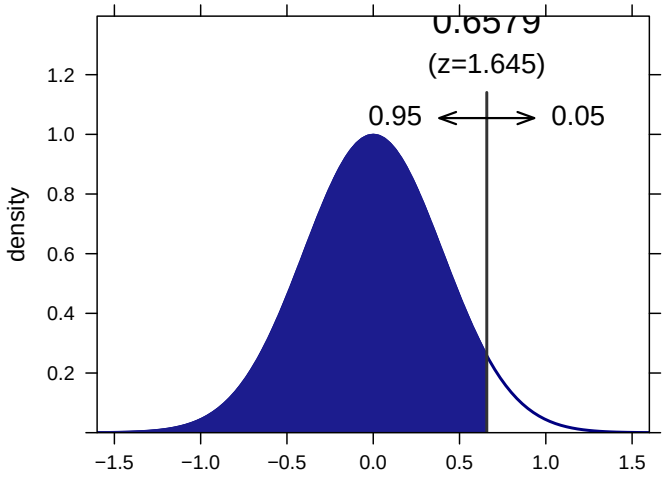
In Example 6.84 (page 396), we test for significance. Note the use of a one-sided test.

```
> z1 = (541.4 - 525) / (100 / sqrt(100))
> pnorm(z1, lower.tail=FALSE)

[1] 0.0505

> z2 = (541.5 - 525) / (100 / sqrt(100))
> pnorm(z2, lower.tail=FALSE)

[1] 0.0495
```

4 Power and Inference as a Decision

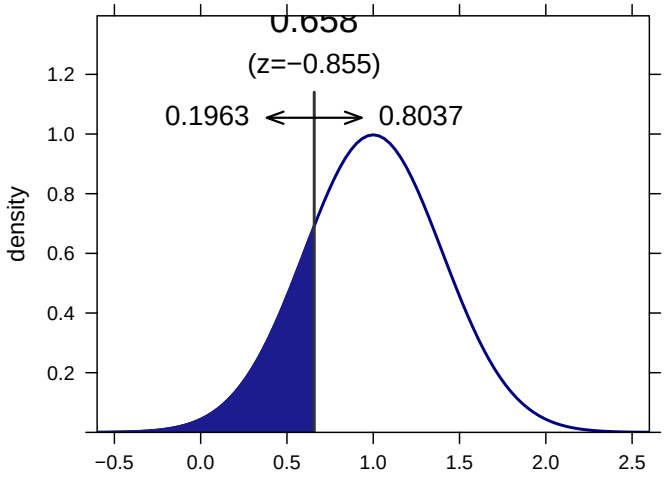
Example 6.29 (page 402) considers the power for a study with n=25 subjects, where a one-sided alternative is tested at an α level of 0.05 and the population standard deviation is assumed known and equals $\sigma = 2$.

```
> xqnorm(.95, mean=0, sd=2/sqrt(25))

P(X <= 0.657941450780589) = 0.95
P(X >  0.657941450780589) = 0.05
[1] 0.658
```



We can now compare this to the distribution when the alternative is true ($\mu = 1$).

```
> xpnorm(0.658, mean=1, sd=2/sqrt(25))


If X ~ N(1,0.4), then

P(X <= 0.658) = P(Z <= -0.855) = 0.1963
P(X >  0.658) = P(Z >  -0.855) = 0.8037
[1] 0.196
```



We see that the power is 0.80.

Introduction to the Practice of Statistics using R: Chapter 7

Nicholas J. Horton* Ben Baumer

March 29, 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* (2002) 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`.

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:

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

```
> 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 7: Inference for Distributions.

1 Inference for the mean of a population

It is straightforward to undertake inference for a single population using R. For example, the results from Example 7.1 (page 421) can be reproduced using the following commands

```
> x = c(5,6, 0, 4, 11, 9, 2, 3)
> favstats(x)

 min   Q1 median   Q3 max mean   sd n missing
   0 2.75    4.5 6.75  11    5 3.63 8       0

> length(x)

[1] 8

> tstar = qt(.975, df=length(x) - 1)
> tstar

[1] 2.36

> moe = sd(x) / sqrt(length(x))
> moe

[1] 1.28

> mean(x) + c(-tstar, tstar) * moe

[1] 1.97 8.03
```

As the authors note (page 421), we are 95% confident that the US population's average time spent listening to full-track music on a cell phone is between 2.0 and 8.0 hours per month. Since

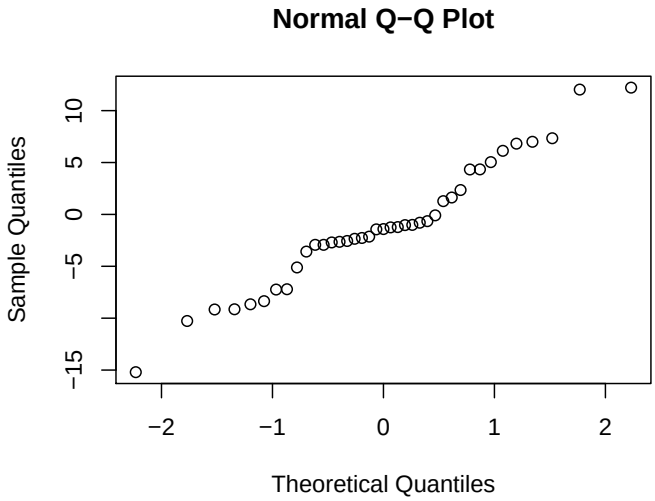
this interval does not contain the null value of 8.3 hours, these data suggest that on average, a US subscriber listens to less full-track music.

Example 7.2 (pages 422–423) continues this example using a one-sample t-test, specifically assessing whether the mean in the US is different than 8.3 hours.

```
> t = (mean(x) - 8.3) / (sd(x)/sqrt(length(x)))
> t

[1] -2.57

> pt(t, df=length(x) - 1)   # one tail

[1] 0.0184

> 1 - pt(abs(t), df=length(x) - 1)   # right tail

[1] 0.0184

> 2* pt(t, df=length(x) - 1)   # two sided test (since our statistic was negative)

[1] 0.0368

> 2*(1 - pt(abs(t), df=length(x) - 1))   # two sided test (this always works)

[1] 0.0368
```

Example 7.4 (pages 424–425) considers a one-sample test of stock portfolio diversification.

```
> ds = read.csv("http://www.math.smith.edu/ips6eR/ch07/ta07_001.csv")
> favstats(~ return, data=ds)

   min    Q1 median   Q3  max mean   sd  n missing
 -15.2 -3.25  -1.41 1.99 12.2 -1.1 5.99 39       0

> with(ds, qqnorm(return))
> with(ds, t.test(return-0.95))


One Sample t-test

data:  return - 0.95
t = -2.14, df = 38, p-value = 0.03914
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -3.989 -0.107
sample estimates:
mean of x
    -2.05
```



Example 7.7 (pages 428–429) considers whether there is a statistically significant difference in the aggressive behaviors of dementia patients on moon days vs. other days.

```
> ds = read.csv("http://www.math.smith.edu/ips6eR/ch07/ta07_002.csv")
> favstats(~ aggdiff, data=ds)

   min   Q1 median   Q3  max mean   sd  n missing
 -0.02 1.84   2.68 3.35 4.41 2.43 1.46 15       0

> with(ds, stem(aggdiff))


  The decimal point is at the |

  -0 | 0
   0 | 11
   1 | 6
   2 | 1347
   3 | 11167
   4 | 44

> with(ds, t.test(aggdiff))


One Sample t-test

data:  aggdiff
t = 6.45, df = 14, p-value = 1.518e-05
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
```

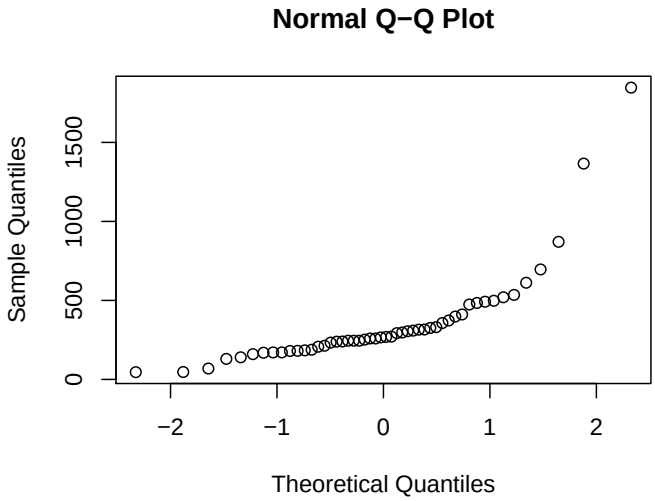
```
 1.62 3.24
sample estimates:
mean of x
     2.43
```

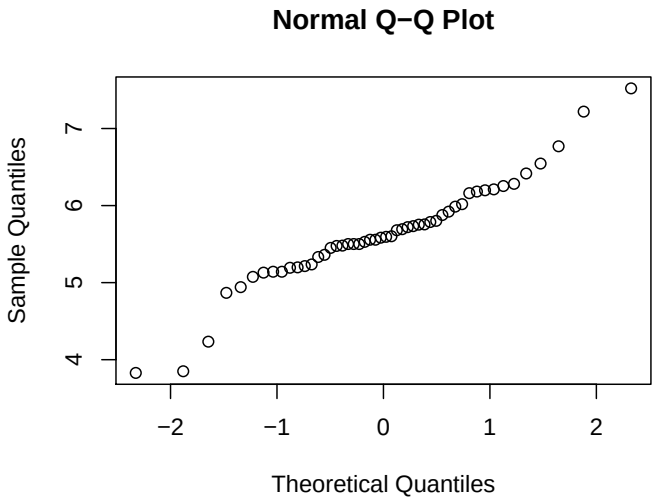
These results are consistent with those from the text.

1.1 Transformations

Example 7.11 (pages 436–437) considers the length of audio files on an iPod (which are dramatically right skewed). A log transformation is indicated (as seen below).

```
> ds = read.csv("http://www.math.smith.edu/ips6eR/ch07/ta07_003.csv")
> names(ds)

[1] "min"        "sec"        "total_secs"

> ds = transform(ds, logtotal = log(total_secs))
> with(ds, qqnorm(total_secs))
> with(ds, qqnorm(logtotal))
```






```
> with(ds, t.test(logtotal))


One Sample t-test

data:  logtotal
t = 58.2, df = 49, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 5.44 5.83
sample estimates:
mean of x
     5.63
```

1.2 Sign test

The sign test can be undertaken using the `pbinom()` command (as described in Example 7.12 on pages 438–439).

```
> 1 - pbinom(13, size=15, prob=0.5)

[1] 0.000488
```

2 Comparing two means

Example 7.14 (pages 450–453) compares the DRP scores from two samples of third-graders randomly assigned to a treatment group and control group. We can replicate the parts and pieces of this comparison as well as undertake a two sample (equal variance) t-test.

```
> ds = read.csv("http://www.math.smith.edu/ips6eR/ch07/ta07_004.csv")
> mean(drp ~ group, data=ds)

Control   Treat
   41.5    51.5

> diff(mean(drp ~ group, data=ds))

Treat
 9.95

> sd(drp ~ group, data=ds)

Control   Treat
   17.1    11.0
```

We wouldn't undertake a one-sided test, but that's the approach suggested by the authors (page 453):

```
> t.test(drp ~ group, alternative="less", data=ds)


Welch Two Sample t-test

data:  drp by group
t = -2.31, df = 37.9, p-value = 0.01319
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
  -Inf -2.69
sample estimates:
mean in group Control   mean in group Treat
                 41.5                  51.5
```

2.1 Confidence interval

Example 7.15 (page 454) calculates a 95% confidence interval for the mean improvement in the entire population of third-graders.

```
> t.test(drp ~ group, data=ds)   # to match bottom of page 452


Welch Two Sample t-test

data:  drp by group
t = -2.31, df = 37.9, p-value = 0.02638
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -18.68  -1.23
sample estimates:
mean in group Control   mean in group Treat
                 41.5                  51.5
```

This interval matches the results at the top of page 455.

2.2 Pooled two-sample t procedure

While not generally recommended, the pooled two sample procedures can be fit within R. By default the *unequal* variance test is calculated. The `var.equal=` option can be set to change this. (But first note the error in the dataset, where the eighth placebo subject is miscoded (should be 114, 112, decrease=2, not -2).

```
> ds = read.csv("http://www.math.smith.edu/ips6eR/ch07/ta07_005.csv")
> ds[18,]

   id   group g beg end dec
18 18 Placebo 1 112 114  -2

> ds[18,"dec"] = 2    # note error from table 7.5 page 463
> ds[18,]

   id   group g beg end dec
18 18 Placebo 1 112 114   2

> t.test(dec ~ group, alternative="greater", var.equal=TRUE, data=ds)


Two Sample t-test

data:  dec by group
t = 1.63, df = 19, p-value = 0.05935
```

```
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.307    Inf
sample estimates:
mean in group Calcium mean in group Placebo
                5.000                -0.273
```

Again: the one-sided test is used (though we might question this).

To get a two-sided confidence interval, we fit the t.test without the `alternative=` option:

```
> t.test(dec ~ group, var.equal=TRUE, data=ds)


Two Sample t-test

data:  dec by group
t = 1.63, df = 19, p-value = 0.1187
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.48 12.03
sample estimates:
mean in group Calcium mean in group Placebo
                5.000                -0.273
```

We would probably still suggest reporting the unequal variance (unpooled) results:

```
> t.test(dec ~ group, var.equal=FALSE, data=ds)


Welch Two Sample t-test

data:  dec by group
t = 1.6, df = 15.6, p-value = 0.1288
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.71 12.26
sample estimates:
mean in group Calcium mean in group Placebo
                5.000                -0.273
```

These are slighter wide (but require less unverifiable assumptions).

3 Optional topics

3.1 F test for equality of spread

While we don't recommend using the test for equality for spread, this is straightforward to undertake in R. The values displayed on page 475 can be generated with the command:

```
> qf(c(0.90, 0.95, 0.0975, 0.99, 0.999), df1=9, df2=10)

[1] 2.35 3.02 0.41 4.94 8.96
```

We can carry out the test of equal variances using `var.test()`:

```
> var.test(dec ~ group, data=ds)


F test to compare two variances

data:  dec by group
F = 2.2, num df = 9, denom df = 10, p-value = 0.2365
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.581 8.703
sample estimates:
ratio of variances
               2.2
```

3.2 Power

Power calculations, such as the one described in Example 7.23 (page 478) can be undertaken using the `power.t.test()` function. Suppose that we wanted to plan a new study to provide convincing evidence at the $\alpha = 0.01$ level, with 45 subjects in each of our two groups. We believe that the true different in means is 5 points, and we assume that the population standard deviation is 7.4 for both groups (this corresponds to an effect size of $5/7.4 = 0.676$).

```
> power.t.test(delta=5, n=45, sd=7.4, alternative="one.sided", sig.level=0.01)


     Two-sample t test power calculation

              n = 45
          delta = 5
             sd = 7.4
      sig.level = 0.01
          power = 0.797
    alternative = one.sided

 NOTE: n is number in *each* group
```

We'd still do the two-sided test (which will have slightly less power):

```
> power.t.test(delta=5, n=45, sd=7.4, sig.level=0.01)


     Two-sample t test power calculation

              n = 45
          delta = 5
             sd = 7.4
      sig.level = 0.01
          power = 0.715
    alternative = two.sided

 NOTE: n is number in *each* group
```

An even more flexible approach would be to simulate data to calculate the power. This can be extended to other settings for which the existing power functions do not handle.

We first need to write a function which samples the two groups under the assumptions of the power calculation.

```
> gendata = function() {
   n = 45
   sd = 7.4
   diff = 5
   y1 = rnorm(n, mean=0, sd=sd)
   y2 = rnorm(n, mean=diff, sd=sd)
   y = c(y1, y2)
   x = c(rep("Control", n), rep("Treatment", n))
   return(data.frame(y, x))
 }
```

We can repeatedly call this function, carry out our t-test, and save the p-value (using the `pval()` function in the `mosaic` package. Using 5000 simulations is sufficient to estimate the proportion of times with fair accuracy (if the power was 0.50, then the standard error of the proportion would be $\sqrt{0.5^2/5000} = 0.007$).

Finally, we tally how many of these were less than our desired alpha level (0.01).

```
> powersim = do(5000) * pval(t.test(y ~ x, data=gendata()))
> head(powersim)

   p.value
1 0.015684
2 0.062431
3 0.013455
4 0.000212
5 0.008053
6 0.010064
```

```
> options(digits=5)
> tally(~ p.value <= 0.01, format="percent", data=powersim)


  TRUE  FALSE  Total
 71.96  28.04 100.00
```

The results are quite consistent with the analytic power calculation.

Introduction to the Practice of Statistics using R: Chapter 16

Ben Baumer Nicholas J. Horton*

April 8, 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* (2002) 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`.

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:

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

```
> 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 10: Inference for Regression.

1 Simple Linear Regression

The first example from Chapter 10 is 10.4 (page 566), which assesses fuel economy for 60 cars.

```
> fuel = read.csv("http://math.smith.edu/ips6eR/ch10/eg10_001.csv")
> head(fuel)

  MILES  MPG  MPH LOGMPH  RESID
1 12457 14.8 18.6   2.92 -0.421
2 12658 15.1 19.5   2.97 -0.493
3 13439 17.5 24.2   3.19  0.206
4 13518 14.3 17.9   2.88 -0.619
5 13799 15.9 21.2   3.05 -0.352
6 14097 17.9 32.0   3.47 -1.594
```

In this case we are building a model for MPG as a function of $LOGMPG$, which is a pre-computed variable. Output similar to that shown in Figure 10.5 can be produced by applying the `summary()` command to an `lm` object.

```
> fm1 = lm(MPG ~ LOGMPH, data=fuel)
> summary(fm1)


Call:
lm(formula = MPG ~ LOGMPH, data = fuel)

Residuals:
   Min     1Q Median     3Q    Max
-3.717 -0.519  0.112  0.659  2.149

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -7.796      1.155   -6.75  7.7e-09 ***
LOGMPH         7.874      0.354   22.24  < 2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1 on 58 degrees of freedom
Multiple R-squared: 0.895,Adjusted R-squared: 0.893
F-statistic:  494 on 1 and 58 DF,  p-value: <2e-16
```

Note that R can compute the same model without using the precomputed variables, by applying the `log()` function to the MPH variables on-the-fly.

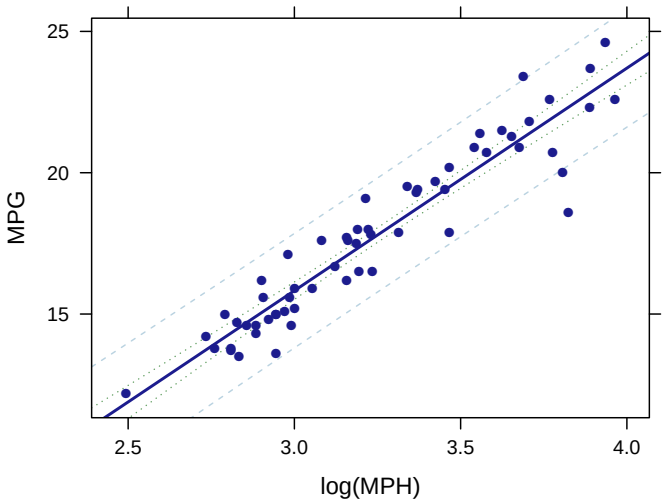
```
> fm1a = lm(MPG ~ log(MPH), data=fuel)
> summary(fm1a)


Call:
lm(formula = MPG ~ log(MPH), data = fuel)

Residuals:
   Min     1Q Median     3Q    Max
-3.717 -0.519  0.112  0.659  2.149

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -7.796      1.155   -6.75  7.7e-09 ***
log(MPH)       7.874      0.354   22.24  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1 on 58 degrees of freedom
Multiple R-squared: 0.895,Adjusted R-squared: 0.893
F-statistic:  494 on 1 and 58 DF,  p-value: <2e-16
```

Like other statistical software packages, R performs a t-test for the null hypothesis that $\beta_i = 0$ for all coefficients β_i present in the model. The third column of the `summary()` output (labeled `t value`) gives the t-statistic, and the fourth column gives the corresponding p-value. Confidence intervals can be retrieved using the `confint()` command, which by default returns a 95% confidence interval.

```
> confint(fm1)

             2.5 % 97.5 %
(Intercept) -10.11  -5.48
LOGMPH        7.17   8.58
```

Confidence intervals for the mean response, as well as prediction intervals for future observations, can be plotted using the `panel.lmbands` argument to `xyplot()`. The following plot is a mashup of Figure 10.9 (page 573) and Figure 10.10 (page 575).

```
> xyplot(MPG ~ log(MPH), panel=panel.lmbands, data=fuel)
```



To retrieve the actual values, we can apply the `predict()` command to our regression model object, and specify whether we want confindence intervals or prediction intervals.

```
> # only show the first six rows for clarity
> head(predict(fm1, interval="confidence"))

   fit  lwr  upr
1 15.2 14.9 15.6
2 15.6 15.3 15.9
3 17.3 17.0 17.6
4 14.9 14.6 15.3
5 16.3 16.0 16.5
6 19.5 19.2 19.8

> # only show the first six rows for clarity
> head(predict(fm1, interval="predict"))

Warning:  Predictions on current data refer to _future_ responses

   fit  lwr  upr
1 15.2 13.2 17.3
2 15.6 13.6 17.6
3 17.3 15.3 19.3
4 14.9 12.9 17.0
5 16.3 14.2 18.3
6 19.5 17.5 21.5
```

2 More Detail about Simple Linear Regression

2.1 The ANOVA F-test

An ANOVA table similar to the one shown in Figure 10.12 (page 583) can be produced by applying the `anova()` command to a regression model object.

```
> anova(fm1)

Analysis of Variance Table

Response: MPG
          Df Sum Sq Mean Sq F value Pr(>F)
LOGMPH     1    494     494     494 <2e-16 ***
Residuals 58     58       1
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2 Inference for Correlation

We can test for zero correlation using the `cor.test()` command. In Example 10.22, a t-test for non-zero correlation is conducted between the MPG and $LOGMPH$ of 60 cars

```
> with(fuel, cor.test(MPG, LOGMPH))


Pearson's product-moment correlation

data:  MPG and LOGMPH
t = 22.2, df = 58, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.911 0.968
sample estimates:
  cor
0.946
```

Introduction to the Practice of Statistics using R: Chapter 11

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`.

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`).

Additional examples of fitting multiple regression models can be found in the companion site which implements the examples within *The Statistical Sleuth* in R (`http://www.math.smith.edu/~nhorton/sleuth`).

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
```

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

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 11: Multiple Regression.

1 Case study: GPA for computer science majors

1.1 Univariate analyses

As always, we begin with a description of the predictor variables and outcome (as displayed on pages 615 and 616).

```
> ds = read.csv("http://www.math.smith.edu/ips6e/appendix/csdata.csv")
> names(ds)

[1] "obs"  "gpa"  "hsm"  "hss"  "hse"  "satm" "satv" "sex"
```

```
> favstats(~ gpa, data=ds)

  min   Q1 median   Q3 max mean    sd   n missing
 0.12 2.17   2.74 3.21   4 2.64 0.779 224       0
```

```
> favstats(~ satm, data=ds)

 min  Q1 median  Q3 max mean   sd   n missing
 300 540    600 650 800  595 86.4 224       0
```

```
> favstats(~ satv, data=ds)

 min  Q1 median  Q3 max mean   sd   n missing
 285 440    490 570 760  505 92.6 224       0
```

```
> favstats(~ hsm, data=ds)

 min Q1 median Q3 max mean   sd   n missing
   2  7      9 10  10 8.32 1.64 224       0
```

```
> favstats(~ hss, data=ds)

 min Q1 median Q3 max mean  sd   n missing
   3  7      8 10  10 8.09 1.7 224       0
```

```
> favstats(~ hse, data=ds)

 min Q1 median Q3 max mean   sd   n missing
   3  7      8  9  10 8.09 1.51 224       0
```

```
> tally(~ sex, data=ds)


   1    2 Total
 145   79   224
```

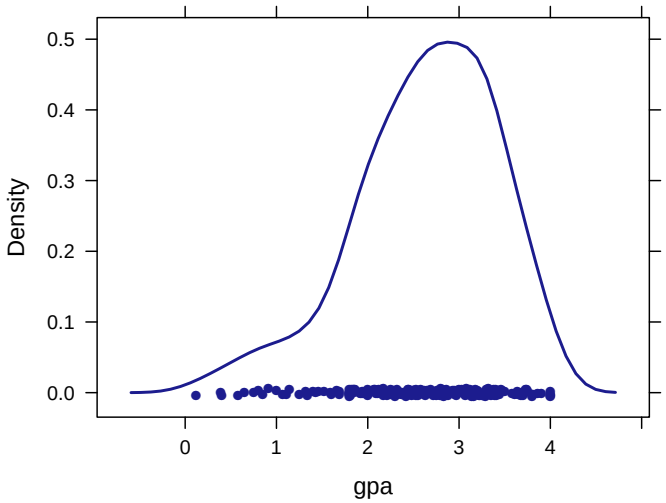
```
> tally(~ sex, format="percent", data=ds)


    1     2 Total
 64.7  35.3 100.0
```

```
> tally(~ hsm, data=ds)


   2    3    4    5    6    7    8    9   10 Total
   1    1    4    6   23   28   36   59   66   224
```

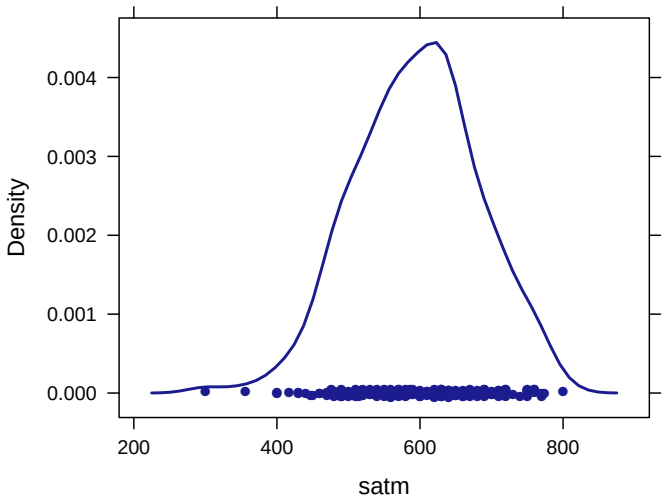
```
> tally(~ hss, data=ds)


   3    4    5    6    7    8    9   10 Total
   1    7    9   24   42   31   50   60   224
```

```
> tally(~ hse, data=ds)


  3   4   5   6   7   8   9  10 Total
  1   4   5  23  43  49  52  47   224
```

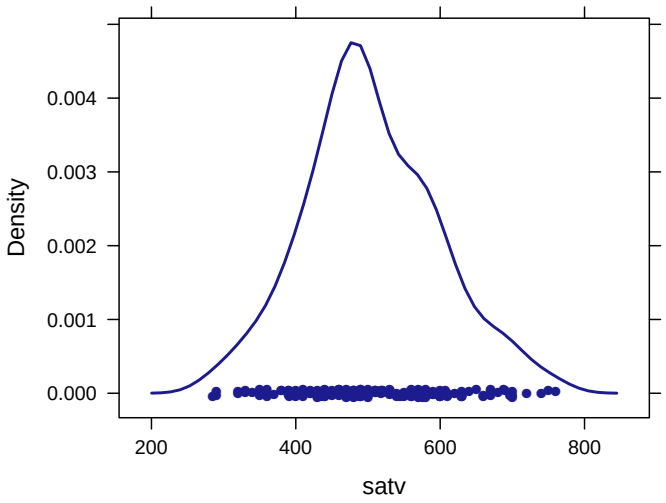
```
> densityplot(~ gpa, data=ds)
```




```
> densityplot(~ satm, data=ds)
```



```
> densityplot(~ satv, data=ds)
```



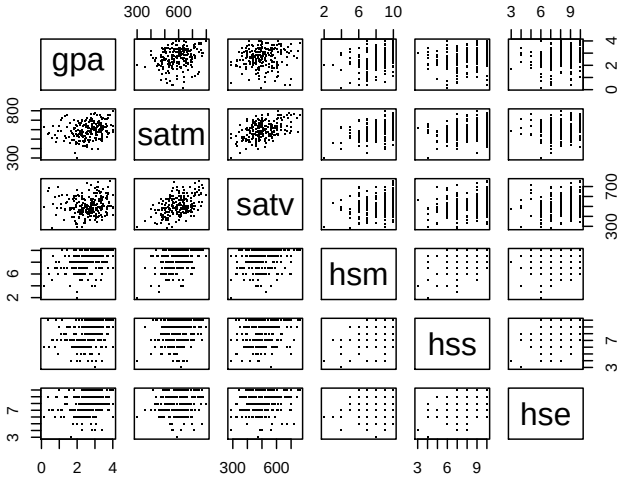
1.2 Bivariate comparisons

We can replicate the correlation matrix in Figure 11.3 (page 617).

```
> smallds = subset(ds, select=c("gpa", "satm", "satv", "hsm", "hss", "hse"))
> with(ds, cor(smallds))

       gpa  satm  satv   hsm   hss   hse
gpa  1.000 0.252 0.114 0.436 0.329 0.289
satm 0.252 1.000 0.464 0.454 0.240 0.108
satv 0.114 0.464 1.000 0.221 0.262 0.244
hsm  0.436 0.454 0.221 1.000 0.576 0.447
hss  0.329 0.240 0.262 0.576 1.000 0.579
hse  0.289 0.108 0.244 0.447 0.579 1.000
```

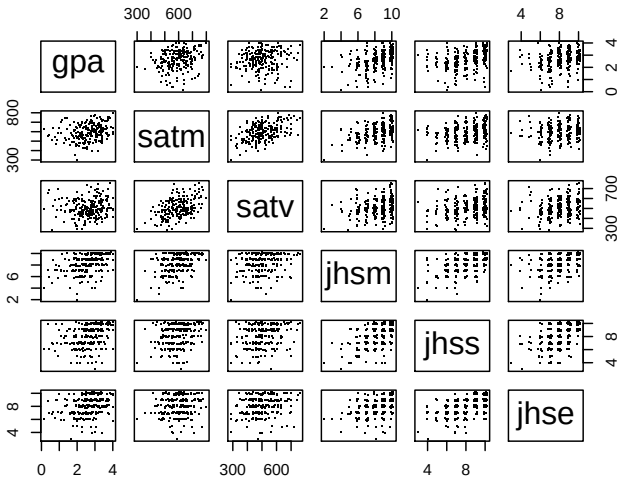
A graphical display may also be helpful:

```
> pairs(smallds, pch=".")
```



Note: jittering the categorical high school scores may improve the readability:

```
> ds = transform(ds, jhsm = jitter(hsm))
> ds = transform(ds, jhss = jitter(hss))
> ds = transform(ds, jhse = jitter(hse))
> smallds = subset(ds, select=c("gpa", "satm", "satv", "jhsm", "jhss", "jhse"))
> pairs(smallds, pch=".")
```



1.3 Multiple regression model

The output in Figure 11.4 (page 618) can be reproduced after fitting the model, which will be saved in the object called `lm1`.

```
> lm1 = lm(gpa ~ hsm + hss + hse, data=ds)
> coef(lm1)

(Intercept)         hsm         hss         hse
     0.5899      0.1686      0.0343      0.0451

> r.squared(lm1)

[1] 0.205

> summary(lm1)


Call:
lm(formula = gpa ~ hsm + hss + hse, data = ds)

Residuals:
    Min      1Q  Median      3Q     Max
-2.1289 -0.3407  0.0757  0.4744  1.7537

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.5899     0.2942    2.00    0.046 *
hsm           0.1686     0.0355    4.75  3.7e-06 ***
hss           0.0343     0.0376    0.91    0.362
hse           0.0451     0.0387    1.17    0.245
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7 on 220 degrees of freedom
Multiple R-squared: 0.205,Adjusted R-squared: 0.194
F-statistic: 18.9 on 3 and 220 DF,  p-value: 6.36e-11

> anova(lm1)

Analysis of Variance Table

Response: gpa
           Df Sum Sq Mean Sq F value  Pr(>F)
hsm         1   25.8   25.81   52.70 6.6e-12 ***
hss         1    1.2    1.24    2.53    0.11
hse         1    0.7    0.67    1.36    0.25
Residuals 220  107.8    0.49
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Predicted values can be calculated from this model.

```
> lm1fun = makeFun(lm1)
> lm1fun(hsm=9, hss=8, hse=7)

  1
2.7

> lm1fun(hsm=9, hss=8, hse=7:9)

   1    2    3
2.70 2.74 2.79
```

In Figure 11.6 (page 621), the HSS predictor is dropped from the model.

```
> lm2 = lm(gpa ~ hsm + hse, data=ds)
> summary(lm2)


Call:
lm(formula = gpa ~ hsm + hse, data = ds)

Residuals:
    Min      1Q  Median      3Q     Max
-2.0588 -0.3883  0.0695  0.4687  1.7332

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.6242     0.2917    2.14    0.033 *
hsm           0.1827     0.0320    5.72  3.5e-08 ***
hse           0.0607     0.0347    1.75    0.082 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7 on 221 degrees of freedom
Multiple R-squared: 0.202,Adjusted R-squared: 0.194
F-statistic: 27.9 on 2 and 221 DF,  p-value: 1.58e-11

> anova(lm2)

Analysis of Variance Table

Response: gpa
           Df Sum Sq Mean Sq F value  Pr(>F)
hsm         1   25.8   25.81   52.74 6.4e-12 ***
hse         1    1.5    1.49    3.05   0.082 .
```

```
Residuals 221  108.2    0.49
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In Figure 11.7 (page 622), the model is fit using SAT scores as explanatory variables.

```
> lm3 = lm(gpa ~ satm + satv, data=ds)
> summary(lm3)


Call:
lm(formula = gpa ~ satm + satv, data = ds)

Residuals:
    Min      1Q  Median      3Q     Max
-2.5948 -0.3792  0.0826  0.5573  1.3993

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.29e+00   3.76e-01    3.43  0.00073 ***
satm         2.28e-03   6.63e-04    3.44  0.00069 ***
satv        -2.46e-05   6.19e-04   -0.04  0.96836
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.758 on 221 degrees of freedom
Multiple R-squared: 0.0634,Adjusted R-squared: 0.0549
F-statistic: 7.48 on 2 and 221 DF,  p-value: 0.000722

> anova(lm3)

Analysis of Variance Table

Response: gpa
           Df Sum Sq Mean Sq F value  Pr(>F)
satm        1    8.6    8.58    14.9 0.00015 ***
satv        1    0.0    0.00     0.0 0.96836
Residuals 221  126.9    0.57
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Finally, in Figure 11.8 (page 624), all possible explanatory variables are used.

```
> lm.full = lm(gpa ~ satm + satv + hsm + hss + hse, data=ds)
> summary(lm.full)
```

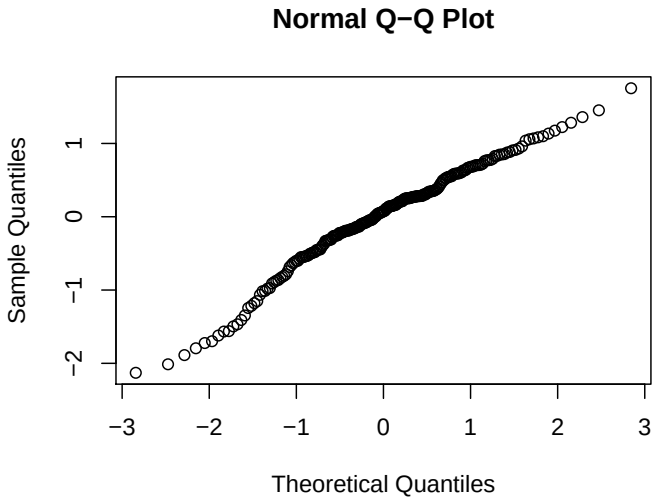
```
Call:
lm(formula = gpa ~ satm + satv + hsm + hss + hse, data = ds)

Residuals:
    Min      1Q  Median      3Q     Max
-2.0649 -0.3084  0.0689  0.4876  1.7054

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.326719   0.399996    0.82  0.41493
satm         0.000944   0.000686    1.38  0.17018
satv        -0.000408   0.000592   -0.69  0.49152
hsm          0.145961   0.039261    3.72  0.00026 ***
hss          0.035905   0.037798    0.95  0.34321
hse          0.055293   0.039569    1.40  0.16372
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7 on 218 degrees of freedom
Multiple R-squared: 0.211,Adjusted R-squared: 0.193
F-statistic: 11.7 on 5 and 218 DF,  p-value: 5.06e-10

> anova(lm.full)

Analysis of Variance Table

Response: gpa
           Df Sum Sq Mean Sq F value  Pr(>F)
satm        1    8.6    8.58   17.52 4.1e-05 ***
satv        1    0.0    0.00    0.00   0.966
hsm         1   17.7   17.73   36.18 7.5e-09 ***
hss         1    1.4    1.38    2.81   0.095 .
hse         1    1.0    0.96    1.95   0.164
Residuals 218  106.8    0.49
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

1.4 Regression diagnostics

As always, we want to assess the fit of the model, and the assumptions needed for it.

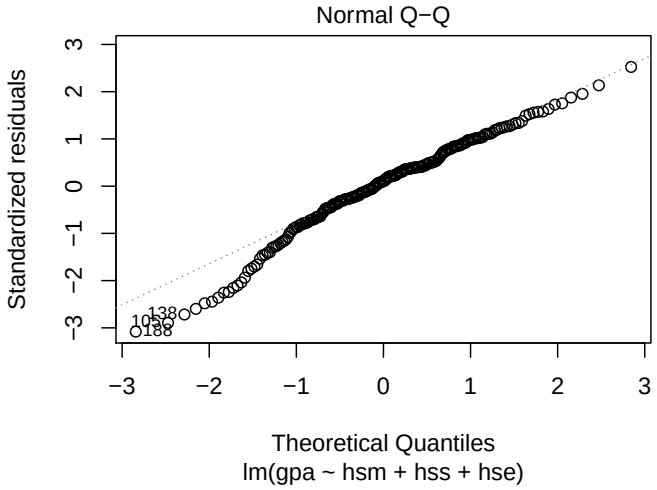
We begin by considering the distribution of the residuals. Figure 11.5 (page 620) displays the normal quantile plot, which can be generated using the `qqnorm()` function.

```
> qqnorm(residuals(lm1))
```



This can also be generated using a built-in plot option:

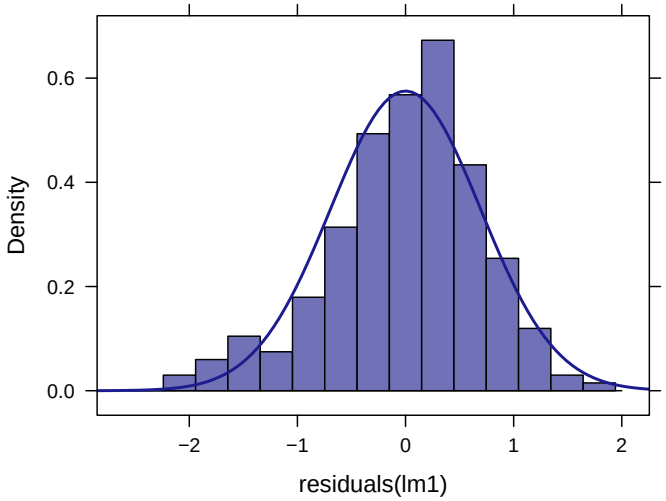
```
> plot(lm1, which=2)
```



Both displays indicate that the distribution of the residuals is approximately normal (with some evidence for a slightly heavy left tail).

We could also generate a histogram with overlaid normal density (mean 0 and standard deviation equal to the root MSE from the model).

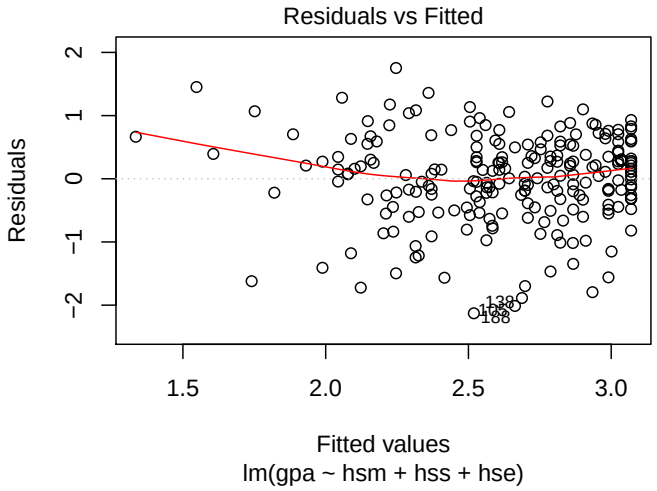
```
> xhistogram(~ residuals(lm1), fit="normal")

Loading required package:  MASS
```



Next we want to consider the distribution of the residuals as a function of the fitted (predicted) values, as we don't want to see a systematic pattern in the relationship between these quantities.

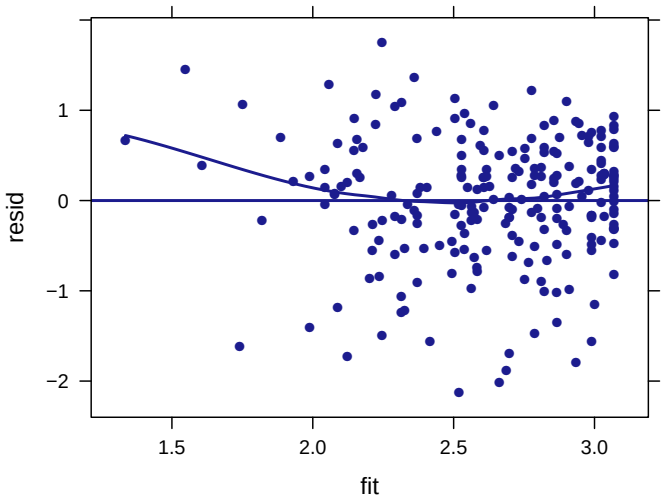
As is often the case, there are multiple ways to generate these plots within R.

```
> plot(lm1, which=1)
```



This can also be created in other ways:

```
> ds = transform(ds, fit=fitted(lm1))
> ds = transform(ds, resid=residuals(lm1))
> xyplot(resid ~ fit, type=c("p", "r", "smooth"), data=ds)
```



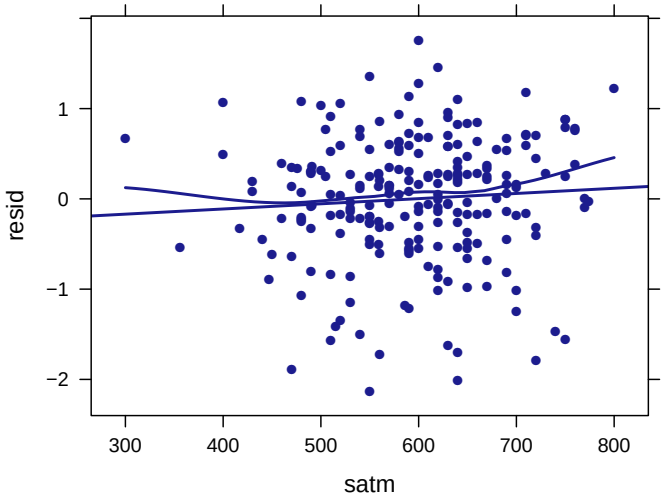
Here the `"r"` option for `type` specifies a regression (which will be a straight line), and the `"smooth"` option adds a lowess (smooth line). There is some indication of non-linearity, particularly in the tails (but that's exactly where the lowess isn't to be trusted, as there's little data).

Subject 188 may bear some additional scrutiny (as it has a very large negative residual):

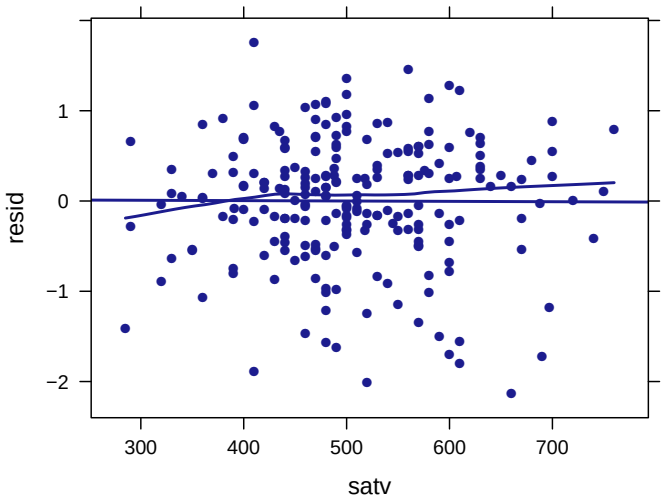
```
> ds[188,]

    obs  gpa hsm hss hse satm satv sex jhsm jhss jhse  fit resid
188 188 0.39   7  10   9  550  660   2 7.16 9.97 8.85 2.52 -2.13
```

We also want to display the residuals against each of the continuous predictors in the model.

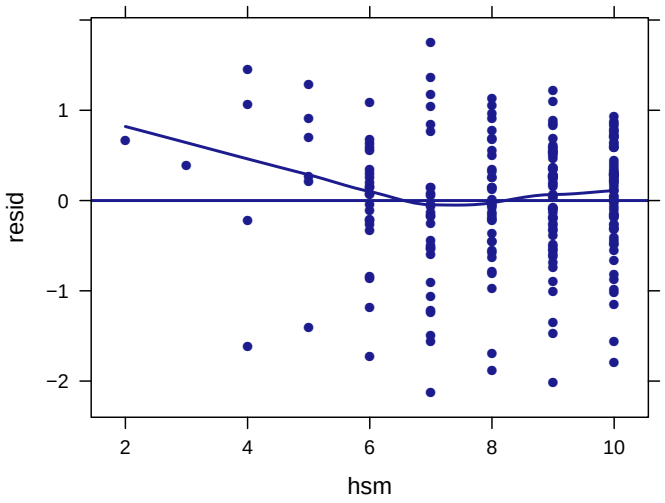
```
> xyplot(resid ~ satm, type=c("p", "r", "smooth"), data=ds)
```



```
> xyplot(resid ~ satv, type=c("p", "r", "smooth"), data=ds)
```

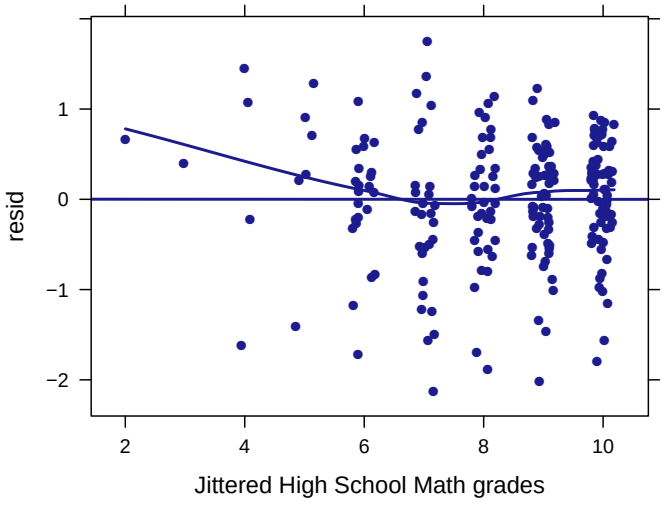



```
> xyplot(resid ~ hsm, type=c("p", "r", "smooth"), data=ds)

Warning:  pseudoinverse used at 9
Warning:  neighborhood radius 1
Warning:  reciprocal condition number  0
Warning:  pseudoinverse used at 9
Warning:  neighborhood radius 1
Warning:  reciprocal condition number  0
Warning:  pseudoinverse used at 9
Warning:  neighborhood radius 1
Warning:  reciprocal condition number  0
Warning:  pseudoinverse used at 9
Warning:  neighborhood radius 1
Warning:  reciprocal condition number  0
Warning:  pseudoinverse used at 9
Warning:  neighborhood radius 1
Warning:  reciprocal condition number  0
```

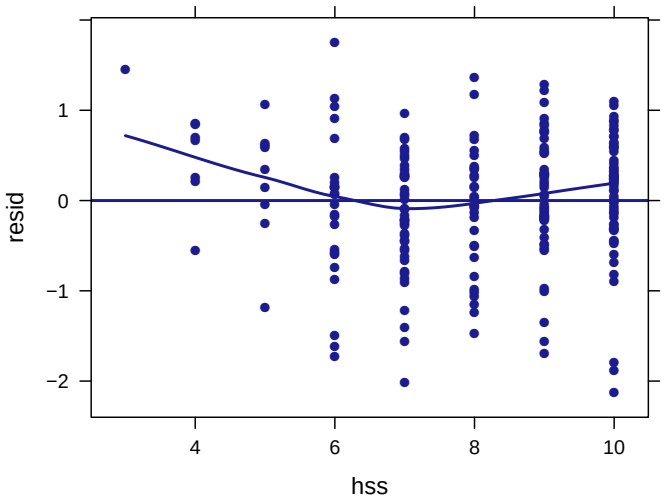


The warnings are due to the discrete nature of the high school math variable, which takes on relatively few values. Jittering will help here:

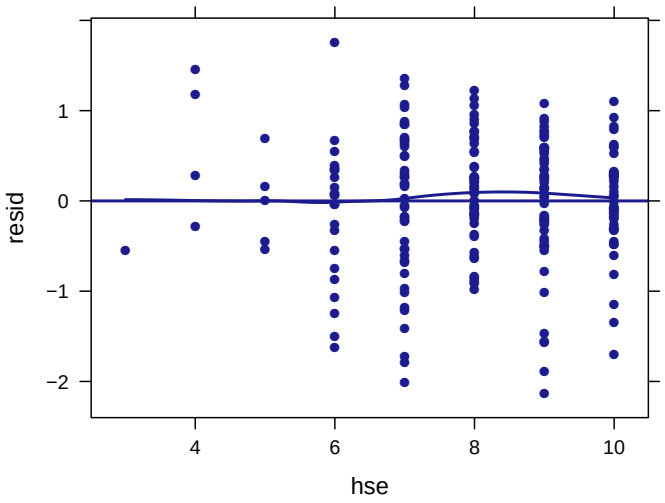
```
> xyplot(resid ~ jhsm, type=c("p", "r", "smooth"),
   xlab="Jittered High School Math grades", data=ds)
```




```
> xyplot(resid ~ hss, type=c("p", "r", "smooth"), data=ds)
```




```
> xyplot(resid ~ hse, type=c("p", "r", "smooth"), data=ds)
```



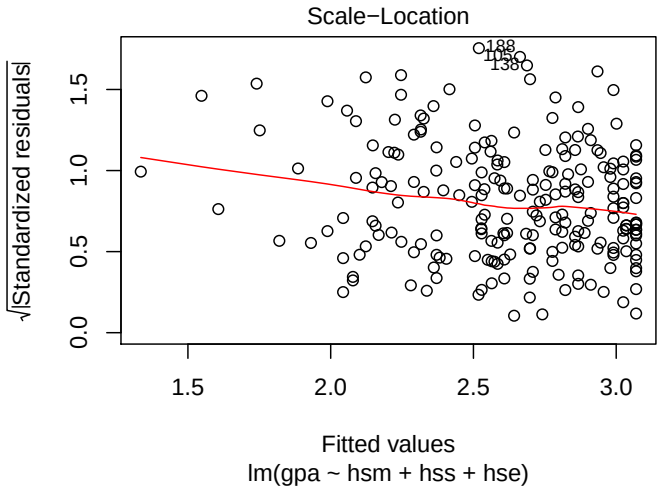
Overall, we see reasonable linearity in the relationships, though for some subjects with low high school math or science grades, the regression model tends to systematically underpredict.

```
> subset(ds, hsm < 4 | hss < 4)

    obs gpa hsm hss hse satm satv sex jhsm jhss jhse  fit resid
8     8   2   3   7   6  460  530   1 2.98 7.03 6.03 1.61 0.394
84   84   3   4   3   4  620  560   1 4.00 2.97 4.09 1.55 1.453
183 183   2   2   4   6  300  290   2 2.00 3.97 6.02 1.33 0.665
```

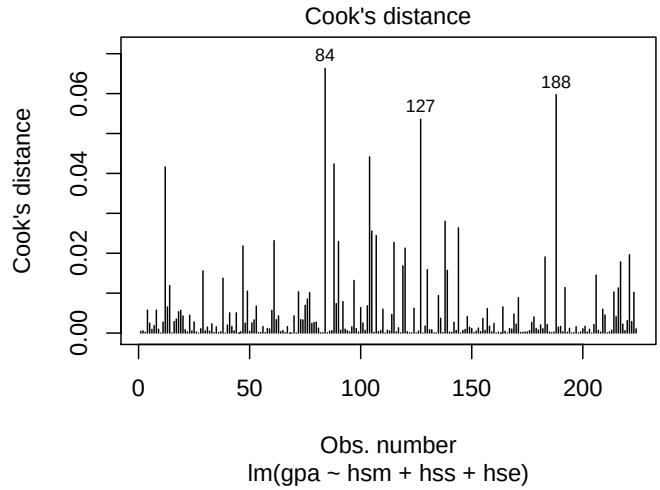
1.5 More advanced residual analysis and regression diagnostics

Additional built-in residual plots can be requested (but we won't be using these much: check out *The Statistical Sleuth* for more information).

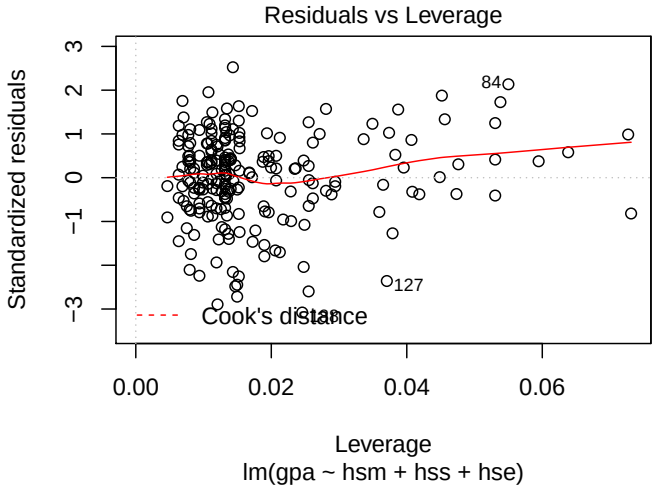
```
> plot(lm1, which=3)
```



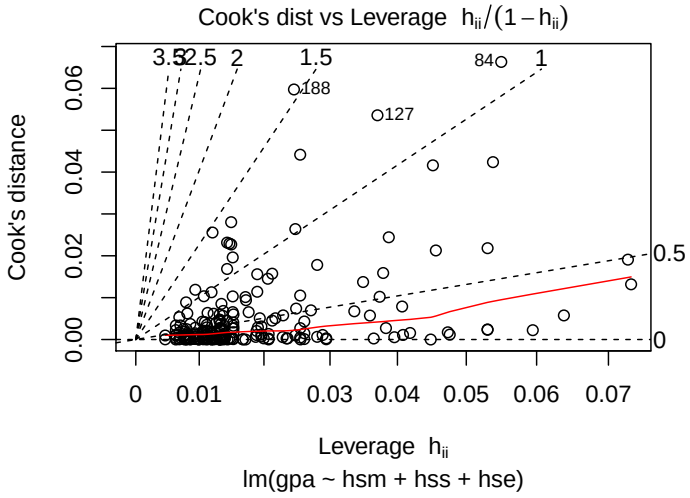

```
> plot(lm1, which=4)
```




```
> plot(lm1, which=5)
```




```
> plot(lm1, which=6)
```



Introduction to the Practice of Statistics using R: Chapter 12

Nicholas J. Horton* Ben Baumer

April 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* (2002) 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`.

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`).

Additional examples of fitting multiple regression models can be found in the companion site which implements the examples within *The Statistical Sleuth* in R (`http://www.math.smith.edu/~nhorton/sleuth`).

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

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

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

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 12: One-Way Analysis of Variance.

1 Inference for One-way ANOVA

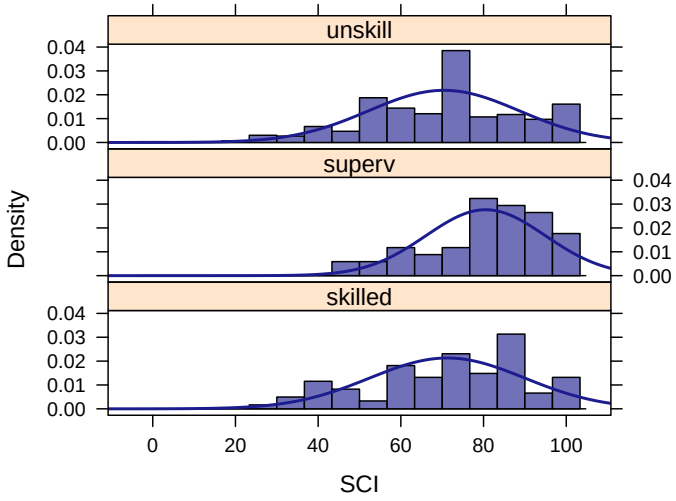
1.1 Exploratory analysis

We consider the case study on workplace safety introduced on page 641 (Example 12.3).

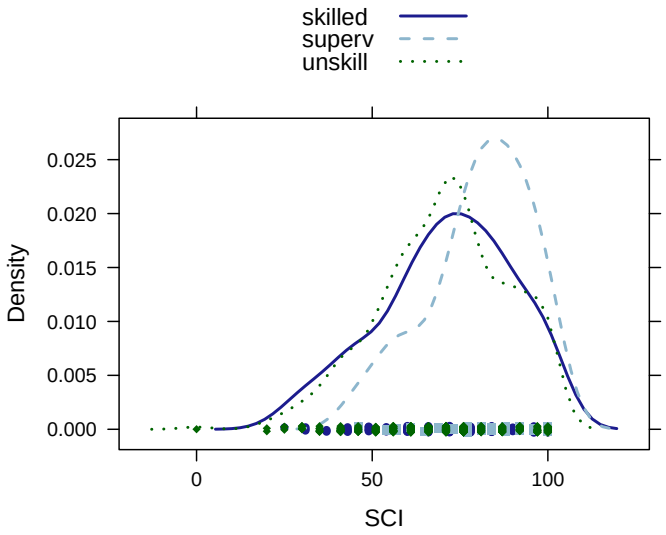
```
> ds = read.csv("http://www.math.smith.edu/ips6e/Ch12/ex12_003.csv")
> favstats(SCI ~ jobcat, data=ds)

         min Q1 median   Q3 max mean   sd   n missing
skilled   25 60     72 84.0 100 71.2 18.8  91       0
superv    46 73     81 92.0 100 80.5 14.6  51       0
unskill    0 61     71 82.5 100 70.4 18.3 448       0
```

Variants of the graphical displays (from Figure 12.3, page 642) are reproduced below. Note that the histograms (with overlaid normal curve) can be generated using separate stacked figures, or a single display can be created using overlapping density plots.

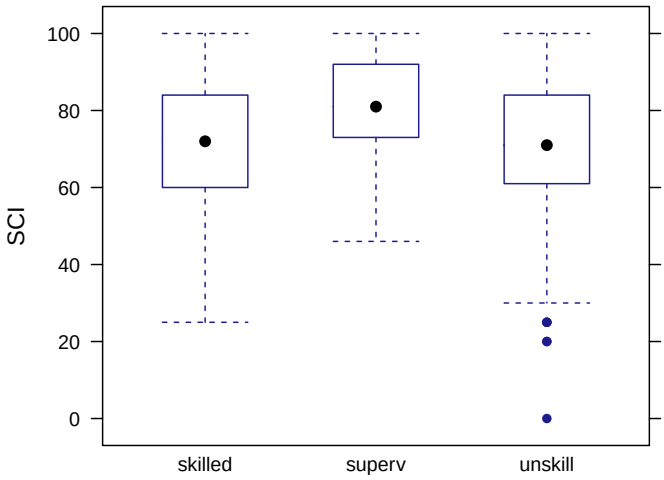
```
> xhistogram(~ SCI| jobcat, fit="normal", layout=c(1, 3), data=ds)

Loading required package:  MASS
```




```
> densityplot(~ SCI, groups=jobcat, auto.key=TRUE, data=ds)
```




```
> bwplot(SCI ~ jobcat, data=ds)
```



1.2 Pooled standard deviation

The pooled standard deviation can be easily calculated through the `lm()` command:

```
> ex12.5 = lm(SCI ~ jobcat, data=ds)
> summary(ex12.5)


Call:
lm(formula = SCI ~ jobcat, data = ds)

Residuals:
   Min     1Q Median     3Q    Max
-70.42 -11.21   0.58  12.79  29.58

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    71.209      1.895   37.59   <2e-16 ***
jobcatsuperv    9.301      3.161    2.94   0.0034 **
jobcatunskill  -0.785      2.078   -0.38   0.7059
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.1 on 587 degrees of freedom
Multiple R-squared: 0.0237,Adjusted R-squared: 0.0204
F-statistic: 7.14 on 2 and 587 DF,  p-value: 0.000866
```

The value of 18.07 matches the results in Example 12.5 (page 647).

1.3 ANOVA table

The ANOVA table (Figure 12.8, page 649) can be generated from this linear model object.

```
> anova(ex12.5)

Analysis of Variance Table

Response: SCI
           Df Sum Sq Mean Sq F value  Pr(>F)
jobcat      2   4662    2331    7.14 0.00087 ***
Residuals 587 191729     327
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

1.4 Decomposition

As always, the total variability (SST) can be decomposed into part explained by the model (SSM or SSG, as described in Example 12.9, page 651) and part unexplained (SSE, or sums of squares for error).

```
> meanval = mean(~ SCI, data=ds)
> SST = with(ds, sum((SCI - meanval)^2))
> SST

[1] 196391

> SSM = sum((fitted(ex12.5) - meanval)^2)
> SSM

[1] 4662

> SSE = sum((residuals(ex12.5)^2))
> SSE

[1] 191729

> SSM + SSE

[1] 196391
```

We can use these results to verify the value of s_p (the pooled estimate of the parameter σ) from our model.

```
> MSE = SSE / (nrow(ds) - 2 - 1); MSE

[1] 327
```

```
> sqrt(MSE)

[1] 18.1
```

This matches the value on page 652 (Example 12.11).

1.5 The F test

The F distribution is used to test the overall hypotheses (and other multiple degree of freedom tests). The p-value is the probability that a random variable having the $F(I-1, N-I)$ distribution is greater or equal to the calculated value of the F statistic. The values from Example 12.12 (page 653) can be found using the `qf()` function:

```
> qf(c(.90, .95, .975, .99, .999), df1 = 2, df2 = 587)

[1] 2.31 3.01 3.71 4.64 6.99
```

(Note that the values in the book are only available for denominator degrees of freedom equal to 200, so the results are conservative).

1.6 Coefficient of determination

As usual, the R^2 (or coefficient of determination) can be calculated in multiple ways:

```
> r.squared(ex12.5)

[1] 0.0237

> SSM/SST

[1] 0.0237
```

2 Comparing the means

2.1 Contrasts

Contrasts can be used to calculate specific one degree of freedom tests of hypotheses. Recall the means from the worker data:

```
> mean(SCI ~ jobcat, data=ds)

skilled  superv unskill
   71.2    80.5    70.4
```

We can also calculate these in terms of the regression parameter estimates:

```
> mycoef = coef(ex12.5); mycoef

  (Intercept)  jobcatsuperv jobcatunskill
       71.209         9.301        -0.785

> mycoef[1]

(Intercept)
       71.2

> mycoef[1] + mycoef[2]

(Intercept)
       80.5

> mycoef[1] + mycoef[3]

(Intercept)
       70.4
```

Contrasts can be fit using the `fit.contrast()` function within the `gmodels` package.

```
> require(gmodels)

Loading required package: gmodels

> fit.contrast(ex12.5, "jobcat", c(-1/2, 1, -1/2))

                         Estimate Std. Error t value Pr(>|t|)
jobcat c=( -0.5 1 -0.5 )     9.69       2.74    3.54 0.000427
```

This matches the results for the first contrast (Example 12.18, pages 658–659).
A similar process is used to test the second contrast (Example 12.20, page 659):

```
> fit.contrast(ex12.5, "jobcat", c(-1, 0, 1))

                     Estimate Std. Error t value Pr(>|t|)
jobcat c=( -1 0 1 )   -0.785       2.08  -0.378    0.706
```

These values can be used to calculate a 95% confidence interval for the difference in means:

```
> -0.79 + c(-1, 1)*qt(.975, df=587) * 2.08

[1] -4.88  3.30
```

2.2 Multiple comparisons

A number of packages support the comparison of multiple tests using R (see for example the `multcomp` package).

2.3 Power

The `power.anova.test()` function can be used to calculate power and sample size for a one-way ANOVA. For the power of a reading comprehension study (Example 12.27, pages 668-669), this yields power of approximately 35%.

```
> power.anova.test(groups=3, n=10, within.var=7^2, between.var=var(c(41, 47, 44)))


     Balanced one-way analysis of variance power calculation

         groups = 3
              n = 10
    between.var = 9
     within.var = 49
      sig.level = 0.05
          power = 0.349

 NOTE: n is number in each group
```

Introduction to the Practice of Statistics using R: Chapter 16

Ben Baumer Nicholas J. Horton*

March 29, 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* (2002) 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`.

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:

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

```
> 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)
```

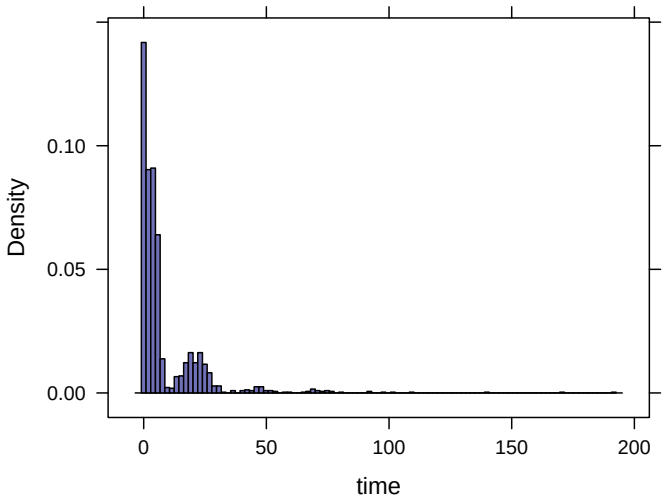
The specific goal of this document is to demonstrate how to replicate the analysis described in Chapter 16: Bootstrap Methods and Permutation Tests.

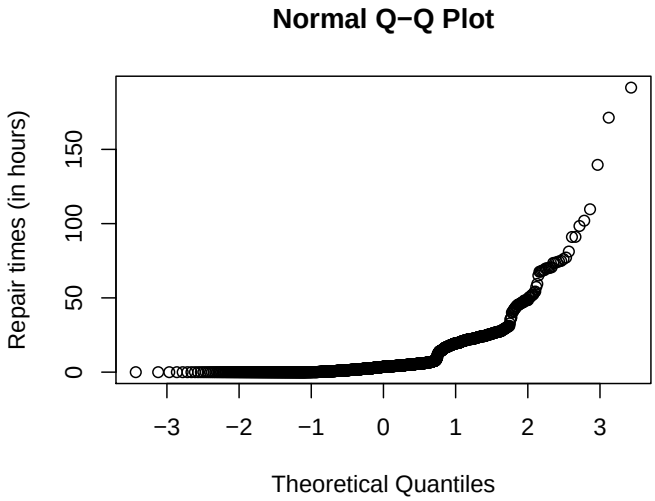
1 The Bootstrap Idea

The bootstrap is a fundamental concept in statistical computing, and the requisite calculations are very easy to perform in R.

The repair time data from Verizon shown in Figure 16.1 (page 16-4) can be plotted thusly:

```
> verizon = read.csv("http://www.math.smith.edu/ips6eR/ch16/eg16_001.csv")
> xhistogram(~time, data=verizon, nint=100)
> with(verizon, qqnorm(time, ylab="Repair times (in hours)"))
```





A command to facilitate resampling within the `mosaic` package is `resample()`. We get our first example on page 16-5, which considers a subset of size $n = 6$ from the Verizon dataset.

```
> data = c(3.12, 0, 1.57, 19.67, 0.22, 2.2)
> mean(data)

[1] 4.46

> s1 = resample(data)
> s1

[1] 0.00 0.22 1.57 2.20 2.20 3.12

> mean(s1)

[1] 1.55

> s2 = resample(data)
> s2

[1] 19.67  2.20 19.67  1.57  2.20  1.57

> mean(s2)

[1] 7.81

> s3 = resample(data)
> s3

[1]  0.22 19.67  3.12  2.20  0.00  3.12

> mean(s3)

[1] 4.72
```

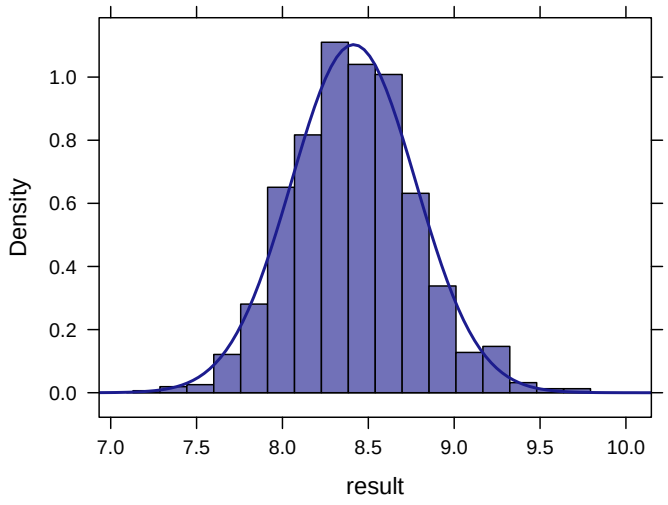
Note that the results shown here do not match the book, due to the random nature of resampling.

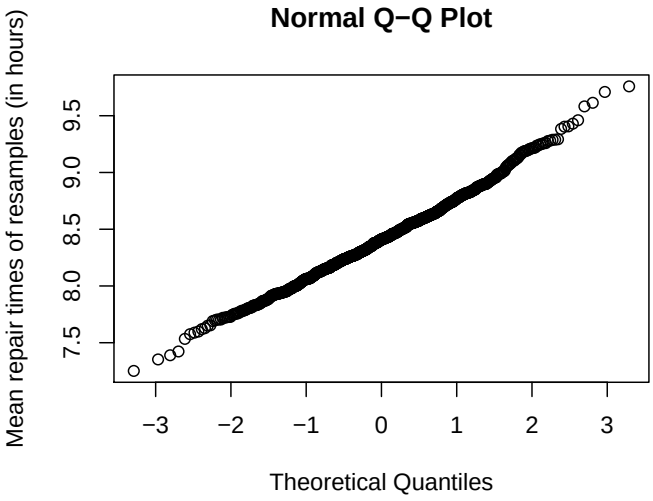
In Figure 16.3 (page 16-6) we visualize a bootstrap distribution. To construct such a thing, we use the `do()` command, which simply repeats some operation many times, and collects the results in a data frame.

```
> mean(~time, data=verizon)

[1] 8.41

> mean(~time, data=resample(verizon))

[1] 8.26

> mean(~time, data=resample(verizon))

[1] 8.94

> mean(~time, data=resample(verizon))

[1] 8.66

> bootstrap = do(1000) * mean(time, data=resample(verizon))
> favstats(~result, data=bootstrap)

  min   Q1 median   Q3  max mean    sd    n missing
 7.25 8.17   8.41 8.63 9.76 8.41 0.362 1000       0

> # Theoretical standard error
> 14.69 / sqrt(1664)

[1] 0.36
```

Note how the theoretical standard error (i.e. standard deviation of the sampling distribution of the mean) compares to the standard deviation from the bootstrap sample.

```
> xhistogram(~result, data=bootstrap, fit="normal")
> with(bootstrap, qqnorm(result, ylab="Mean repair times of resamples (in hours)"))
```

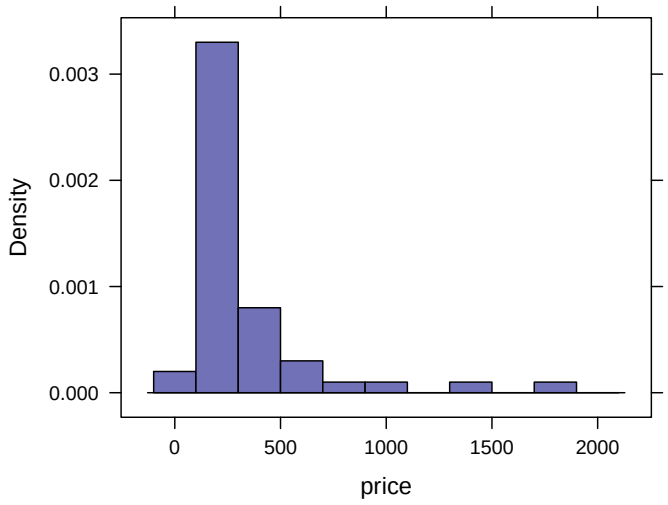


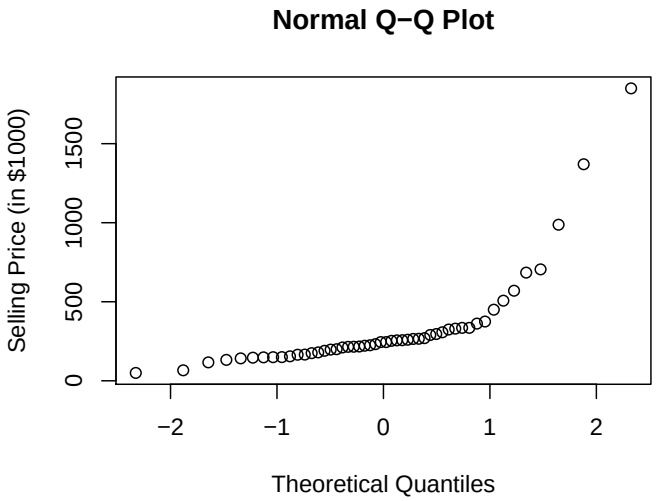


2 First Steps in Using the Bootstrap

Table 16.1 and Figure 16.6 (page 16-14) display residential and commercial real estate prices in Seattle.

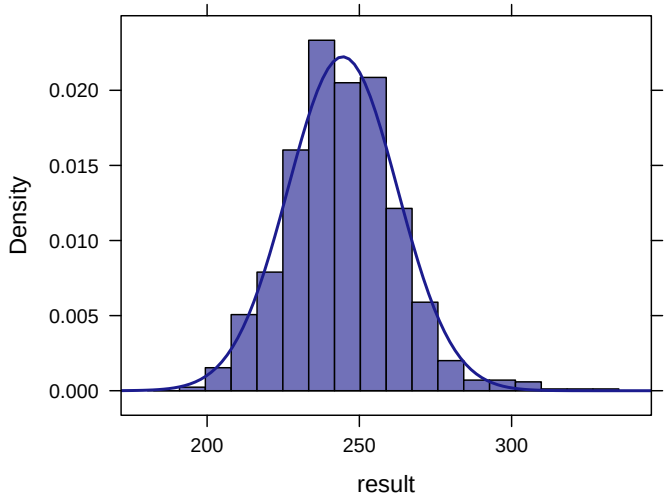
```
> seattle = read.csv("http://www.math.smith.edu/ips6eR/ch16/ta16_001.csv")
> names(seattle) = c("price")
> xhistogram(~price, data=seattle)
> with(seattle, qqnorm(price, ylab="Selling Price (in $1000)"))
```

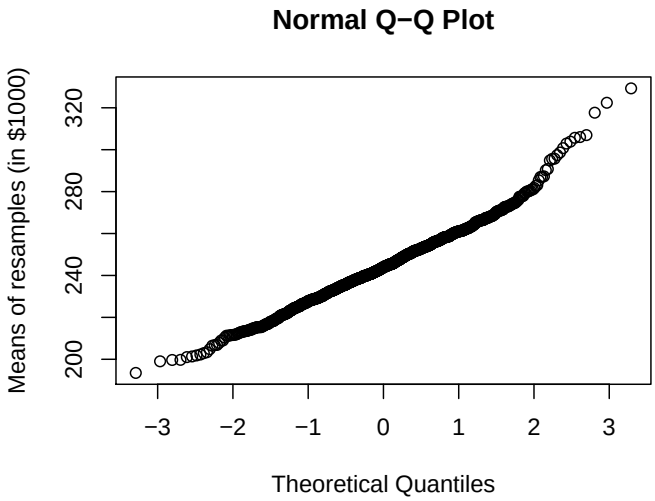




In this example we are working with the 25% trimmed mean. To find the 25% trimmed mean, we grab only the middle 50% of the data, and compute the mean on this subset. This can be achieved using the `trim` argument to `mean()`.

```
> mean(~price, trim=0.25, data=seattle)

[1] 244

> bootstrap = do(1000) * mean(~price, trim=0.25, data=resample(seattle))
> favstats(~result, data=bootstrap)

 min  Q1 median  Q3 max mean sd    n missing
 194 233    244 256 329  245 18 1000       0

> xhistogram(~result, data=bootstrap, fit="normal")
> with(bootstrap, qqnorm(result, ylab="Means of resamples (in $1000)"))
```





We compute the bias as the difference between the average of the bootstrapped means and the trimmed mean from the original sample.

```
> # bias
> mean(~result, data=bootstrap) - mean(~price, trim=0.25, data=seattle)

[1] 0.539
```

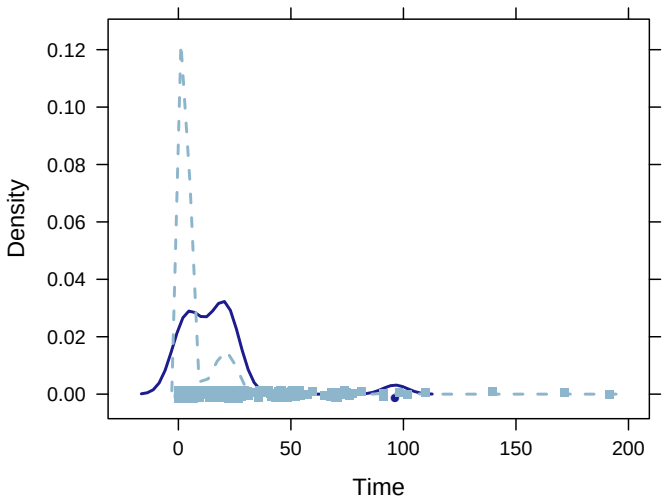
The computation of the confidence interval in Example 16.5 (page 16-16) makes use of the t-distribution.

```
> se.boot = sd(~result, bootstrap)
> t.star = qt(0.975, df=49)
> t.star
```

```
[1] 2.01

> moe = t.star * se.boot
> mean(~price, trim=0.25, data=seattle) + c(-moe, moe)

[1] 208 280
```

In Example 16.6, we compare the means of two groups of service providers.

```
> CLEC = read.csv("http://www.math.smith.edu/ips6eR/ch16/eg16_006.csv")
> mean(Time ~ Group, data=CLEC)

 CLEC  ILEC
16.51  8.41

> densityplot(~Time, groups=Group, data=CLEC)
```



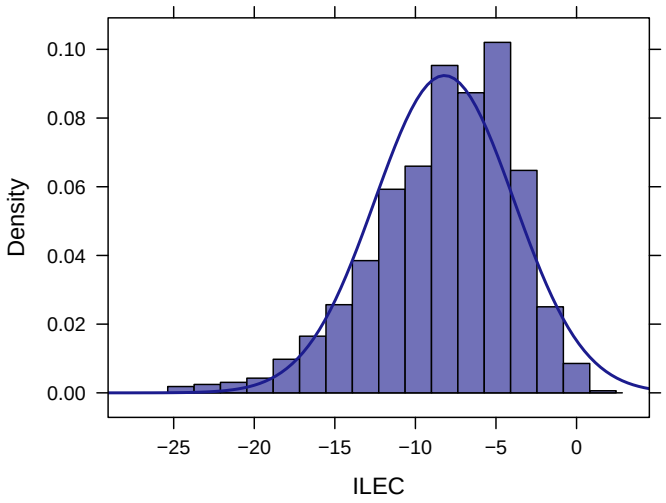
We then construct a bootstrap distribution for the difference in means among the two groups.

```
> bstrap = do(1000) * diff(mean(Time ~ Group, data=resample(CLEC)))
> favstats(~ILEC, data=bstrap)

   min    Q1 median Q3   max mean   sd    n missing
 -25.3 -10.8  -7.64 -5 0.917 -8.2 4.32 1000       0
```

Note that the resulting distribution is not quite so normal. Thus, we can use the quantile method to produce a bootstrap percentile confidence interval for the mean.

```
> xhistogram(~ILEC, fit="normal", data=bstrap)
> qdata(c(0.025, 0.975), vals=ILEC, data=bstrap)
```

```
 2.5% 97.5%
-18.0  -1.4
```



3 How Accurate is a Bootstrap Distribution?

4 Bootstrap Confidence Intervals

We return to the construction of a confidence interval for the mean price of real estate in Seattle explored in Example 16-5. To the t-based confidence interval we constructed previously, we can add the percentile-based confidence interval

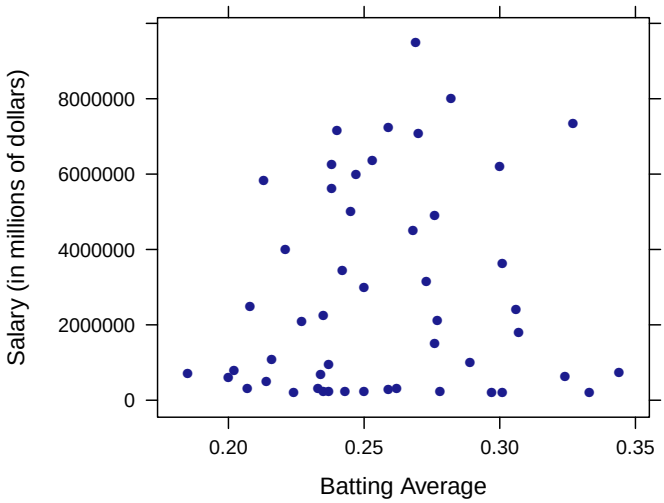
```
> mean(~price, trim=0.25, data=seattle) + c(-moe, moe)

[1] 208 280

> qdata(c(0.025, 0.975), vals=result, data=bootstrap)

 2.5% 97.5%
  212   281
```

Note that the bootstrapped confidence interval is not quite symmetric with respect to the sample mean of 244.

4.1 Confidence intervals for the correlation

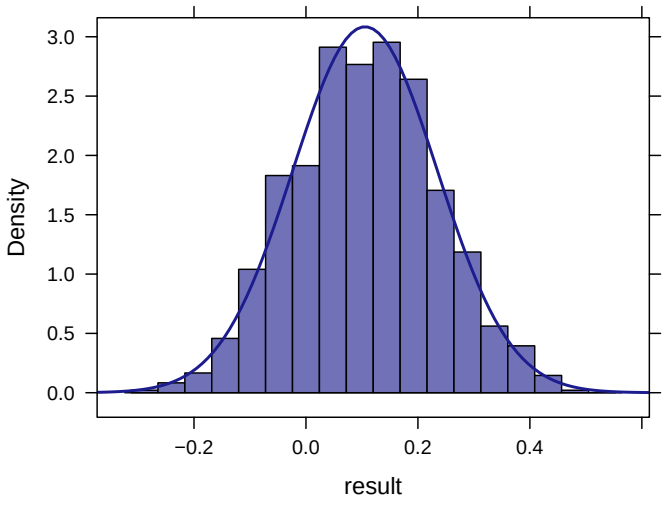
In Example 16.10 (page 16-35), we explore the correlation between batting average and player salary in Major League Baseball. The value of the correlation coefficient among the 50 players in Table 16.2 (page 16-36) is relatively small.

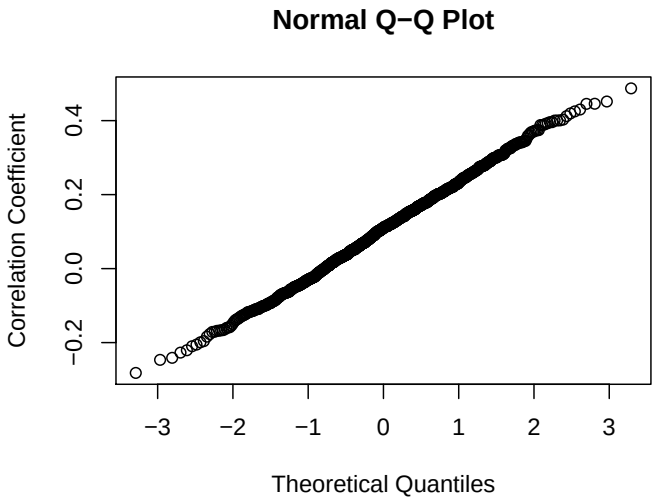
```
> MLB = read.csv("http://www.math.smith.edu/ips6eR/ch16/ta16_002.csv")
> names(MLB)[2] = "Salary"
> xyplot(Salary ~ Average, data=MLB, xlab="Batting Average"
        , ylab="Salary (in millions of dollars)")
> with(MLB, cor(Salary, Average))

[1] 0.107
```



To construct a bootstrap distribution for the correlation between batting average and salary, we resample the players and compute the correlation coefficient.

```
> cor.boot = do(1000) * with(resample(MLB), cor(Salary, Average))
> xhistogram(~result, data=cor.boot, fit="normal")
> with(cor.boot, qqnorm(result, ylab="Correlation Coefficient"))
```





In this case, the t-based confidence interval for the correlation coefficient

```
> se.boot = sd(~result, cor.boot)
> t.star = qt(0.975, df=(nrow(MLB) - 1))
> t.star

[1] 2.01

> moe = t.star * se.boot
> with(MLB, cor(Salary, Average)) + c(-moe, moe)

[1] -0.153  0.367
```

is in reasonable agreement with the percentile-based method.

```
> qdata(c(0.025, 0.975), vals=result, data=cor.boot)

  2.5%  97.5%
-0.137  0.366
```