Lock5 with R: A companion to Unlocking the Power of Data

Randall Pruim and Lana Park

February 26, 2014

©2014

Contents

 ©2014

1 Collecting Data

1.1 The Structure of Data

A Student Survey

Data sets in R are usually stored as **data frames** in a rectangular arrangement with rows corresponding to observational units and columns corresponding to variables. A number of data sets are built into R and its packages. The package for our text is Lock5Data which comes with a number of data sets.

```
require(Lock5Data)  # Tell R to use the package for our text book
data(StudentSurvey)  # load the StudentSurvey data set
```

Let's take a look at the data frame for the Student Survey example in the text. If we type the name of the data set, R will display it in its entirety for us. However, StudentSurvey is a larger data set, so it is more useful to look at some sort of summary or subset of the data.

```
head(StudentSurvey)  # first six cases of the data set

       Year Gender Smoke   Award HigherSAT Exercise TV Height Weight Siblings BirthOrder
1    Senior      M    No Olympic      Math       10  1     71    180        4          4
2 Sophomore      F   Yes Academy      Math        4  7     66    120        2          2
3 FirstYear      M    No   Nobel      Math       14  5     72    208        2          1
4    Junior      M    No   Nobel      Math        3  1     63    110        1          1
5 Sophomore      F    No   Nobel    Verbal        3  3     65    150        1          1
6 Sophomore      F    No   Nobel    Verbal        5  4     65    114        2          2
  VerbalSAT MathSAT  SAT  GPA Pulse Piercings    Sex
1       540     670 1210 3.13    54         0   Male
2       520     630 1150 2.50    66         3 Female
3       550     560 1110 2.55   130         0   Male
4       490     630 1120 3.10    78         0   Male
5       720     450 1170 2.70    40         6 Female
6       600     550 1150 3.20    80         4 Female
```

We can easily classify variables as either **categorical** or **quantitative** by studying the result of head(), but there are some summaries of the data set which reveal such information.

```
str(StudentSurvey)  # structure of the data set
```

```
'data.frame': 362 obs. of  18 variables:
 $ Year      : Factor w/ 5 levels "","FirstYear",..: 4 5 2 3 5 5 2 5 3 2 ...
 $ Gender    : Factor w/ 2 levels "F","M": 2 1 2 2 1 1 1 2 1 1 ...
 $ Smoke     : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 1 1 1 1 1 ...
 $ Award     : Factor w/ 3 levels "Academy","Nobel",..: 3 1 2 2 2 2 3 3 2 2 ...
 $ HigherSAT : Factor w/ 3 levels "","Math","Verbal": 2 2 2 2 3 3 2 2 3 2 ...
 $ Exercise  : num  10 4 14 3 3 5 10 13 3 12 ...
 $ TV        : int  1 7 5 1 3 4 10 8 6 1 ...
 $ Height    : int  71 66 72 63 65 65 66 74 61 60 ...
 $ Weight    : int  180 120 208 110 150 114 128 235 NA 115 ...
 $ Siblings  : int  4 2 2 1 1 2 1 1 2 7 ...
 $ BirthOrder: int  4 2 1 1 1 2 1 1 2 8 ...
 $ VerbalSAT : int  540 520 550 490 720 600 640 660 550 670 ...
 $ MathSAT   : int  670 630 560 630 450 550 680 710 550 700 ...
 $ SAT       : int  1210 1150 1110 1120 1170 1150 1320 1370 1100 1370 ...
 $ GPA       : num  3.13 2.5 2.55 3.1 2.7 3.2 2.77 3.3 2.8 3.7 ...
 $ Pulse     : int  54 66 130 78 40 80 94 77 60 94 ...
 $ Piercings : int  0 3 0 0 6 4 8 0 7 2 ...
 $ Sex       : Factor w/ 2 levels "Female","Male": 2 1 2 2 1 1 1 2 1 1 ...
```

```
summary(StudentSurvey)  # summary of each variable
```

```
        Year     Gender  Smoke         Award      HigherSAT      Exercise
          :  2   F:169   No :319   Academy: 31          :  7   Min.   : 0.00
 FirstYear: 94   M:193   Yes: 43   Nobel  :149   Math   :205   1st Qu.: 5.00
 Junior   : 35                     Olympic:182   Verbal:150    Median : 8.00
 Senior   : 36                                                 Mean   : 9.05
 Sophomore:195                                                 3rd Qu.:12.00
                                                               Max.   :40.00
                                                               NA's   :1
       TV           Height          Weight        Siblings      BirthOrder     VerbalSAT
 Min.   : 0.0   Min.   :59.0   Min.   : 95   Min.   :0.00   Min.   :1.00   Min.   :390
 1st Qu.: 3.0   1st Qu.:65.0   1st Qu.:138   1st Qu.:1.00   1st Qu.:1.00   1st Qu.:550
 Median : 5.0   Median :68.0   Median :155   Median :1.00   Median :2.00   Median :600
 Mean   : 6.5   Mean   :68.4   Mean   :160   Mean   :1.73   Mean   :1.83   Mean   :594
 3rd Qu.: 9.0   3rd Qu.:71.0   3rd Qu.:180   3rd Qu.:2.00   3rd Qu.:2.00   3rd Qu.:640
 Max.   :40.0   Max.   :83.0   Max.   :275   Max.   :8.00   Max.   :8.00   Max.   :800
 NA's   :1      NA's   :7      NA's   :5                    NA's   :3
    MathSAT          SAT            GPA            Pulse          Piercings          Sex
 Min.   :400   Min.   : 800   Min.   :2.00   Min.   : 35.0   Min.   : 0.00   Female:169
 1st Qu.:560   1st Qu.:1130   1st Qu.:2.90   1st Qu.: 62.0   1st Qu.: 0.00   Male  :193
 Median :610   Median :1200   Median :3.20   Median : 70.0   Median : 0.00
 Mean   :609   Mean   :1204   Mean   :3.16   Mean   : 69.6   Mean   : 1.67
 3rd Qu.:650   3rd Qu.:1270   3rd Qu.:3.40   3rd Qu.: 77.8   3rd Qu.: 3.00
 Max.   :800   Max.   :1550   Max.   :4.00   Max.   :130.0   Max.   :10.00
                              NA's   :17                     NA's   :1
```

Here are some more summaries:

```
nrow(StudentSurvey)  # number of rows
```

```
[1] 362
```

```
ncol(StudentSurvey)  # number of columns
```

```
[1] 18

dim(StudentSurvey)  # number of rows and columns

[1] 362  18
```

Data on Countries

```
head(AllCountries)

         Country Code LandArea Population Energy Rural Military Health HIV Internet
1    Afghanistan  AFG   652230     29.021     NA  76.0      4.4    3.7  NA      1.7
2        Albania  ALB    27400      3.143   2088  53.3       NA    8.2  NA     23.9
3        Algeria  ALG  2381740     34.373  37069  34.8     13.0   10.6 0.1     10.2
4 American Samoa  ASA      200      0.066     NA   7.7       NA     NA  NA       NA
5        Andorra  AND      470      0.084     NA  11.1       NA   21.3  NA     70.5
6         Angola  ANG  1246700     18.021  10972  43.3       NA    6.8 2.0      3.1
  Developed BirthRate ElderlyPop LifeExpectancy     CO2    GDP   Cell Electricity
1        NA      46.5        2.2           43.9 0.02503  501.5  37.81          NA
2         1      14.6        9.3           76.6 1.31286 3678.2 141.93      1747.1
3         1      20.8        4.6           72.4 3.23296 4494.9  92.42       971.0
4        NA        NA         NA             NA      NA     NA     NA          NA
5        NA      10.4         NA             NA 6.52783     NA  77.18          NA
6         1      42.9        2.5           47.0 1.35109 4422.5  46.69       202.2
   kwhPerCap
1       <NA>
2 Under 2500
3 Under 2500
4       <NA>
5       <NA>
6 Under 2500

summary(AllCountries)

           Country          Code        LandArea          Population        Energy
 Afghanistan    :  1          :  3   Min.    :       2   Min.   :   0.0   Min.   :    159
 Albania        :  1    AFG   :  1   1st Qu.:   10830   1st Qu.:   0.8   1st Qu.:   5252
 Algeria        :  1    ALB   :  1   Median :   94080   Median :   5.6   Median :  17478
 American Samoa:  1    ALG   :  1   Mean   :  608120   Mean   :  31.5   Mean   :  86312
 Andorra        :  1    AND   :  1   3rd Qu.:  446300   3rd Qu.:  20.6   3rd Qu.:  52486
 Angola         :  1    ANG   :  1   Max.   :16376870   Max.   :1324.7   Max.   :2283722
 (Other)        :207    (Other):205                     NA's   :1        NA's   :77
     Rural          Military         Health           HIV            Internet
 Min.   : 0.0   Min.   : 0.00   Min.   : 0.7   Min.   : 0.10   Min.   : 0.20
 1st Qu.:22.9   1st Qu.: 3.80   1st Qu.: 8.0   1st Qu.: 0.10   1st Qu.: 5.65
 Median :40.4   Median : 5.85   Median :11.3   Median : 0.40   Median :22.80
 Mean   :42.1   Mean   : 8.28   Mean   :11.2   Mean   : 1.98   Mean   :28.96
 3rd Qu.:63.2   3rd Qu.:12.18   3rd Qu.:14.4   3rd Qu.: 1.30   3rd Qu.:48.15
 Max.   :89.6   Max.   :29.30   Max.   :26.1   Max.   :25.90   Max.   :90.50
                NA's   :115     NA's   :26     NA's   :68      NA's   :14
   Developed       BirthRate       ElderlyPop     LifeExpectancy      CO2
 Min.   :1.00   Min.   : 8.2   Min.   : 1.00   Min.   :43.9   Min.   : 0.02
 1st Qu.:1.00   1st Qu.:12.1   1st Qu.: 3.40   1st Qu.:62.8   1st Qu.: 0.62
 Median :1.00   Median :19.4   Median : 5.40   Median :71.9   Median : 2.74
```

```
 Mean   :1.76   Mean   :22.0   Mean   : 7.47   Mean   :68.9   Mean   : 5.09
 3rd Qu.:3.00   3rd Qu.:28.9   3rd Qu.:11.60   3rd Qu.:76.0   3rd Qu.: 7.02
 Max.   :3.00   Max.   :53.5   Max.   :21.40   Max.   :82.8   Max.   :49.05
 NA's   :78     NA's   :16     NA's   :22      NA's   :17     NA's   :15
      GDP              Cell          Electricity         kwhPerCap
 Min.   :   192   Min.   :  1.24   Min.   :   36   Under 2500 :73
 1st Qu.:  1253   1st Qu.: 59.21   1st Qu.:  800   2500 - 5000:21
 Median :  4409   Median : 93.70   Median : 2238   Over 5000  :41
 Mean   : 11298   Mean   : 91.09   Mean   : 4109   NA's       :78
 3rd Qu.: 12431   3rd Qu.:121.16   3rd Qu.: 5824
 Max.   :105438   Max.   :206.43   Max.   :51259
 NA's   :40       NA's   :12       NA's   :78
```

1.2 Sampling from a Population

Imagine data as a 2-dimensional structure (like a spreadsheet).

- Rows correspond to **observational units** (people, animals, plants, or other objects we are collecting data about).
- Columns correspond to **variables** (measurements collected on each observational unit).
- At the intersection of a row and a column is the **value** of the variable for a particular observational unit.

Observational units go by many names, depending on the kind of thing being studied. Popular names include subjects, individuals, and cases. Whatever you call them, it is important that you always understand what your observational units are.

Variable terminology

categorical variable a variable that places observational units into one of two or more categories (examples: color, sex, case/control status, species, etc.)

These can be further sub-divided into ordinal and nominal variables. If the categories have a natural and meaningful order, we will call them **ordered** or **ordinal** variables. Otherwise, they are **nominal** variables.

quantitative variable a variable that records measurements along some scale (examples: weight, height, age, temperature) or counts something (examples: number of siblings, number of colonies of bacteria, etc.)

Quantitative variables can be **continuous** or **discrete**. Continuous variables can (in principle) take on any real-number value in some range. Values of discrete variables are limited to some list and "in-between values" are not possible. Counts are a good example of discrete variables.

response variable a variable we are trying to predict or explain

explanatory variable a variable used to predict or explain a response variable

 ©2014

Estimation

Often we are interested in knowing (approximately) the value of some parameter. A statistic used for this purpose is called an **estimate**. For example, if you want to know the mean length of the tails of lemurs (that's a *parameter*), you might take a sample of lemurs and measure their tails. The mean length of the tails of the lemurs in your sample is a *statistic*. It is also an *estimate*, because we use it to estimate the parameter.

Statistical estimation methods attempt to

- reduce **bias**, and
- increase **precision**.

bias the systematic tendency of sample estimates to either overestimate or underestimate population parameters; that is, a *systematic tendency to be off in a particular direction*.

precision the measure of how close estimates are to the thing being estimated (called the **estimand**).

1.3 Samples and Populations

population the collection of animals, plants, objects, etc. that we want to know about

sample the (smaller) set of animals, plants, objects, etc. about which we have data

parameter a number that describes a population or model.

statistic a number that describes a sample.

Much of statistics centers around this question:

What can we learn about a population from a sample?

Sampling

Sampling is the process of selecting a sample. Statisticians use **random samples**

- to avoid (or at least reduce) **bias**, and
- so they can quantify **sampling variability** (the amount samples differ from each other), which in turn allows us to quantify precision.

The simplest kind of random sample is called a **simple random sample** (aren't statisticians clever about naming things?). A simple random sample is equivalent to putting all individuals in the population into a big hat, mixing thoroughly, and selecting some out of the hat to be in the sample. In particular, in a simple random sample, *every individual has an equal chance to be in the sample*, in fact, every subset of the population of a fixed size has an equal chance to be in the sample.

Other sampling methods include

convenience sampling using whatever individuals are easy to obtain

This is usually a terrible idea. If the convenient members of the population differ from the inconvenient members, then the sample will not be representative of the population.

©2014

volunteer sampling using people who volunteer to be in the sample

This is usually a terrible idea. Most likely the volunteers will differ in some ways from the non-volunteers, so again the sample will not be representative of the population.

systematic sampling sampling done in some systematic way (every tenth unit, for example).

This can sometimes be a reasonable approach.

stratified sampling sampling separately in distinct sub-populations (called *strata*)

This is more complicated (and sometimes necessary) but fine as long as the sampling methods in each stratum are good and the analysis takes the sampling method into account.

1.4 Experiments and Observational Studies

Types of Statistical Studies

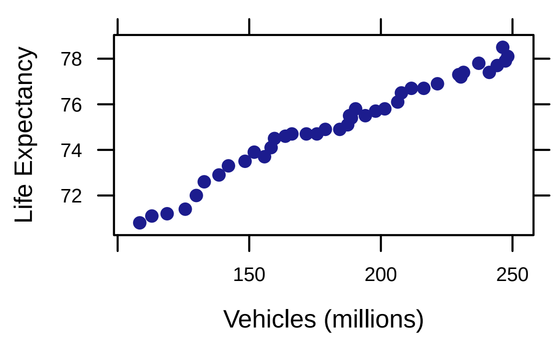
Statisticians use the word experiment to mean something very specific. *In an **experiment**, the researcher determines the values of one or more (explanatory) variables*, typically by random assignment. If there is no such assignment by the researcher, the study is an **observational study**.

Vehicles and Life Expectancy

```
head(LifeExpectancyVehicles, 10)
```

```
   Year LifeExpectancy Vehicles
1  1970           70.8    108.4
2  1971           71.1    113.0
3  1972           71.2    118.8
4  1973           71.4    125.7
5  1974           72.0    129.9
6  1975           72.6    132.9
7  1976           72.9    138.5
8  1977           73.3    142.1
9  1978           73.5    148.4
10 1979           73.9    151.9
```

```
xyplot(LifeExpectancy ~ Vehicles, xlab = "Vehicles (millions)", ylab = "Life Expectancy",
       data = LifeExpectancyVehicles)
```



 ©2014

Many of the datasets in R have useful help files that describe the data and explain how they were collected or give references to the original studies. You can access this information for the `AllCountries` data set by typing

```
?AllCountries
```

We'll learn how to make more customized summaries (numerical and graphical) soon. For now, it is only important to observe how the organization of data in R reflects the observational units and variables in the data set.

This is important if you want to construct your own data set (in Excel or a google spreadhseet, for example) that you will later import into R. You want to be sure that the structure of your spread sheet uses rows and columns in this same way, and that you don't put any extra stuff into the spread sheet. It is a good idea to include an extra row at the top which names the variables. Take a look at Chapter 0 to learn how to get the data from Excel into R.

 ©2014

2 Describing Data

In this chapter we discuss graphical and numerical summaries of data.

2.1 Categorical Variables

One Categorical Variable

Let us investigate categorical variables in R by taking a look at the data set for the One True Love survey. Notice that the data set is not readily available in our textbook's package. However, the authors do provide us with the necessary information to create our own data spreadsheet (in either Excel or Google) and import it into R. (See Chapter 0 for instructions.)

From the dataset we named as `OneTrueLove`, we can tabulate the categorical variable to display the *frequency* by using the `tally()` function. The default in tallying is to not include the row totals, or column totals when there are two variables. These are called marginal totals and if you want them, you can change the default.

tally-1way

```
OneTrueLove <- read.file("OneTrueLove.csv")
tally(~Response, margin = TRUE, data = OneTrueLove)
```

```
     Agree   Disagree Don't know      Total
       735       1812         78       2625
```

Note the use of the R template in Chapter 0.

In R, we can use the `prop()` function to quickly find proportions.

prop-1way

```
prop(~Response, data = OneTrueLove)
```

```
Agree
 0.28
```

Example 2.3

To find the proportion of responders who *disagree* or *don't know* we can use the `level=` argument in the function to find proportions.

```
prop(~Response, level = "Disagree", data = OneTrueLove)

Disagree
  0.6903

prop(~Response, level = "Don't know", data = OneTrueLove)

Don't know
   0.02971
```

Further, we can also display the *relative frequencies*, or **proportions** in a table.

tally-1way-proportion

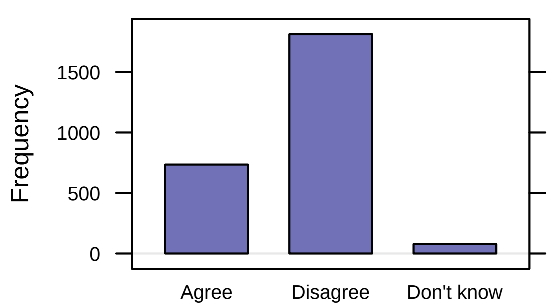
```
tally(~Response, format = "proportion", margin = TRUE, data = OneTrueLove)

     Agree   Disagree Don't know      Total
   0.28000    0.69029    0.02971    1.00000
```

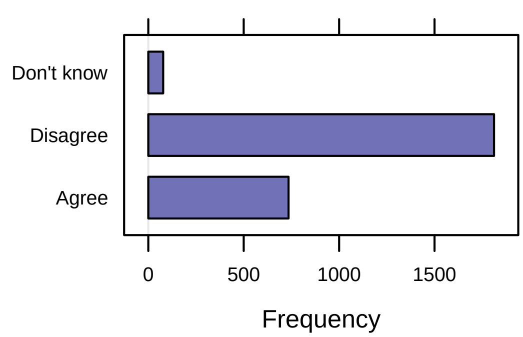
Visualizing the Data in One Categorical Variable

To visualize counts or proportions, R provides many different chart and plot functions, including *bar charts* and *pie charts*. Bar charts, also known as bar graphs, are a way of displaying the distribution of a categorical variable.

bargraph

```
bargraph(~Response, data = OneTrueLove)
bargraph(~Response, data = OneTrueLove, horizontal = TRUE)
```





 ©2014

Two Categorical Variables: Two-Way Tables

Often, it is useful to compute cross tables for two (or more) variables. We can again use `tally()` for several ways to investigate a two-way table.

tally-2way

```
tally(~Response + Gender, data = OneTrueLove)

            Gender
Response     Female Male
  Agree         363  372
  Disagree     1005  807
  Don't know     44   34

tally(~Response + Gender, margins = TRUE, data = OneTrueLove)

            Gender
Response     Female Male Total
  Agree         363  372   735
  Disagree     1005  807  1812
  Don't know     44   34    78
  Total        1412 1213  2625
```

Example 2.5

Similar to one categorical variable, we can use the `prop()` function to find the proportion of two variables. The first line results in the proportion of females who agree and the proportion of males who agree. The second line shows the proportion who agree that are female and the proportion who disagree that are female. The third results in the proportion of all the survey responders that are female.

prop-2way

```
prop(Response ~ Gender, data = OneTrueLove)

Agree.Female   Agree.Male
      0.2571       0.3067

prop(Gender ~ Response, data = OneTrueLove)

     Female.Agree   Female.Disagree Female.Don't know
           0.4939            0.5546            0.5641

prop(~Gender, data = OneTrueLove)

Female
0.5379
```

See though that because we have multiple levels of each variable, this process can become quite tedious if we want to find the proportions for all of the levels. Using the tally function a little differently will result in these proportions.

tally-2way-options

```
tally(Response ~ Gender, data = OneTrueLove)

            Gender
Response      Female    Male
  Agree      0.25708 0.30668
  Disagree   0.71176 0.66529
  Don't know 0.03116 0.02803

tally(~Response | Gender, data = OneTrueLove)

            Gender
Response      Female    Male
  Agree      0.25708 0.30668
  Disagree   0.71176 0.66529
  Don't know 0.03116 0.02803

tally(Gender ~ Response, data = OneTrueLove)

        Response
Gender    Agree Disagree Don't know
  Female 0.4939   0.5546     0.5641
  Male   0.5061   0.4454     0.4359

tally(~Gender | Response, data = OneTrueLove)

        Response
Gender    Agree Disagree Don't know
  Female 0.4939   0.5546     0.5641
  Male   0.5061   0.4454     0.4359
```

Notice that (by default) some of these use counts and some use proportions. Again, we can change the format.

tally-2way-percent

```
tally(~Gender, format = "percent", data = OneTrueLove)

Female   Male
 53.79  46.21
```

Example 2.6

tally-2way-2

```
tally(~Gender + Award, margin = TRUE, data = StudentSurvey)

       Award
Gender  Academy Nobel Olympic Total
  F          20    76      73   169
```


©2014

```
  M           11     73       109    193
  Total       31    149       182    362
```

Also, we can arrange the table differently by converting it to a data frame.

tally-2way-data-frame

```
as.data.frame(tally(~Gender + Award, data = StudentSurvey))
```

```
  Gender   Award Freq
1      F Academy   20
2      M Academy   11
3      F   Nobel   76
4      M   Nobel   73
5      F Olympic   73
6      M Olympic  109
```

prop-1way-2

```
prop(~Award, level = "Olympic", data = StudentSurvey)
```

```
Olympic
 0.5028
```

Example 2.7

To calculate the difference of certain statistics, we can use the `diff()` function. Here we use it to find the difference in proportions, but it can be used for means, medians, and etc.

diff-in-prop

```
diff(prop(Award ~ Gender, level = "Olympic", data = StudentSurvey))
```

```
Olympic.M
   0.1328
```

We will contine more with proportions in Chapter 3.

Visualizing a Relationship between Two Categorical Variables

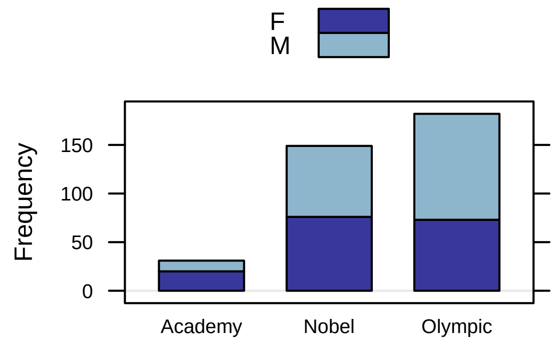
A way to look at multiple groups simultaneously is by using *comparative plots* such as a *segmented bar chart* or *side-by-side bar chart*. We use te `groups` argument for this. What groups does depends a bit on the type of graph. Using `groups` with `histogram()` doesn't work so well because it is difficult to overlay histograms.[1] Density plots work better for this.

Notice the addition of `groups=` (to group), `stack=` (to segment the graph), and `auto.key=TRUE` (to build a simple legend so we can tell which groups are which).

[1] The `mosaic` function `histogram()` does do something meaningful with `groups` in some situations.

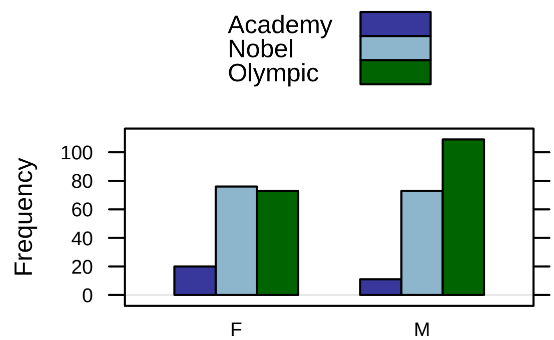
bargraph-comp

```
bargraph(~Award, groups = Gender, stack = TRUE, auto.key = TRUE, data = StudentSurvey)
```



bargraph-groups

```
bargraph(~Gender, groups = Award, data = StudentSurvey, auto.key = TRUE)
```



2.2 One Quantitative Variable: Shape and Center

The distribution of a variable answers two questions:

- *What values* can the variable have?
- *With what frequency* does each value occur?

  Again, the frequency may be described in terms of counts, proportions (often called relative frequency), or densities (more on densities later).

A distribution may be described using a table (listing values and frequencies) or a graph (e.g., a histogram) or with words that describe general features of the distribution (e.g., symmetric, skewed).

 ©2014

The Shape of a Distribution

Statisticians have devised a number of graphs to help us see distributions visually. The general syntax for making a graph of one variable in a data frame is

```
plotname(~variable, data = dataName)
```

In other words, there are three pieces of information we must provide to R in order to get the plot we want:

- The kind of plot (`histogram()`, `bargraph()`, `densityplot()`, `bwplot()`, etc.)
- The name of the variable
- The name of the data frame this variable is a part of.

This should look familiar from the previous section.

Dot Plot

Let's make a *dot plot* of the variable Longevity in the `MammalLongevity` data set for a quick and simple look at the distribution. We use the syntax provided above with two additional arguments to make the figure look the way we want it to. The next few sections will explain a few of the different arguments available for plots in R.

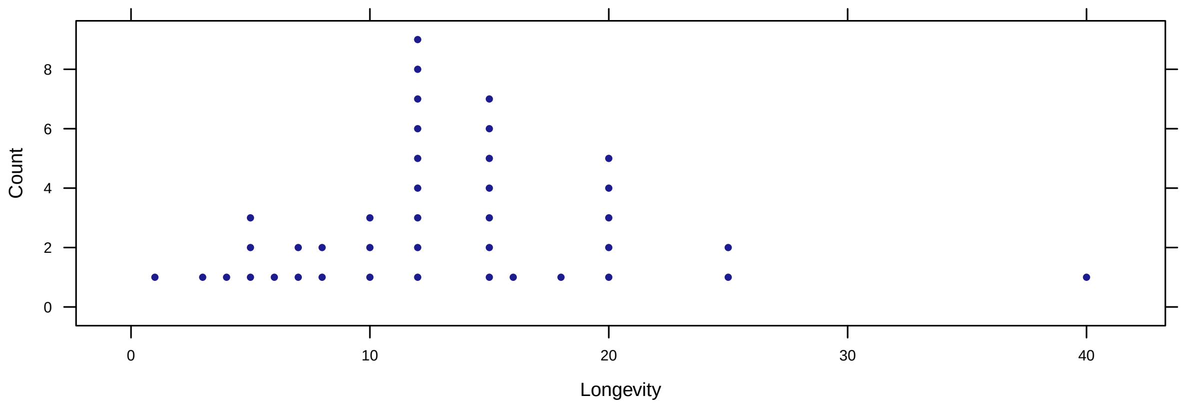
dotplot

```
MammalLongevity
```

```
         Animal Gestation Longevity
1        baboon       187        20
2    bear,black       219        18
3  bear,grizzly       225        25
4    bear,polar       240        20
5        beaver       122         5
6       buffalo       278        15
7         camel       406        12
8           cat        63        12
9    chimpanzee       231        20
10     chipmunk        31         6
11          cow       284        15
12         deer       201         8
13          dog        61        12
14       donkey       365        12
15     elephant       645        40
16          elk       250        15
17          fox        52         7
18      giraffe       425        10
19         goat       151         8
20      gorilla       257        20
21   guinea pig        68         4
22 hippopotamus       238        25
23        horse       330        20
24     kangaroo        42         7
25      leopard        98        12
26         lion       100        15
27       monkey       164        15
28        moose       240        12
```

©2014

```
29        mouse     21    3
30      opposum     15    1
31          pig    112   10
32         puma     90   12
33       rabbit     31    5
34   rhinoceros    450   15
35     sea lion    350   12
36        sheep    154   12
37     squirrel     44   10
38        tiger    105   16
39         wolf     63    5
40        zebra    365   15

dotPlot(~Longevity, width = 1, cex = 0.1, data = MammalLongevity)
```



Histograms and Density Plots

Although `tally()` works with quantitative variables as well as categorical variables, this is only useful when there are not too many different values for the variable.

xabs-quantitative

```
tally(~Longevity, margin = TRUE, data = MammalLongevity)


    1     3     4     5     6     7     8    10    12    15    16    18    20    25    40
    1     1     1     3     1     2     2     3     9     7     1     1     5     2     1
Total
   40
```

Sometimes, it is more convenient to group them into bins. We just have to tell R what the bins are. For example, suppose we wanted to group together by 5.

cut

```
binnedLong <- cut(MammalLongevity$Longevity, breaks = c(0, 5, 10, 15, 20, 25, 30, 35, 40))
tally(~binnedLong)  # no data frame given because its not in a data frame
```

```
 (0,5]  (5,10] (10,15] (15,20] (20,25] (25,30] (30,35] (35,40]
     6       8      16       7       2       0       0       1
```

Suppose we wanted to group the 1s, 10s, 20s, etc. together. We want to make sure then that 10 is with the 10s, so we should add another argument.

cut2

```
binnedLong2 <- cut(MammalLongevity$Longevity, breaks = c(0, 10, 20, 30, 40, 50), right = FALSE)
tally(~binnedLong2)  # no data frame given because it's not in a data frame
```

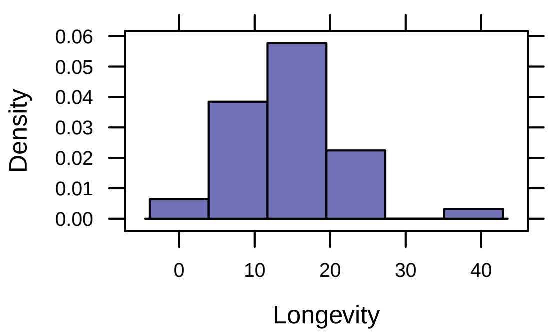
```
 [0,10) [10,20) [20,30) [30,40) [40,50)
     11      21       7       0       1
```

We won't use this very often however, since seeing this information in a histogram is typically more useful.

Histograms are a way of displaying the distribution of a quantitative variable.

histogram

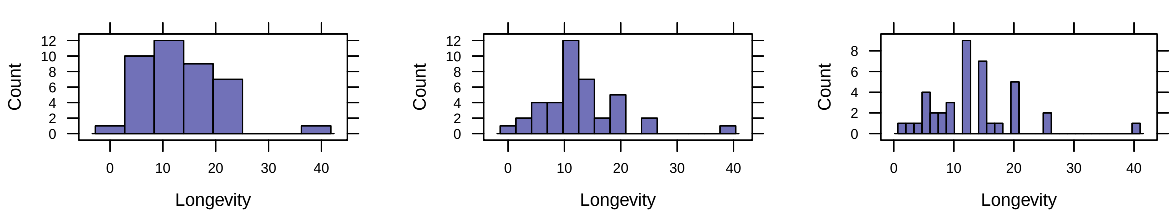
```
histogram(~Longevity, data = MammalLongevity)
```



We can control the (approximate) number of bins using the `nint` argument, which may be abbreviated as `n`. The number of bins (and to a lesser extent the positions of the bins) can make a histogram look quite different.

histogram2

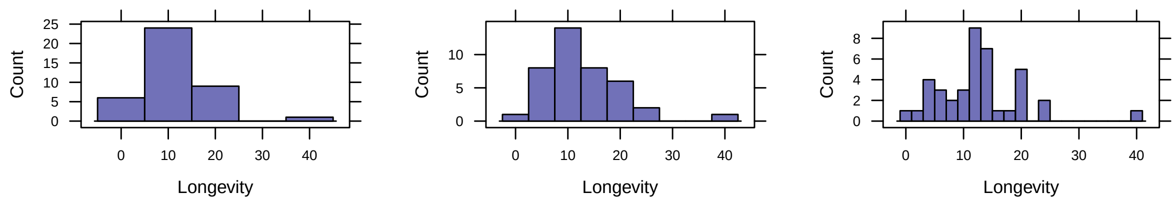
```
histogram(~Longevity, type = "count", data = MammalLongevity, n = 8)
histogram(~Longevity, type = "count", data = MammalLongevity, n = 15)
histogram(~Longevity, type = "count", data = MammalLongevity, n = 30)
```



We can also describe the bins in terms of center and width instead of in terms of the number of bins. This is especially nice for count or other integer data.

xhistogram

```
histogram(~Longevity, type = "count", data = MammalLongevity, width = 10)
histogram(~Longevity, type = "count", data = MammalLongevity, width = 5)
histogram(~Longevity, type = "count", data = MammalLongevity, width = 2)
```

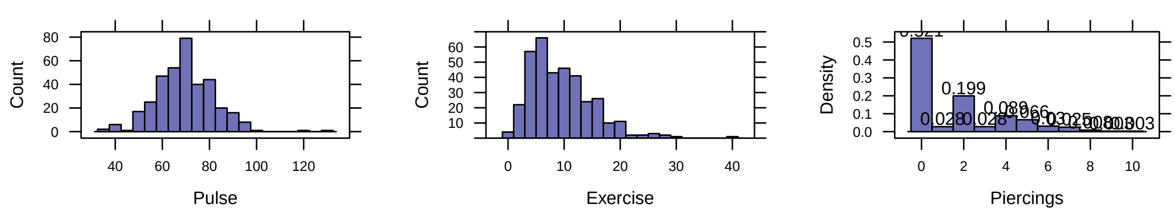


Example 2.9

Note the various options available for the `histogram()` function enable us to replicate Figure 2.8 some including centering, adding counts, labels, and limit to the y-axis (similar for x-axis) .

Example2.9

```
histogram(~ Pulse, type='count', width=5, data=StudentSurvey)
histogram(~ Exercise, type='count', width=2, center=2,
        right=FALSE, ylim=c(0,70), data=StudentSurvey)
histogram(~ Piercings, width=1, label=TRUE, data=StudentSurvey)
```

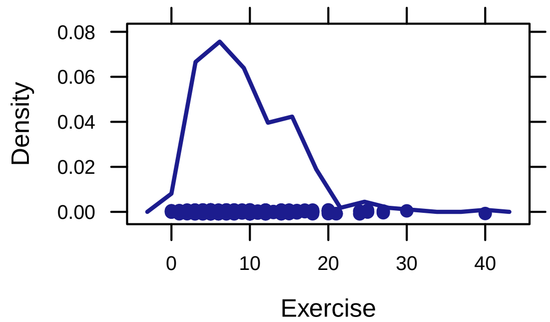


Sometimes a **frequency polygon** provides a more useful view. The only thing that changes is `histogram()` becomes `freqpolygon()`.

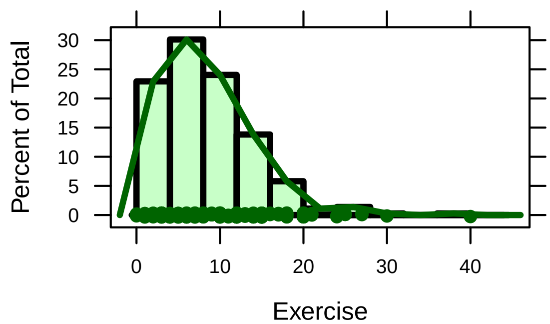
freqpolygon

```
freqpolygon(~Exercise, data = StudentSurvey, width = 5)
```

 ©2014



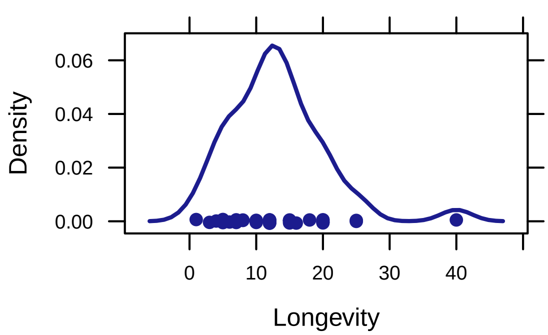
What is a frequency polygon? The picture below shows how it is related to a histogram. The frequency polygon is just a dot-to-dot drawing through the centers of the tops of the bars of the histogram.

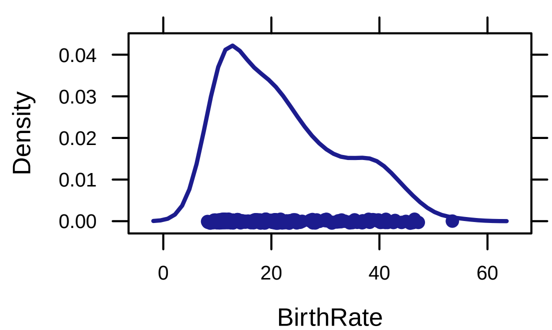


R also provides a "smooth" version called a density plot; just change the function name from `histogram()` to `densityplot()`.

densityplot

```
densityplot(~Longevity, data = MammalLongevity)
densityplot(~BirthRate, data = AllCountries)
```





If we make a histogram (or any of these other plots) of our data, we can describe the overall shape of the distribution. Keep in mind that the shape of a particular histogram may depend on the choice of bins. Choosing too many or too few bins can hide the true shape of the distribution. (When in doubt, make more than one histogram.)

Here are some words we use to describe shapes of distributions.

symmetric The left and right sides are mirror images of each other.

skewed The distribution stretches out farther in one direction than in the other. (We say the distribution is skewed toward the long tail.)

uniform The heights of all the bars are (roughly) the same. (So the data are equally likely to be anywhere within some range.)

unimodal There is one major "bump" where there is a lot of data.

bimodal There are two "bumps".

outlier An observation that does not fit the overall pattern of the rest of the data.

The Center of a Distribution

Recall that a statistic is a number computed from data. The **mean** and the **median** are key statistics which describe the center of a distribution. We can see through Example 2.11 that numerical summaries are computed using the same template as graphical summaries.

Subsets

Note however, that the example asks about subsets of `ICUAdmissions`–specifically about 20-year-old and 55-year-old patients. In this case, we can manipulate the data (to name a new data set) with the subset command. Here are some examples.

1. Select only the males from the `ICUAdmissions` data set.

```
tally(~Sex, data = ICUAdmissions)


  0   1
124  76

ICUMales <- subset(ICUAdmissions, Sex == "Male")  # notice the double =
dim(ICUMales)

[1]  0 42
```

2. Select only the subjects over 50:

```
ICUold <- subset(ICUAdmissions, Age > 50)
```

The `subset()` function can use any condition that evaluates to TRUE or FALSE for each row (case) in the data set.

Example 2.11

center

```
ICU20 <- subset(ICUAdmissions, Age == "20")
mean(~HeartRate, data = ICU20)
```

```
[1] 82.2
```

```
median(~HeartRate, data = ICU20)
```

```
[1] 80
```

```
ICU55 = subset(ICUAdmissions, Age == "55")
mean(~HeartRate, data = ICU55)
```

```
[1] 108.5
```

```
median(~HeartRate, data = ICU55)
```

```
[1] 106
```

Visualizing the Mean and Median on a Graph

Example 2.14

FloridaLakes-example

```
head(FloridaLakes)
```

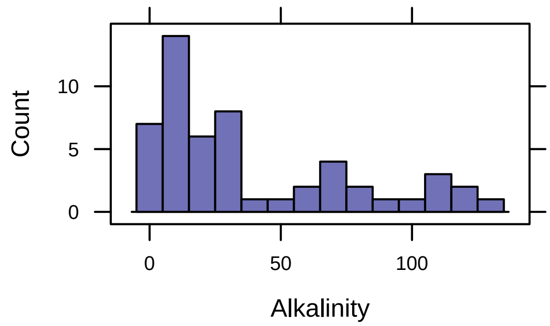
```
  ID         Lake Alkalinity  pH Calcium Chlorophyll AvgMercury NumSamples MinMercury
1  1    Alligator        5.9 6.1     3.0         0.7       1.23          5       0.85
2  2        Annie        3.5 5.1     1.9         3.2       1.33          7       0.92
3  3       Apopka      116.0 9.1    44.1       128.3       0.04          6       0.04
4  4 Blue Cypress       39.4 6.9    16.4         3.5       0.44         12       0.13
5  5        Brick        2.5 4.6     2.9         1.8       1.20         12       0.69
6  6       Bryant       19.6 7.3     4.5        44.1       0.27         14       0.04
  MaxMercury ThreeYrStdMercury AgeData
1       1.43              1.53       1
2       1.90              1.33       0
3       0.06              0.04       0
4       0.84              0.44       0
5       1.50              1.33       1
6       0.48              0.25       1
```

```
histogram(~Alkalinity, width = 10, type = "count", data = FloridaLakes)
mean(~Alkalinity, data = FloridaLakes)
```

```
[1] 37.53
```

```
median(~Alkalinity, data = FloridaLakes)
```

```
[1] 19.6
```



2.3 One Quantitative Variable: Measures of Spread

In the previous section, we investigated center summary statistics. In this section, we will cover some other important statistics.

Example 2.15

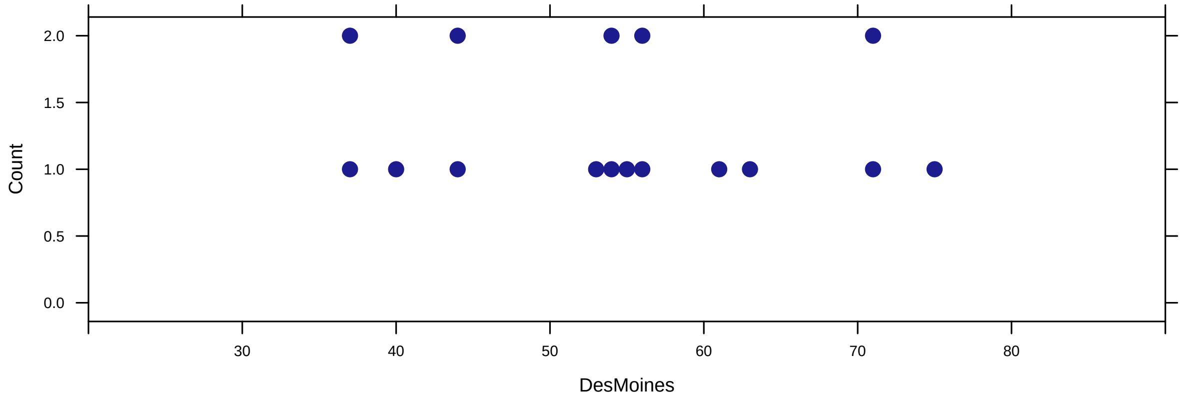
numerical-summaries

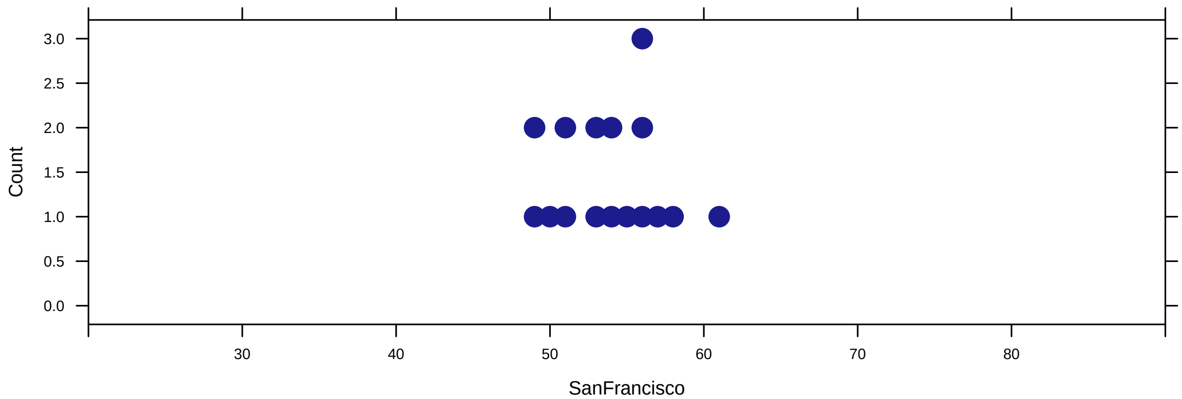
```
summary(April14Temps)

      Year        DesMoines     SanFrancisco
 Min.   :1995   Min.   :37.2   Min.   :48.7
 1st Qu.:1999   1st Qu.:44.4   1st Qu.:51.3
 Median :2002   Median :54.5   Median :54.0
 Mean   :2002   Mean   :54.5   Mean   :54.0
 3rd Qu.:2006   3rd Qu.:61.3   3rd Qu.:55.9
 Max.   :2010   Max.   :74.9   Max.   :61.0

favstats(~DesMoines, data = April14Temps)  # some favorite statistics

  min   Q1 median    Q3  max  mean    sd  n missing
 37.2 44.4   54.5 61.28 74.9 54.49 11.73 16       0
```

Standard Deviation

The density plots of the temperatures of Des Moines and San Francisco reveal that Des Moines has a greater *variability* or *spread*.

 ©2014





Example 2.16

Although both `summary()` and `favstats()` calculate the **standard deviation** of a variable, we can also use `sd()` to find just the standard deviation.

standard-deviation

```
sd(~DesMoines, data = April14Temps)

[1] 11.73

sd(~SanFrancisco, data = April14Temps)

[1] 3.377

var(~DesMoines, data = April14Temps)  # variance = sd^2

[1] 137.6
```

Example 2.17

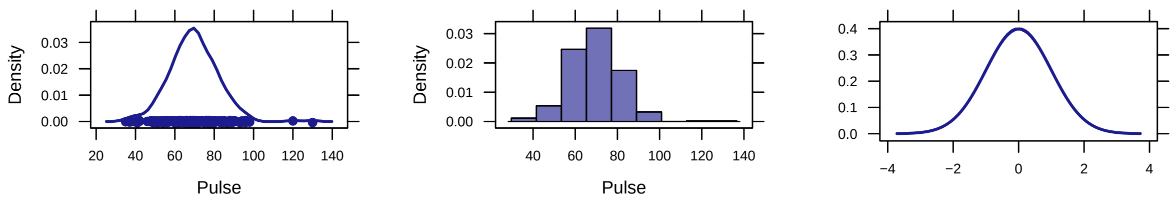
```
densityplot(~Pulse, data = StudentSurvey)
histogram(~Pulse, data = StudentSurvey)
plotDist("norm")
mean(~Pulse, data = StudentSurvey)
```

Example2.17

```
[1] 69.57
```

```
sd(~Pulse, data = StudentSurvey)
```

```
[1] 12.21
```



Example 2.18

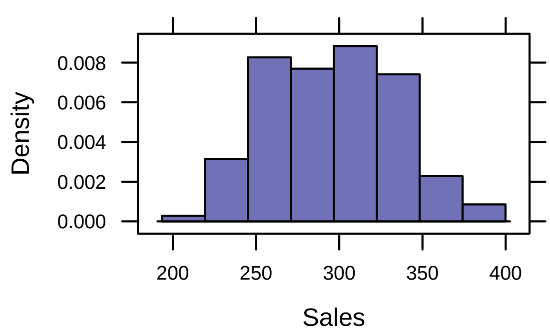
```
histogram(~Sales, data = RetailSales)
mean(~Sales, data = RetailSales)
```

Example2.18

```
[1] 296.4
```

```
sd(~Sales, data = RetailSales)
```

```
[1] 37.97
```



Example 2.19

Z-scores can be computed as follows:

z-score

```
(204 - mean(~Systolic, data = ICUAdmissions))/sd(~Systolic, data = ICUAdmissions)

[1] 2.176

(52 - mean(~HeartRate, data = ICUAdmissions))/sd(~HeartRate, data = ICUAdmissions)

[1] -1.749
```

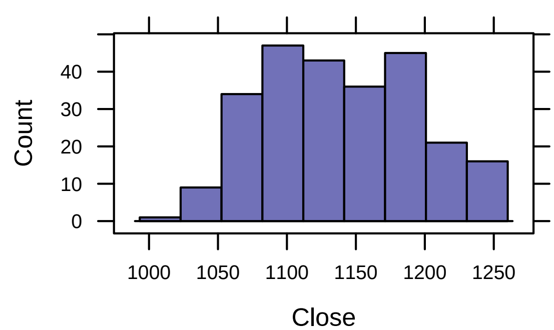
Percentile

Example 2.20

The text uses a histogram to estimate the **percentile** of the daily closing price for the SP 500 but we can also find the exact percentiles using the `quantile()` function.

percentile

```
histogram(~Close, type = "count", data = SandP500)
quantile(SandP500$Close, probs = seq(0, 1, 0.25))

  0%  25%  50%  75% 100%
1023 1095 1137 1183 1260

quantile(SandP500$Close, probs = seq(0, 1, 0.9))

  0%  90%
1023 1217
```



Five Number Summary

We have already covered many different functions which results in the **five number summary** but `fivenum()` is most direct way to obtain in the five number summary.

Example 2.21

```
fivenum(~Exercise, data = StudentSurvey)
```

five-number-summary

```
[1]  0  5  8 12 40
```

Example 2.22

```
fivenum(~Longevity, data = MammalLongevity)
```

range

```
[1]  1.0  8.0 12.0 15.5 40.0

min(~Longevity, data = MammalLongevity)

[1] 1

max(~Longevity, data = MammalLongevity)

[1] 40

range(~Longevity, data = MammalLongevity)  # subtract to get the numerical range value

[1]  1 40

iqr(~Longevity, data = MammalLongevity)  # inter-quartile range

[1] 7.25
```

Note the difference in the quartile and IQR from the textbook. This results because there are several different methods to determine the quartile.

Example 2.23

```
fivenum(~DesMoines, data = April14Temps)
```

Example2.23

```
[1] 37.20 44.40 54.50 61.95 74.90

fivenum(~SanFrancisco, data = April14Temps)

[1] 48.7 51.2 54.0 56.0 61.0

range(~DesMoines, data = April14Temps)
```


©2014

```
[1] 37.2 74.9

74.9 - 37.2

[1] 37.7

range(~SanFrancisco, data = April14Temps)

[1] 48.7 61.0

61 - 48.7

[1] 12.3

iqr(~DesMoines, data = April14Temps)

[1] 16.88

iqr(~SanFrancisco, data = April14Temps)

[1] 4.6
```

2.4 Outliers, Boxplots, and Quantitative/Categorical Relationships

Detection of Outliers

Generally, outliers are considered to be values

- less than $Q_1 - 1.5 \cdot (IQR)$, and
- greater than $Q_3 + 1.5 \cdot (IQR)$.

Example 2.25

outliers

```
fivenum(~Longevity, data = MammalLongevity)

[1]  1.0  8.0 12.0 15.5 40.0

iqr(~Longevity, data = MammalLongevity)

[1] 7.25

8 - 1.5 * 7.25
```

```
[1] -2.875

15.5 + 1.5 * 7.25

[1] 26.38

subset(MammalLongevity, Longevity > 26.375)

     Animal Gestation Longevity
15 elephant       645        40
```

There is no function in R that directly results in outliers because practically, there is no one specific formula for such a determination. However, a boxplot will indirectly reveal outliers.

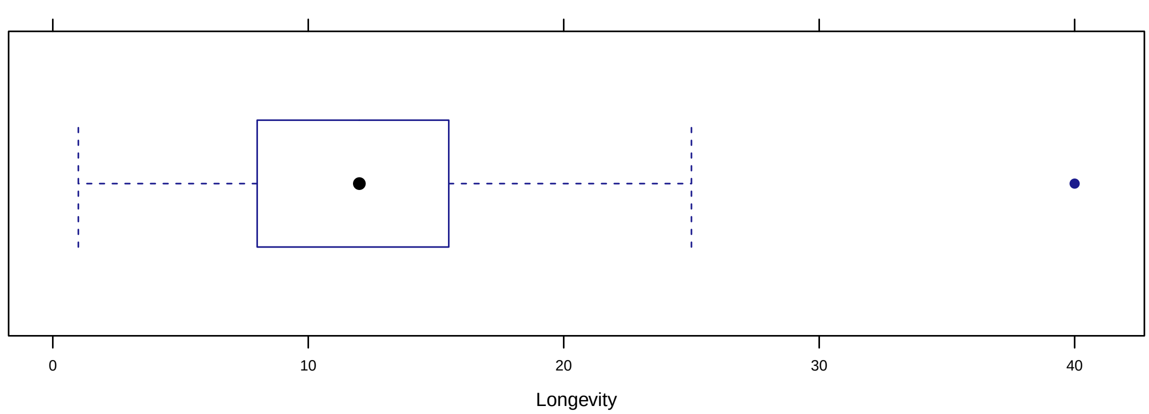
Boxplots

A way to visualize the five number summary and outliers for a variable is to create a boxplot.

Example 2.26

```
favstats(~Longevity, data = MammalLongevity)

 min Q1 median    Q3 max  mean    sd  n missing
   1  8     12 15.25  40 13.15 7.245 40       0

bwplot(~Longevity, data = MammalLongevity)
```

boxplot



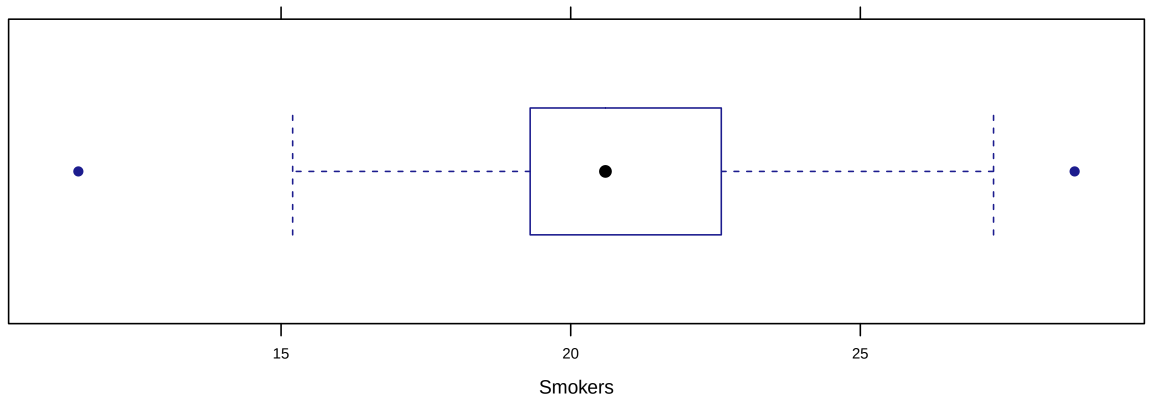
Example 2.27

We can similarity investigate the *Smokers* variable in USStates.

©2014

boxplot2

```
bwplot(~Smokers, data = USStates)
fivenum(~Smokers, data = USStates)

[1] 11.5 19.3 20.6 22.6 28.7
```



The boxplot reveals two outliers. To identify them, we can again use `subset()` for smokers greater or less than the *whiskers* of the boxplot.

subset-outliers

```
subset(USStates, Smokers < 15)

   State HouseholdIncome    IQ McCainVote Region ObamaMcCain Population EighthGradeMath
44  Utah           55619 101.1      0.629      W           M      2.421           279.2
   HighSchool   GSP FiveVegetables Smokers PhysicalActivity Obese College NonWhite
44         91 36758           22.1    11.5             83.1  21.2      31     12.1
   HeavyDrinkers Pres2008
44           2.9   McCain

subset(USStates, Smokers > 28)

      State HouseholdIncome   IQ McCainVote Region ObamaMcCain Population EighthGradeMath
17 Kentucky           38694 99.4      0.575     MW           M      4.142             274
   HighSchool   GSP FiveVegetables Smokers PhysicalActivity Obese College NonWhite
17       81.8 33666           16.8    28.7             70.1  28.6    22.6      9.4
   HeavyDrinkers Pres2008
17           2.7   McCain
```

Example 2.28

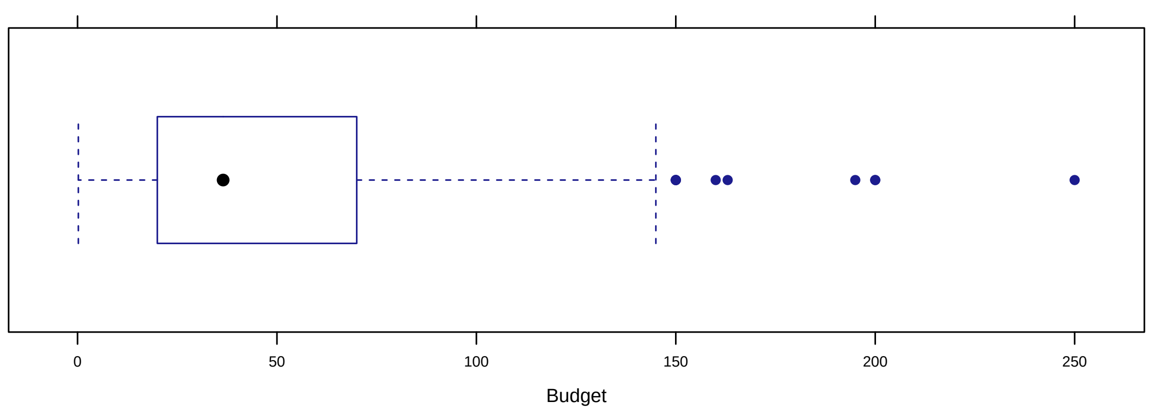
boxplot3

```
bwplot(~Budget, data = HollywoodMovies2011)
subset(HollywoodMovies2011, Budget > 225)

                                          Movie LeadStudio RottenTomatoes AudienceScore
```

```
30 Pirates of the Caribbean:\nOn Stranger Tides     Disney            34             61
   Story  Genre TheatersOpenWeek BOAverageOpenWeek DomesticGross ForeignGross WorldGross
30 Quest Action             4155             21697         241.1        802.8       1044
   Budget Profitability OpeningWeekend
30    250         4.175          90.15
```

```
head(HollywoodMovies2011)
```

```
                                        Movie       LeadStudio RottenTomatoes
1                                   Insidious             Sony             67
2                        Paranormal Activity 3      Independent             68
3                                 Bad Teacher      Independent             44
4 Harry Potter and the Deathly Hallows Part 2      Warner Bros             96
5                                 Bridesmaids Relativity Media             90
6                             Midnight in Paris             Sony             93
  AudienceScore         Story   Genre TheatersOpenWeek BOAverageOpenWeek DomesticGross
1            65 Monster Force  Horror             2408              5511         54.01
2            58 Monster Force  Horror             3321             15829        103.66
3            38        Comedy  Comedy             3049             10365        100.29
4            92       Rivalry Fantasy             4375             38672        381.01
5            77       Rivalry  Comedy             2918              8995        169.11
6            84          Love Romance              944              6177         56.18
  ForeignGross WorldGross Budget Profitability OpeningWeekend
1        43.00      97.01    1.5        64.673          13.27
2        98.24     201.90    5.0        40.379          52.57
3       115.90     216.20   20.0        10.810          31.60
4       947.10    1328.11  125.0        10.625         169.19
5       119.28     288.38   32.5         8.873          26.25
6        83.00     139.18   17.0         8.187           5.83
```



Quantitative and Categorical Variables

The formula for a `lattice` plot can be extended to create multiple panels (sometimes called **facets**) based on a "condition", often given by another variable. This is another way to look at multiple groups simultaneously. The general syntax for this becomes

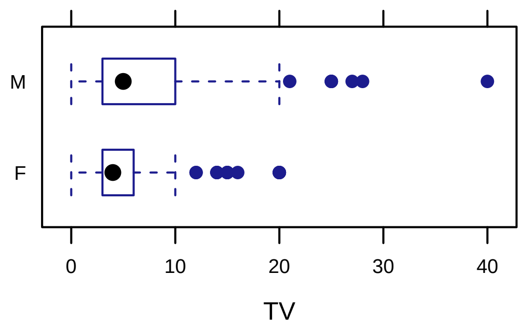
```
plotname(~variable | condition, data = dataName)
```

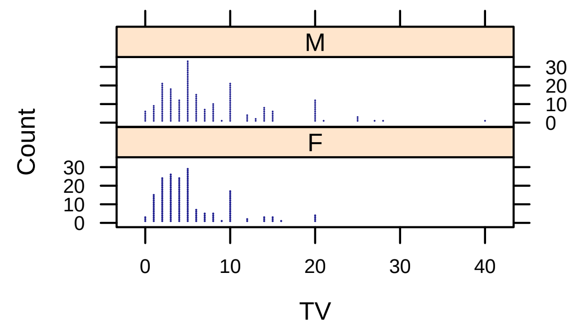
Example 2.29

Depending on the type of plot, you will want to use conditioning.

conditionalplots

```
bwplot(Gender ~ TV, data = StudentSurvey)
dotPlot(~TV | Gender, layout = c(1, 2), width = 1, cex = 0.25, data = StudentSurvey)
```

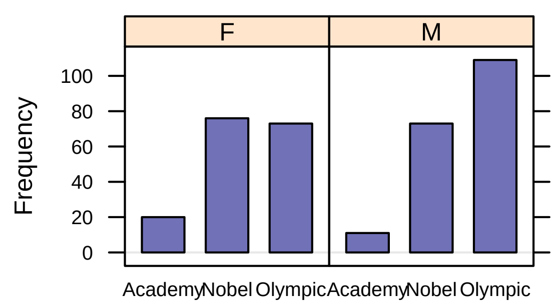




We can do the same thing for bar graphs.

conditionalplot-categorical

```
bargraph(~Award | Gender, data = StudentSurvey)
```



This graph should be familiar as we have plotted these variables together previously. Here we used different panels, but before, in 2.1, we had used grouping. Note that we can combine grouping and conditioning in the same plot.

Example 2.31

comparative-summary-statistics

```
favstats(~TV | Gender, data = StudentSurvey)
```

```
  .group min Q1 median Q3 max  mean    sd   n missing
1      F   0  3      4  6  20 5.237 4.100 169       0
2      M   0  3      5 10  40 7.620 6.427 192       1
```

```
diff(mean(~TV | Gender, data = StudentSurvey))
```

```
 M
NA
```

2.5 Two Quantitative Variables: Scatterplot and Correlation

Example 2.32

```
ElectionMargin
```

Example2.32

```
   Year  Candidate Approval Margin Result
1  1940  Roosevelt       62   10.0    Won
2  1948     Truman       50    4.5    Won
3  1956 Eisenhower       70   15.4    Won
4  1964    Johnson       67   22.6    Won
5  1972      Nixon       57   23.2    Won
6  1976       Ford       48   -2.1   Lost
7  1980     Carter       31   -9.7   Lost
8  1984     Reagan       57   18.2    Won
9  1992 G.H.W.Bush       39   -5.5   Lost
10 1996    Clinton       55    8.5    Won
11 2004   G.W.Bush       49    2.4    Won
```

Visualizing a Relationship between Two Quantitative Variables: Scatterplots

The most common way to look at two quantitative variables is with a scatterplot. The `lattice` function for this is `xyplot()`, and the basic syntax is

xyplot-syntax

```
xyplot(yvar ~ xvar, data = dataName)
```

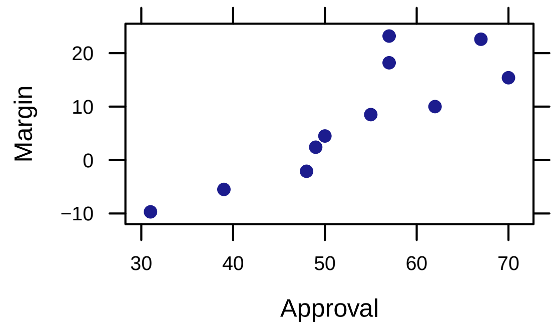
Notice that now we have something on both sides of the ˜ since we need to tell R about two variables.

Example 2.33

```
xyplot(Margin ~ Approval, data = ElectionMargin)
```

xyplot1

 ©2014



Grouping and conditioning work just as before. With large data set, it can be helpful to make the dots semi-transparent so it is easier to see where there are overlaps. This is done with `alpha`. We can also make the dots smaller (or larger) using `cex`.

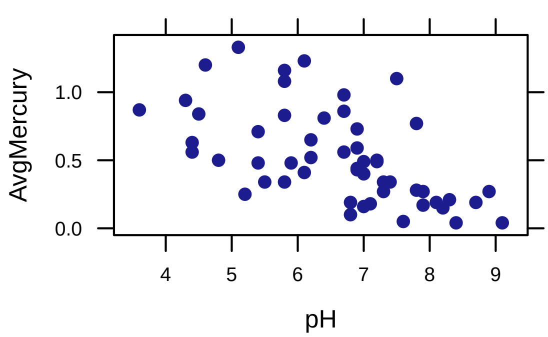
xyplot2

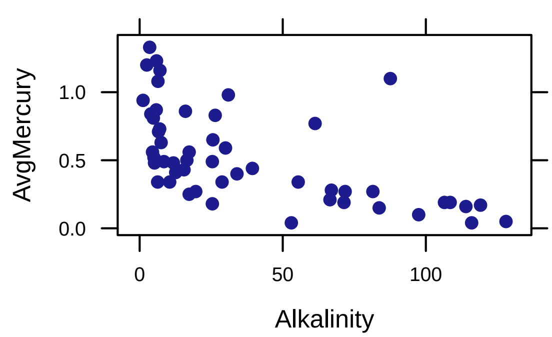
```
xyplot(mcs ~ age | sex, groups = substance, data = HELPrct, alpha = 0.6, cex = 0.5, auto.key = TRUE)
```

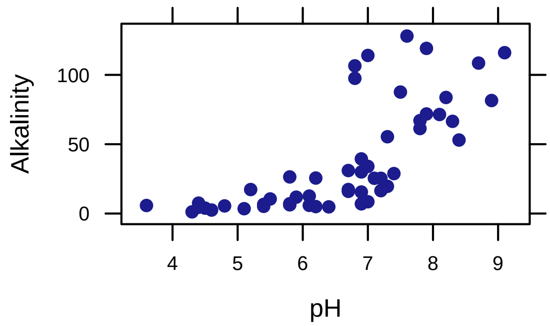
Example 2.34

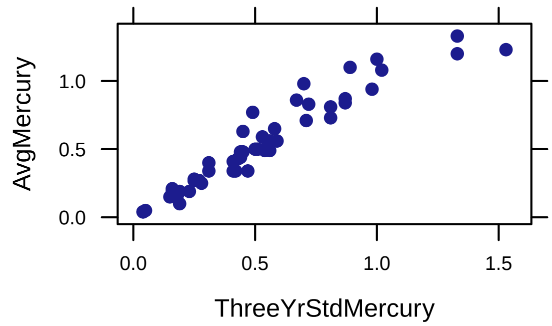
xyplot3

```
xyplot(AvgMercury ~ pH, data = FloridaLakes)
xyplot(AvgMercury ~ Alkalinity, data = FloridaLakes)
xyplot(Alkalinity ~ pH, data = FloridaLakes)
xyplot(AvgMercury ~ ThreeYrStdMercury, data = FloridaLakes)
```









Summarizing a Relationship between Two Quantitative Variables: Correlation

Another key numerical statistic is the **correlation**–the correlation is a measure of the strength and direction of the relationship between two quantitative variables.

correlation

```
cor(Margin ~ Approval, data = ElectionMargin)

[1] 0.863

cor(AvgMercury ~ pH, data = FloridaLakes)

[1] -0.5754

cor(AvgMercury ~ Alkalinity, data = FloridaLakes)

[1] -0.5939

cor(Alkalinity ~ pH, data = FloridaLakes)

[1] 0.7192

cor(AvgMercury ~ ThreeYrStdMercury, data = FloridaLakes)

[1] 0.9592
```

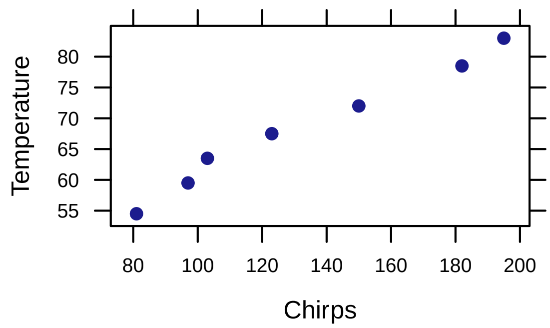
Example 2.35

Example2.35

```
CricketChirps

  Temperature Chirps
1        54.5     81
```

 ©2014

```
2         59.5     97
3         63.5    103
4         67.5    123
5         72.0    150
6         78.5    182
7         83.0    195

xyplot(Temperature ~ Chirps, data = CricketChirps)
cor(Temperature ~ Chirps, data = CricketChirps)

[1] 0.9906
```

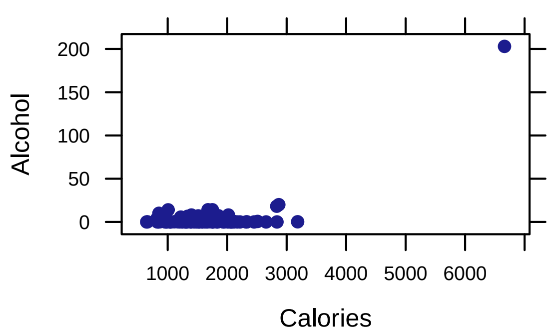


Example 2.38

Further, using the `subset()` function again, we can investigate the correlation between variables with some restrictions.

Example2.38

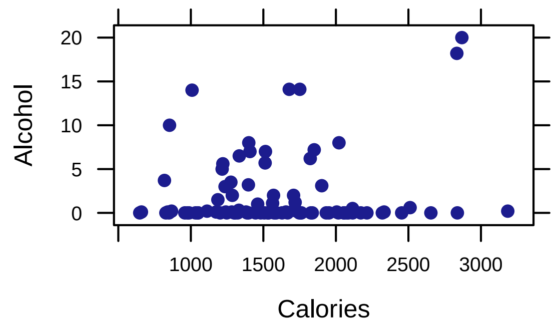
```
xyplot(Alcohol ~ Calories, data = subset(NutritionStudy, Age > 59))
cor(Alcohol ~ Calories, data = subset(NutritionStudy, Age > 59))

[1] 0.72
```



And now we omit the outlier

omitoutlier

```
NutritionStudy60 = subset(NutritionStudy, Age > 59)
xyplot(Alcohol ~ Calories, data = subset(NutritionStudy60, Alcohol < 25))
cor(Alcohol ~ Calories, data = subset(NutritionStudy60, Alcohol < 25))

[1] 0.145
```



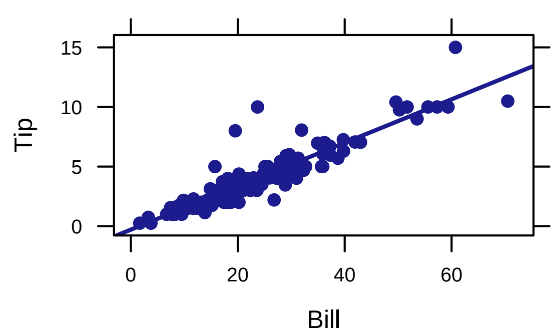
2.6 Two Quantitative Variables: Linear Regression

When the relationship between variables is sufficiently *linear*, you may be able to predict the value of a variable using the other variable. This is possible by fitting a *regression line*. To plot this in R, all we need to do is add an additional argument, `type=c("p", "r")`, to the xyplot.

Example 2.39 2.40

linearregression

```
xyplot(Tip ~ Bill, type = c("p", "r"), data = RestaurantTips)
cor(Tip ~ Bill, data = RestaurantTips)

[1] 0.9151
```



 ©2014

The equation for the regression line, or the *prediction equation* is

$$\widehat{\text{Response}} = a + b \cdot \text{Explanatory}$$

So now, we need to find the values for a, the intercept, and b, the slope using the function to fit linear models.

Example 2.41

```
lm(Tip ~ Bill, data = RestaurantTips)
```

Example2.41

```

Call:
lm(formula = Tip ~ Bill, data = RestaurantTips)

Coefficients:
(Intercept)         Bill
     -0.292        0.182
```

```
coef(lm(Tip ~ Bill, data = RestaurantTips))  # just show me the coefficients
```

```
(Intercept)        Bill
    -0.2923      0.1822
```

This results in the equation

$$\widehat{\text{Tip}} = -0.2923 + 0.1822 \cdot \text{Bill}$$

With this equation, one can predict the tip for different bill amounts.

An important aspect of the linear regression is the difference between the prediction and actual observation. This is called the **residual**, defined

$$\text{residual} = \text{observed response} - \text{predicted response}$$

Example 2.43

```
lm(Margin ~ Approval, data = ElectionMargin)
```

Example2.43

```

Call:
lm(formula = Margin ~ Approval, data = ElectionMargin)

Coefficients:
(Intercept)     Approval
    -36.483        0.836
```

```
residuals(lm(Margin ~ Approval, data = ElectionMargin))
```

```
      1       2       3       4       5       6       7       8       9      10      11
-5.3229 -0.7959 -6.6075  3.0992 12.0551 -5.7247  0.8802  7.0551 -1.6045 -0.9738 -2.0603
```

Example 2.45

```
xyplot(AvgMercury ~ pH, type = c("p", "r"), data = FloridaLakes)
lm(AvgMercury ~ pH, data = FloridaLakes)
```

Example2.45

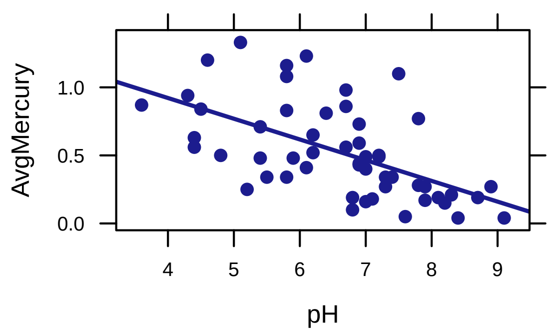
```
Call:
lm(formula = AvgMercury ~ pH, data = FloridaLakes)

Coefficients:
(Intercept)           pH
      1.531       -0.152
```

```
1.53 - 0.1523 * 7.5  # predicted value at 7.5 pH
```

```
[1] 0.3878
```

```
1.1 - 0.388  # residual at 7.5 pH
```

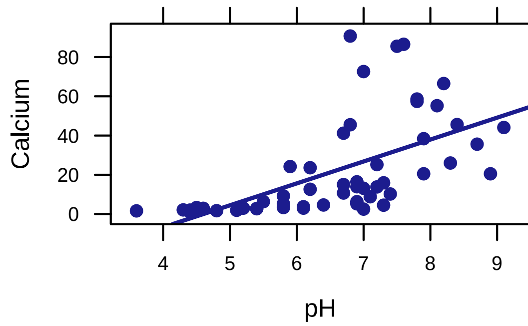
```
[1] 0.712
```



Example 2.46

```
xyplot(Calcium ~ pH, type = c("p", "r"), data = FloridaLakes)
```

Example2.46



2.7 Graphical Summaries – Important Ideas

Patterns and Deviations from Patterns

The goal of a statistical plot is to help us see

- potential patterns in the data, and
- deviations from those patterns.

Different Plots for Different Kinds of Variables

Graphical summaries can help us see the *distribution* of a variable or the *relationships* between two (or more) variables. The type of plot used will depend on the kinds of variables involved. You can use `demo()` to see how to get R to make the plots in this section.

When we do statistical analysis, we will see that the analysis we use will also depend on the kinds of variables involved, so this is an important idea.

Side-by-side Plots and Overlays Can Reveal Importance of Additional Factors

The `lattice` graphics plots make it particularly easy to generate plats that divide the data into groups and either produce a panel for each group (using |) or display each group in a different way (different colors or symbols, using the `groups` argument). These plots can reveal the possible influence of additional variables – sometimes called covariates.

Area = (relative) frequency

Many plots are based on the key idea that our eyes are good at comparing areas. Plots that use area (e.g., histograms, mosaic plots, bar charts, pie charts) should always obey this principle

$$\text{Area} = \text{(relative) frequency}$$

Plots that violate this principle can be deceptive and distort the true nature of the data.

©2014

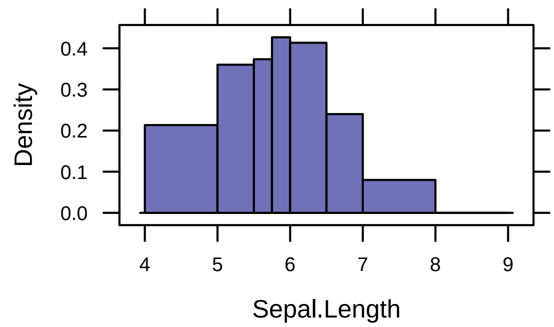
An Example: Histogram with unequal bin widths

It is possible to make histograms with bins that have different widths. But in this case it is important that the height of the bars is chosen so that area (*NOT height*) is proportional to frequency. Using height instead of area would distort the picture.

When unequal bin sizes are specified, `histogram()` by default chooses the density scale:

hist-unequal-bins

```
histogram(~Sepal.Length, data = iris, breaks = c(4, 5, 5.5, 5.75, 6, 6.5, 7, 8, 9))
```



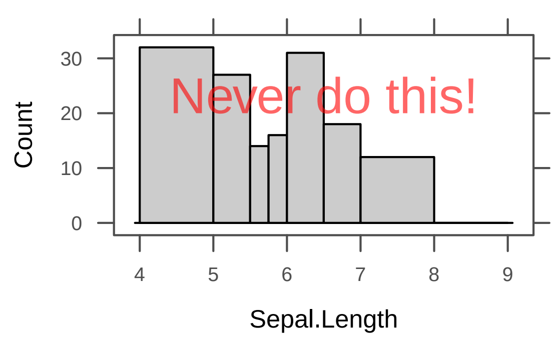
The density scale is important. It tells R to use a scale such that the area (height × width) of the rectangles is equal to the relative frequency. For example, the bar from 5.0 to 5.5 has width $\frac{1}{2}$ and height about 0.36, so the area is 0.18, which means approximately 18% of the sepal lengths are between 5.0 and 5.5.

It would be incorrect to choose `type="count"` or `type="proportion"` since this distorts the picture of the data. Fortunately, R will warn you if you try:

hist-unequal-bins-bad-echo

```
histogram(~Sepal.Length, data = iris, breaks = c(4, 5, 5.5, 5.75, 6, 6.5, 7, 8, 9), type = "count")
```

```
Warning:  type='count' can be misleading in this context
```



Notice how different this looks. Now the heights are equal to the relative frequency, but this makes the wider bars have too much area.

 ©2014

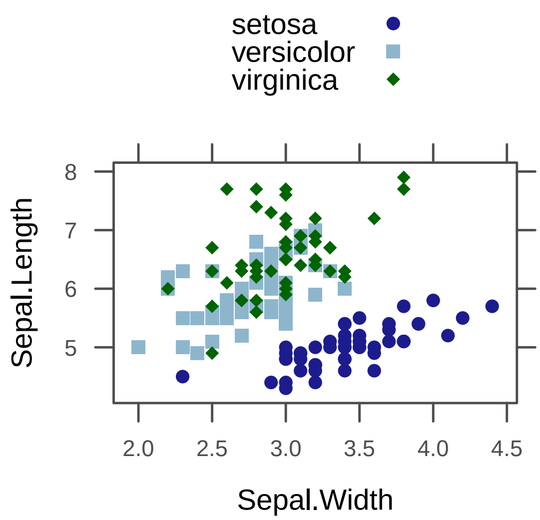
More on Plots

There are lots of arguments that control how these plots look. Here are just a few examples, some of which we have already seen.

auto.key

`auto.key=TRUE` turns on a simple legend for the grouping variable. (There are ways to have more control, if you need it.)

iris-xyplot-key
```
xyplot(Sepal.Length ~ Sepal.Width, groups = Species, data = iris, auto.key = TRUE)
```

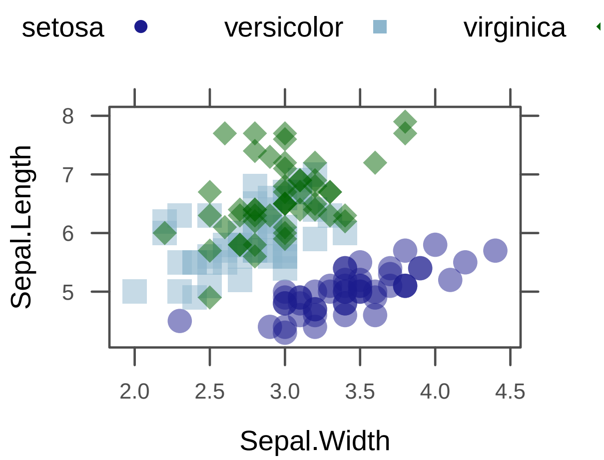


alpha, cex

Sometimes it is nice to have elements of a plot be partly transparent. When such elements overlap, they get darker, showing us where data are "piling up." Setting the `alpha` argument to a value between 0 and 1 controls the degree of transparency: 1 is completely opaque, 0 is invisible. The `cex` argument controls "character expansion" and can be used to make the plotting "characters" larger or smaller by specifying the scaling ratio.

Here is another example using data on 150 iris plants of three species.

iris-xyplot-alpha
```
xyplot(Sepal.Length ~ Sepal.Width, groups = Species, data = iris, auto.key = list(columns = 3),
    alpha = 0.5, cex = 1.3)
```

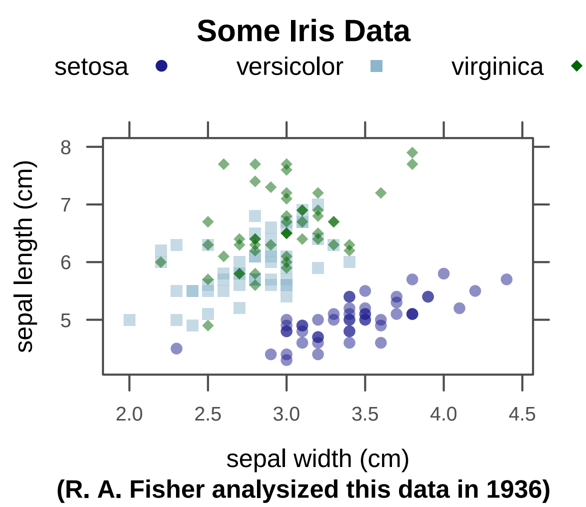


main, sub, xlab, ylab

You can add a title or subtitle, or change the default labels of the axes.

iris-xyplot-text

```
xyplot(Sepal.Length ~ Sepal.Width, groups = Species, data = iris, main = "Some Iris Data",
    sub = "(R. A. Fisher analysized this data in 1936)", xlab = "sepal width (cm)", ylab = "sepal length (cm)",
    alpha = 0.5, auto.key = list(columns = 3))
```



layout

`layout` can be used to control the arrangement of panels in a multi-panel plot. The format is

```
layout = c(cols, rows)
```

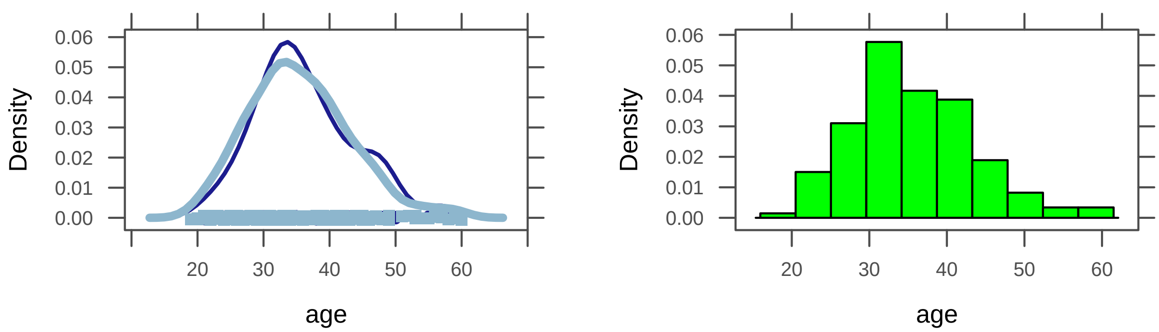
where `cols` is the number of columns and `rows` is the number of rows. (Columns first because that is the *x*-coordinate of the plot.)

lty, lwd, pch, col

These can be used to change the line type, line width, plot character, and color. To specify multiples (one for each group), use the `c()` function (see below).

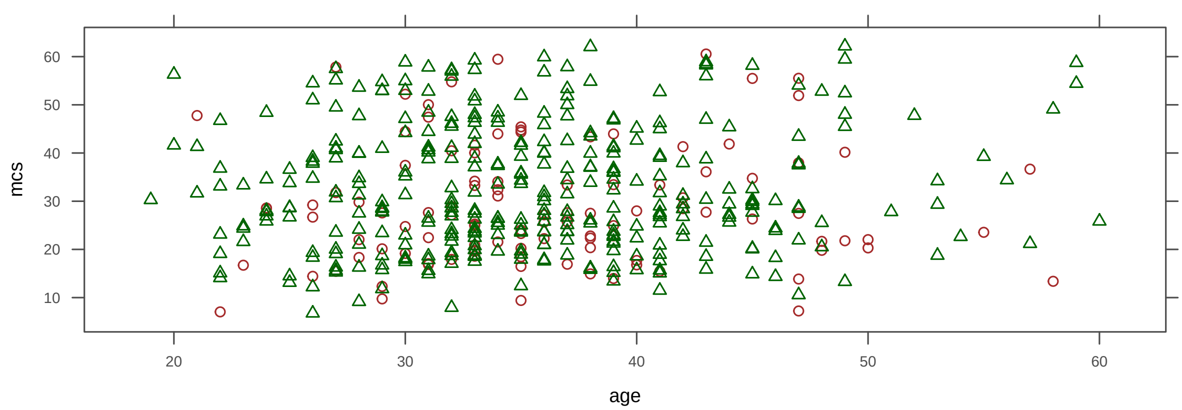
pch-lwd-lty

```
densityplot(~age, data = HELPrct, groups = sex, lty = 1, lwd = c(2, 4))
histogram(~age, data = HELPrct, col = "green")
```



col

```
# There are 25 numbered plot symbols; pch=plot character
xyplot( mcs ~ age, data=HELPrct, groups=sex,
        pch=c(1,2), col=c('brown', 'darkgreen'), cex=.75 )
```



Note: If you change this this way, they will *not* match what is generated in the legend using `auto.key=TRUE`. So it can be better to set these things in a different way if you are using `groups`. See below.

You can a list of the hundreds of available color names using

colors

```
colors()
```

trellis.par.set()

Default settings for lattice graphics are set using `trellis.par.set()`. Don't like the default font sizes? You can change to a 7 point (base) font using

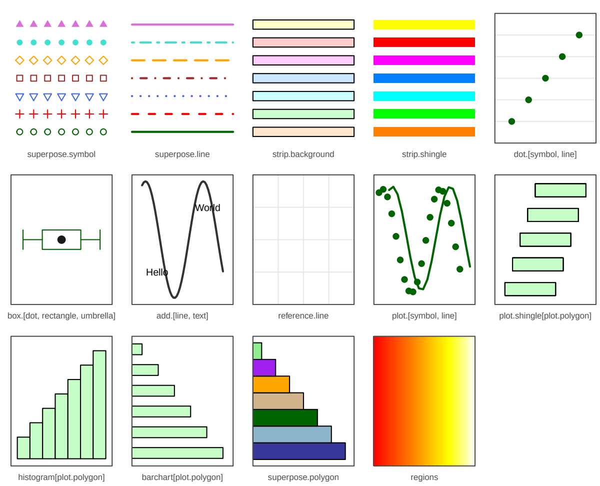
fontsize

```
trellis.par.set(fontsize = list(text = 7))  # base size for text is 7 point
```

Nearly every feature of a lattice plot can be controlled: fonts, colors, symbols, line thicknesses, colors, etc. Rather than describe them all here, we'll mention only that groups of these settings can be collected into a theme. `show.settings()` will show you what the theme looks like.

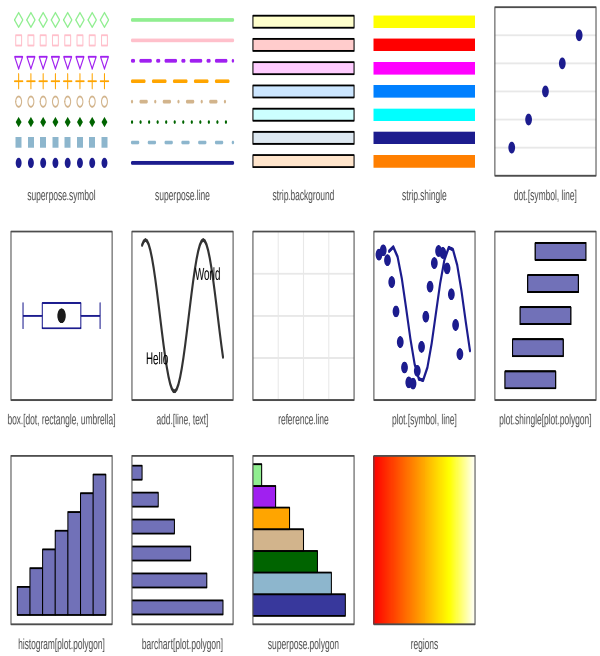
themes-whitbg

```
trellis.par.set(theme = col.whitebg())  # a theme in the lattice package
show.settings()
```



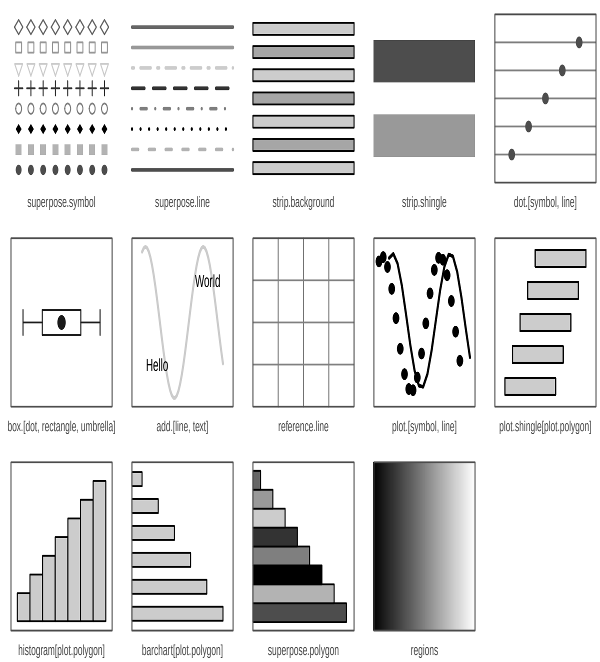
themes-mosaic

```
trellis.par.set(theme = col.mosaic())  # a theme in the mosaic package
show.settings()
```



themes-mosaic2

```
trellis.par.set(theme = col.mosaic(bw = TRUE))  # black and white version
show.settings()
```



themes-mosaic-redo

```
trellis.par.set(theme = col.mosaic())  # back to the mosaic theme
trellis.par.set(fontsize = list(text = 9))  # and back to a 9 point font
```

Want to save your settings?

save-settings

```
# save current settings
mySettings <- trellis.par.get()
# switch to mosaic defaults
trellis.par.set(theme = col.mosaic())
# switch back to my saved settings
trellis.par.set(mySettings)
```

Exporting Plots

You can save plots to files or copy them to the clipboard using the Export menu in the Plots tab. It is quite simple to copy the plots to the clipboard and then paste them into a Word document, for example. You can even adjust the height and width of the plot first to get it the shape you want.

R code and output can be copied and pasted as well. It's best to use a fixed width font (like Courier) for R code so that things align properly.

RStudio also provides a way (actually multiple ways) to create documents that include text, R code, R output, and graphics all in one document so you don't have to do any copying and pasting. This is a much better workflow since it avoids copy-and-paste which is error prone and makes it easy to regenerate an entire report should the data change (because you get more of it or correct an error, for example).

 ©2014

Exercises

For problems 1–7, include both the plots and the code you used to make them as well as any required discussion. Once you get the plots figured out, feel free to use some of the bells and whistles to make the plots even better.

1 Use R's help system to find out what the `i1` and `i2` variables are in the `HELPrct` data frame. Make histograms for each variable and comment on what you find out. How would you describe the shape of these distributions? Do you see any outliers (observations that don't seem to fit the pattern of the rest of the data)?

2 Compare the distributions of `i1` and `i2` among men and women.

3 Compare the distributions of `i1` and `i2` among the three `substance` groups.

4 Where do the data in the `CPS85` data frame (in the `mosaic` package) come from? What are the observational units? How many are there?

5 Choose a quantitative variable that interests you in the `CPS85` data set. Make an appropriate plot and comment on what you see.

6 Choose a categorical variable that interests you in the `CPS85` data set. Make an appropriate plot and comment on what you see.

7 Create a plot that displays two or more variables from the `CPS85` data. At least one should be quantitative and at least one should be categorical. Comment on what you can learn from your plot.

8 The `fusion2` data set in the `fastR` package contains genotypes for another SNP. Merge `fusion1`, `fusion2`, and `pheno` into a single data frame.

Note that `fusion1` and `fusion2` have the same columns.

```
head(fusion1, 2)

     id     marker markerID allele1 allele2 genotype Adose Cdose Gdose Tdose
1  9735 RS12255372        1       3       3       GG     0     0     2     0
2 10158 RS12255372        1       3       3       GG     0     0     2     0

head(fusion2, 2)

     id    marker markerID allele1 allele2 genotype Adose Cdose Gdose Tdose
1  9735 RS7903146        2       2       2       CC     0     2     0     0
2 10158 RS7903146        2       2       2       CC     0     2     0     0
```

You may want to use the `suffixes` argument to `merge()` or rename the variables after you are done merging to make the resulting data frame easier to navigate.

Tidy up your data frame by dropping any columns that are redundant or that you just don't want to have in your final data frame.

9

 ©2014

3 Confidence Intervals

3.1 Sampling Distributions

The key idea in this chapter is the notion of a sampling distribution. Do not confuse it with the population (what we would like to know about) or the sample (what we actually have data about). If we could repeatedly sample from a population, and if we computed a statistic from each sample, the distribution of those statistics would be the sampling distribution. Sampling distributions tell us how things vary from sample to sample and are the key to interpreting data.

Population Parameters and Sample Statistics

Example 3.4

```
head(StatisticsPhD)
```

Example3.4

```
                       University    Department FTGradEnrollment
1               Baylor University    Statistics               26
2               Boston University Biostatistics               39
3                Brown University Biostatistics               21
4      Carnegie Mellon University    Statistics               39
5 Case Western Reserve University    Statistics               11
6       Colorado State University    Statistics               14
```

```
mean(~FTGradEnrollment, data = StatisticsPhD)  # mean enrollment in original population
```

```
[1] 53.54
```

Example 3.5

To select a random sample of a certain size in R, we can use the `sample()` function.

©2014

Example3.5

```
sample10 = sample(StatisticsPhD, 10)
sample10
```

```
                            University    Department FTGradEnrollment orig.ids
25      North Carolina State University    Statistics              163       25
57               University of Kentucky    Statistics               40       57
18             Johns Hopkins University Biostatistics               41       18
29            Oklahoma State University    Statistics               22       29
79     Virginia Commonwealth University    Statistics               15       79
75             University of Washington    Statistics               53       75
49                University of Chicago    Statistics              109       49
2                     Boston University Biostatistics               39        2
22 Medical University of South Carolina Biostatistics               46       22
9                    Cornell University    Statistics               78        9
```

```
x.bar = mean(~FTGradEnrollment, data = sample10)
x.bar  # mean enrollment in sample10
```

```
[1] 60.6
```

Note that this sample has been assigned a name to which we can refer back to find the mean of that particular sample.

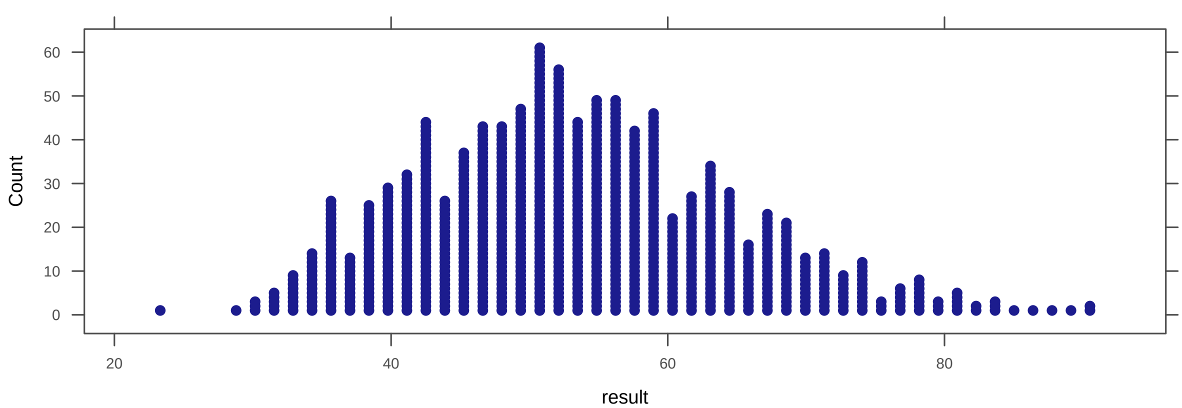
```
mean(~FTGradEnrollment, data = sample(StatisticsPhD, 10))  # mean enrollment in another sample
```

```
[1] 43.3
```

```
# Now we'll do it 1000 times
sampledist <- do(1000) * mean(~FTGradEnrollment, data = sample(StatisticsPhD, 10))
```

We should check that that our sample distribution has an appropriate shape:

```
dotPlot(~result, data = sampledist, n = 50)
```



In many (but not all) situations, the sampling distribution is

 ©2014

- unimodal,
- symmetric, and
- bell-shaped

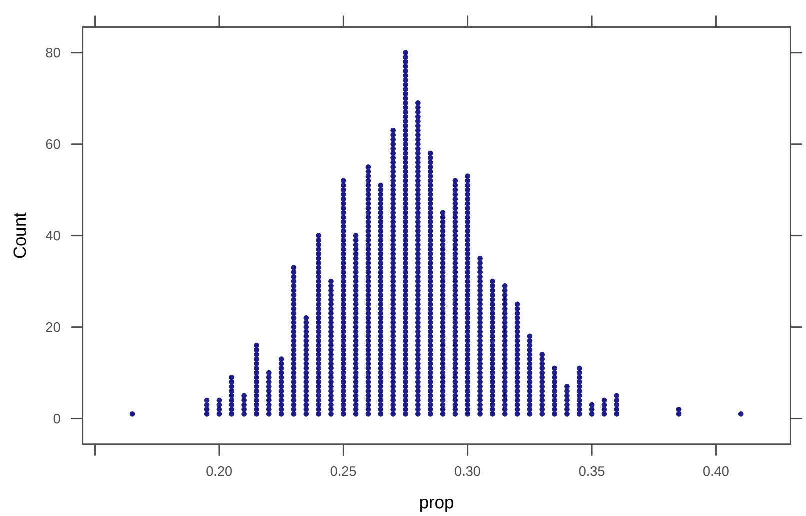
(The technical phrase is "approximately normal".) In these situations, a 95% confidence interval can be estimated with

$$\text{statistic} \pm 2SE$$

Example 3.6

This time we don't have data, but instead we have a summary of the data. We can however, still simulate the sample distribution by using the `rflip()` function.

```
sampledistdeg <- do(1000) * rflip(200, 0.275)  # 1000 samples, each of size 200 and proportion 0.275
head(sampledistdeg, 3)


    n heads tails  prop
1 200    39   161 0.195
2 200    62   138 0.310
3 200    57   143 0.285


dotPlot(~prop, width = 0.005, data = sampledistdeg)
```



Measuring Sampling Variability: The Standard Error

> The standard deviation of a sampling distribution is called the **standard error**, denoted SE.

The standard error is our primary way of measuring how much variability there is from sample statistic to sample statistic, and therefore how precise our estimates are.

Example 3.7

Calculating the SE is the same as calculating the standard deviation of a sampling distribution, so we use `sd()`.

standarderror

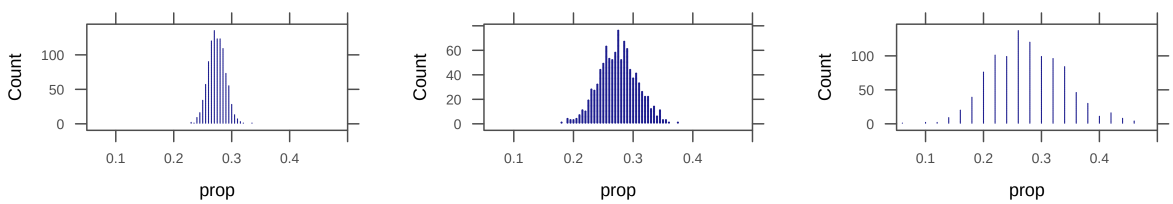
```
SE <- sd(~result, data = sampledist)
SE  # Bootstrap from Example 3.5

[1] 11.13

SE2 <- sd(~prop, data = sampledistdeg)
SE2  # Boot.Deg from Example 3.6

[1] 0.03255
```

The Importance of Sample Size

Example 3.9

bootstrapsamplesize

```
sampledist.1000 <- do(1000) * rflip(1000, 0.275)  # 1000 samples, each of size 1000 and proportion 0.275
sampledist.200 <- do(1000) * rflip(200, 0.275)  # 1000 samples, each of size 200 and proportion 0.275
sampledist.50 <- do(1000) * rflip(50, 0.275)  # 1000 samples, each of size 50 and proportion 0.275
dotPlot(~prop, width = 0.005, xlim = c(0.05, 0.5), data = sampledist.1000)
dotPlot(~prop, width = 0.005, xlim = c(0.05, 0.5), data = sampledist.200)
dotPlot(~prop, width = 0.005, xlim = c(0.05, 0.5), data = sampledist.50)
```



3.2 Understanding and Interpreting Confidence Intervals

Interval Estimates and Margin of Error

An **interval estimate** gives a range of plausible values for a population parameter.

This is better than a single number (also called a point estimate) because it gives some indication of the precision of the estimate.

One way to express an interval estimate is with a point estimate and a **margin of error**.

 ©2014

We can convert margin of error into an interval by adding and subtracting the margin of error to/from the statistic.

Example 3.12

$$0.42 \pm 0.03 \text{ which is the same as } (0.39, 0.45)$$

Example 3.13

```
p.hat = 0.54                    # sample proportion                    marginoferror
MoE = 0.02                      # margin of error
p.hat - MoE                     # lower limit of interval estimate

[1] 0.52

p.hat + MoE                     # upper limit of interval estimate

[1] 0.56
```

```
p.hat = 0.54
MoE = 0.1
p.hat - MoE

[1] 0.44

p.hat + MoE

[1] 0.64
```

Confidence Intervals

> A confidence interval for a parameter is an interval computed from sample data by a method that will capture the parameter for a specified proportion of all samples

1. The probability of correctly containing the parameter is called the coverage rate or **confidence level**.
2. So 95% of 95% confidence intervals contain the parameter being estimated.
3. The margins of error in the tables above were designed to produce 95% confidence intervals.

Example 3.14

```
x.bar = 61.5            # given sample mean
SE = 11                 # given estimated standard error
MoE = 2 * SE; MoE       # margin of error for 95% CI
```

confidenceinterval

```
[1] 22
```

```
x.bar - MoE             # lower limit of 95% CI
```

```
[1] 39.5
```

```
x.bar + MoE             # upper limit of 95% CI
```

```
[1] 83.5
```

Understanding Confidence Intervals

Example 3.15

```
SE = 0.03
p1 = 0.26
p2 = 0.32
p3 = 0.2
MoE = 2 * SE
```

confidenceinterval2

```
p1 - MoE
```

```
[1] 0.2
```

```
p1 + MoE
```

```
[1] 0.32
```

```
p2 - MoE
```

```
[1] 0.26
```

```
p2 + MoE
```

```
[1] 0.38
```

```
p3 - MoE
```

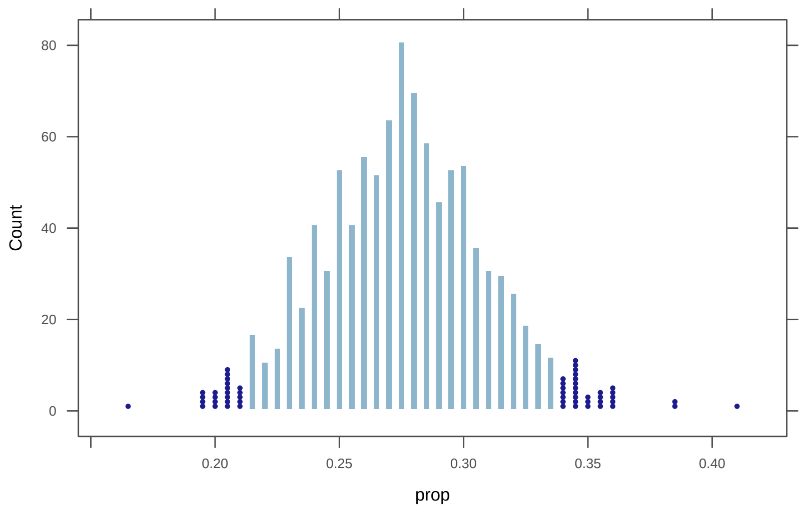
```
[1] 0.14
```

```
p3 + MoE
```

```
[1] 0.26
```


©2014

```
p = 0.275
SE = 0.03
MoE = 2 * SE
p - MoE

[1] 0.215

p + MoE

[1] 0.335

dotPlot(~prop, width = 0.005, groups = (0.215 <= prop & prop <= 0.335), data = sampledistdeg)
```



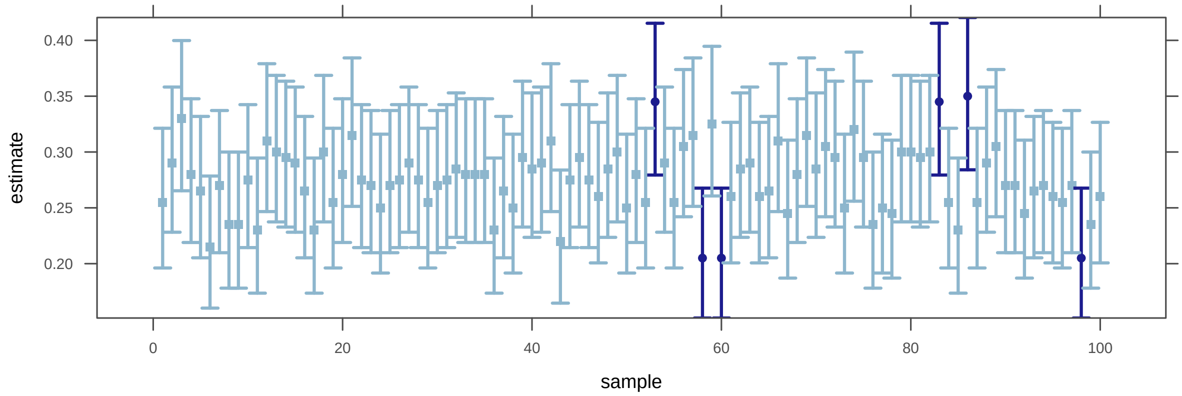
Interpreting Confidence Intervals

Figure 3.13

We can create the data needed for plots like Figure 3.13 using `CIsim()`. The plot itself uses `xYplot()` from the `Hmisc` package.

```
results <- CIsim(200, samples = 100, rdist = rbinom, args = list(size = 1, prob = 0.275), method = binom.test,
    method.args = list(success = 1), verbose = FALSE, estimand = 0.275)
require(Hmisc)
xYplot(Cbind(estimate, lower, upper) ~ sample, data = results, par.settings = col.mosaic(),
    groups = cover)
```

©2014



Example 3.16

```
x.bar = 27.655
SE = 0.009
MoE = 2 * SE
x.bar - MoE


[1] 27.64


x.bar + MoE


[1] 27.67
```

Example 3.17

```
diff.x = -1.915
SE = 0.016
MoE = 2 * SE
diff.x - MoE


[1] -1.947


diff.x + MoE


[1] -1.883
```

3.3 Constructing Bootstrap Confidence Intervals

Here's the clever idea: We don't have the population, but we have a sample. Probably the sample it similar to the population in many ways. So let's sample from our sample. We'll call it **resampling** (also called **bootstrapping**). We want samples the same size as our original sample, so we will need to sample with replacement.

 ©2014

This means that we may pick some members of the population more than once and others not at all. We'll do this many times, however, so each member of our sample will get its fair share. (Notice the similarity to and difference from sampling from populations in the previous sections.)

Commuting in Atlanta

```
head(CommuteAtlanta, 3)
```

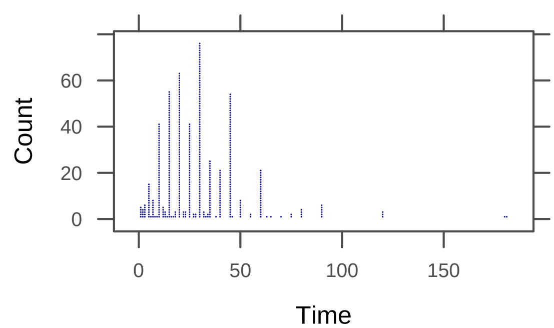
```
     City Age Distance Time Sex
1 Atlanta  19       10   15   M
2 Atlanta  55       45   60   M
3 Atlanta  48       12   45   M
```

```
dotPlot(~Time, width = 1, cex = 0.5, data = CommuteAtlanta)
mean(~Time, data = CommuteAtlanta)
```

```
[1] 29.11
```

```
sd(~Time, data = CommuteAtlanta)
```

```
[1] 20.72
```



Bootstrap Samples

The computer can easily do all of the resampling by using the `resample()`.

```
resample(CommuteAtlanta, 10)
```

```
       City Age Distance Time Sex orig.ids
18  Atlanta  43       25   45   M       18
439 Atlanta  44       28   30   M      439
368 Atlanta  55       10   20   F      368
311 Atlanta  62       45   45   M      311
349 Atlanta  39       12   25   F      349
290 Atlanta  35        6   16   F      290
279 Atlanta  40       30   30   F      279
```

```
275 Atlanta  30        10   30   M       275
464 Atlanta  45        10   15   F       464
138 Atlanta  41         0   10   M       138
```

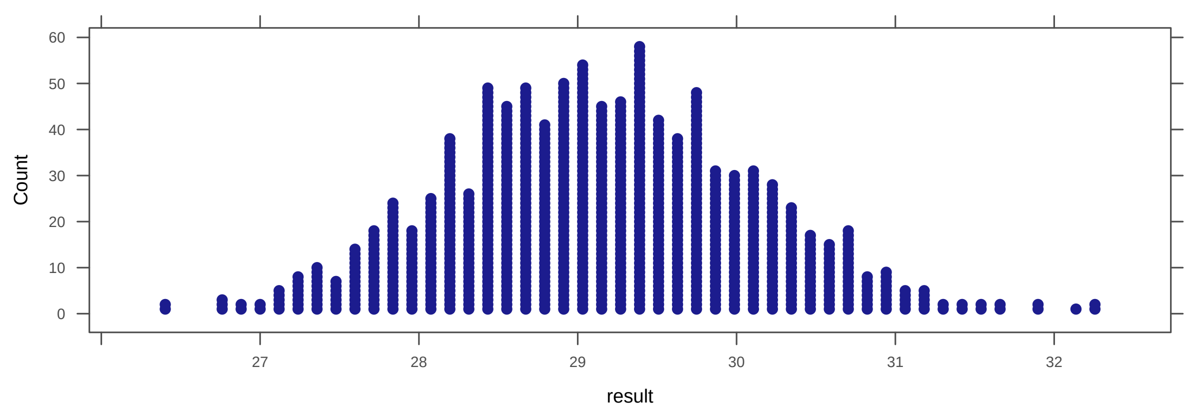
Bootstrap Distribution

The example below uses data from 500 Atlanta commuters.

```
mean(~Time, data = resample(CommuteAtlanta))  # mean commute time in one resample

[1] 30.31

mean(~Time, data = resample(CommuteAtlanta))  # mean commute time in another resample

[1] 30.83

# Now we'll do it 1000 times
Bootstrap <- do(1000) * mean(~Time, data = resample(CommuteAtlanta))
```

We should check that that our bootstrap distribution has an appropriate shape:

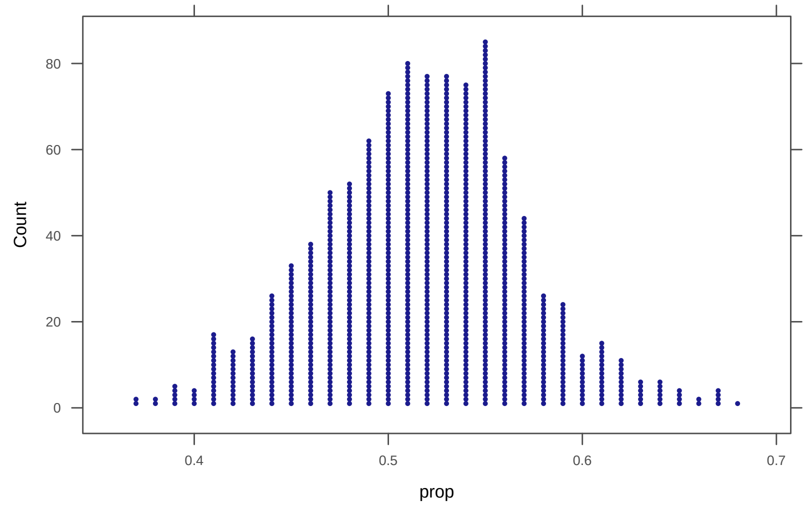
```
dotPlot(~result, data = Bootstrap, n = 50)
```



Example 3.19

```
BootP <- do(1000) * rflip(100, 0.52)
head(BootP, 3)

    n heads tails prop
1 100    49    51 0.49
2 100    47    53 0.47
3 100    52    48 0.52

dotPlot(~prop, width = 0.01, data = BootP)
```


©2014



Example 3.20

Variables can be created in R using the `c()` function then collected into a data frame using the `data.frame()` function.

```
Laughter <- data.frame(NumLaughs = c(16, 22, 9, 31, 6, 42))
mean(~NumLaughs, data = Laughter)


[1] 21
```

```
Boot.L1 <- do(1000) * mean(~NumLaughs, data = resample(Laughter))
mean(~result, data = Boot.L1)


[1] 20.35


Boot.L2 <- do(1000) * mean(~NumLaughs, data = resample(Laughter))
mean(~result, data = Boot.L2)


[1] 20.83


Boot.L3 <- do(1000) * mean(~NumLaughs, data = resample(Laughter))
mean(~result, data = Boot.L3)


[1] 21.14
```

Estimating Standard Error Based on a Bootstrap Distribution

Example 3.21

Since the shape of the bootstrap distribution from Example 3.19 looks good, we can estimate the standard error.

```
SE = sd(~prop, data = BootP)
SE

[1] 0.05168
```

Example 3.22

We can again use the standard error to compute a 95% confidence interval.

```
x.bar <- mean(~Time, data = CommuteAtlanta); x.bar

[1] 29.11

SE <- sd(~result, data = Bootstrap ); SE        # standard error

[1] 0.9365

MoE <- 2 * SE; MoE                              # margin of error for 95% CI

[1] 1.873

x.bar - MoE                                     # lower limit of 95% CI

[1] 27.24

x.bar + MoE                                     # upper limit of 95% CI

[1] 30.98
```

```
p.hat = 0.52
SE = sd(~prop, data = BootP)
SE

[1] 0.05168

MoE = 2 * SE
MoE

[1] 0.1034

p.hat - MoE

[1] 0.4166

p.hat + MoE

[1] 0.6234
```

 ©2014

The same steps used in this example, get used in a wide variety of confidence interval situations.

1. Compute the statistic from the original sample.
2. Create a bootstrap distribution by resampling from the sample.
   (a) same size samples as the original sample
   (b) with replacement
   (c) compute the statistic for each sample

   The distribution of these statistics is the bootstrap distribution
3. Estimate the standard error SE by computing the standard deviation of the bootstrap distribution.
4. 95% CI is
$$\text{statistic} \pm 2SE$$

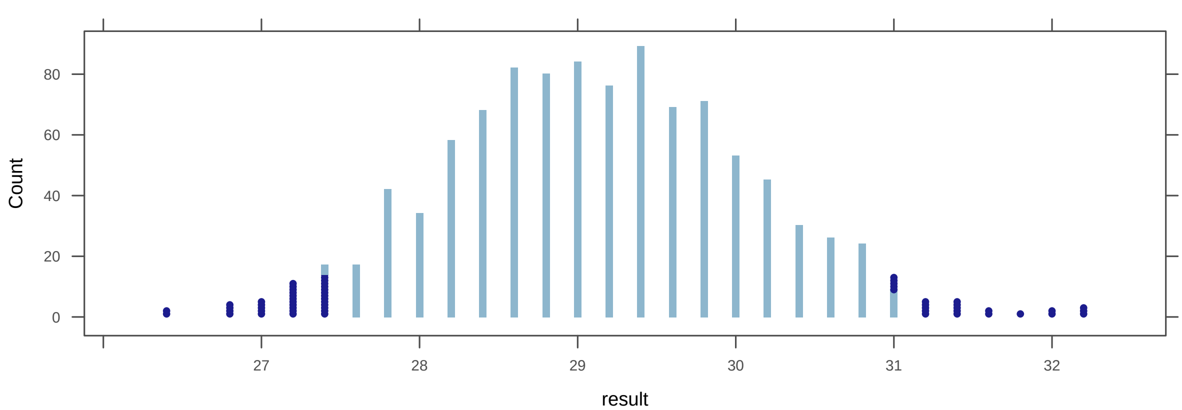
3.4 Bootstrap Confidence Intervals Using Percentiles

Confidence Intervals Based on Bootstrap Percentiles

Example 3.23

Another way to create a 95% confidence interval is to use the middle 95% of the bootstrap distribution. The `cdata()` function can compute this for us as follows:

```
cdata(0.95, result, data = Bootstrap)

      low        hi central.p
    27.37     30.98      0.95

dotPlot(~result, width = 0.2, groups = (27.47 <= result & result <= 31), data = Bootstrap)
```



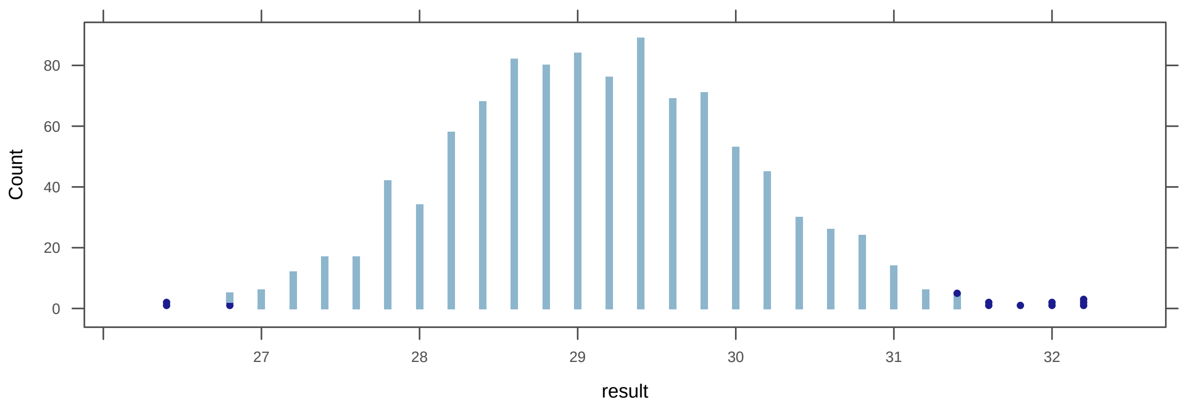
This is not exactly the same as the interval of the original sample, but it is pretty close. Notice the `groups=` for marking the confidence interval.

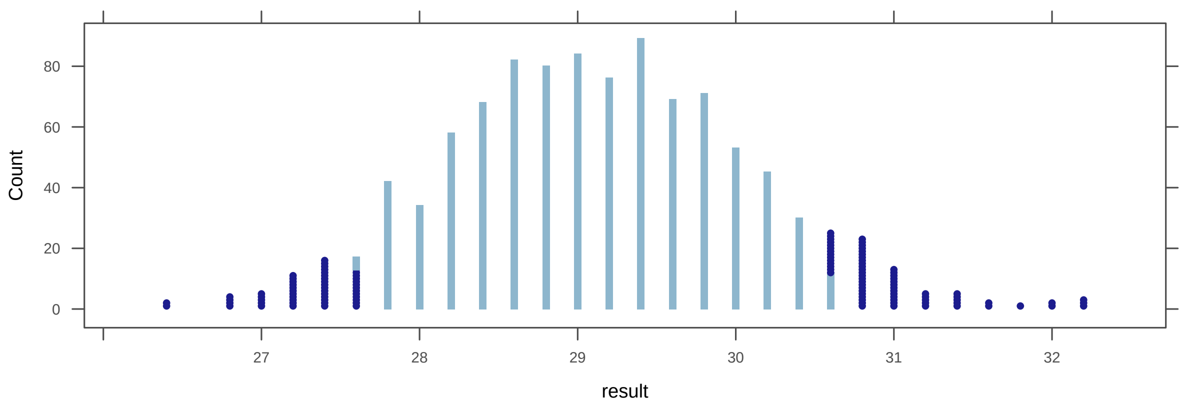
Example 3.24

One advantage of this method is that it is easy to change the confidence level.

To make a 90% confidence interval, we use the middle 90% of the sample distribution instead.

```
cdata(0.99, result, data = Bootstrap)


     low        hi central.p
   26.85     31.70      0.99


dotPlot(~result, width = 0.2, groups = (26.79 <= result & result <= 31.46), data = Bootstrap)
cdata(0.9, result, data = Bootstrap)


     low        hi central.p
   27.63     30.71      0.90


dotPlot(~result, width = 0.2, groups = (27.64 <= result & result <= 30.55), data = Bootstrap)
```





Notice that this interval is narrower. This will always be the case. Higher levels of confidence lead to wider confidence intervals.

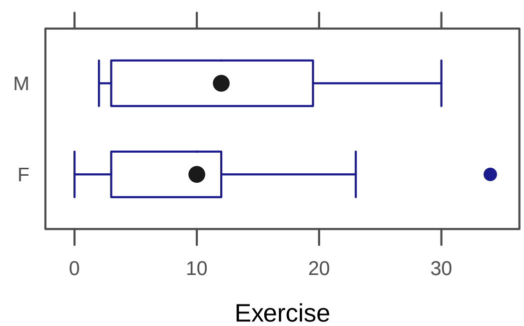
(Method 1 can also be adjusted for other confidence levels as well – the number 2 needs to be replaced by an appropriate alternative.)

 ©2014

Finding Confidence Intervals for Many Different Parameters

Example 3.25

```
head(ExerciseHours)


  Year Gender Hand Exercise TV Pulse Pierces
1    4      M    l       15  5    57       0
2    2      M    l       20 14    70       0
3    3      F    r        2  3    70       2
4    1      F    l       10  5    66       3
5    1      M    r        8  2    62       0
6    1      M    r       14 14    62       0


bwplot(Gender ~ Exercise, data = ExerciseHours)
favstats(~Exercise | Gender, data = ExerciseHours)


  .group min Q1 median    Q3 max mean    sd  n missing
1      F   0  3     10 12.00  34  9.4 7.407 30       0
2      M   2  3     12 19.25  30 12.4 8.798 20       0
```

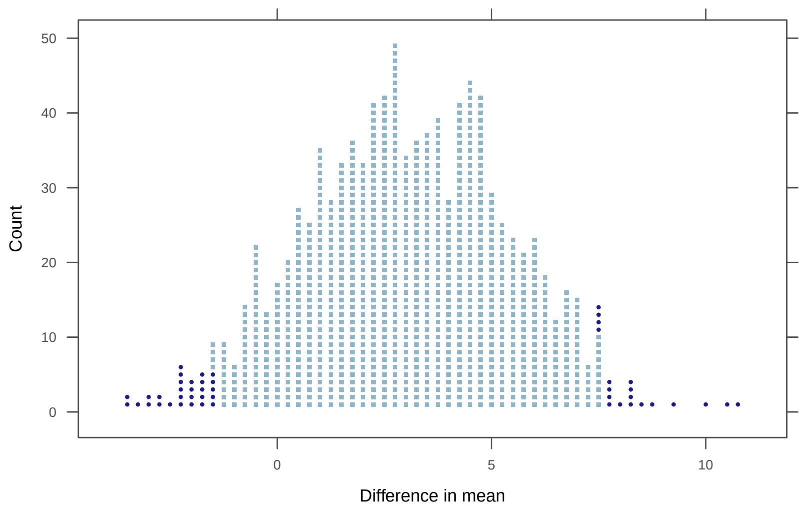



```
stat <- diff(mean(Exercise ~ Gender, data = ExerciseHours))
stat


M
3
```

```
BootE <- do(1000) * diff(mean(Exercise ~ Gender, data = resample(ExerciseHours)))
head(BootE, 3)


      M
1 2.584
2 4.080
3 3.165
```

```
cdata(0.95, M, data = BootE)


      low        hi central.p
   -1.531     7.401     0.950
```

```
dotPlot(~M, width = 0.25, cex = 0.5, groups = (-1.44 <= M & M <= 7.534), xlab = "Difference in mean",
    data = BootE)
```

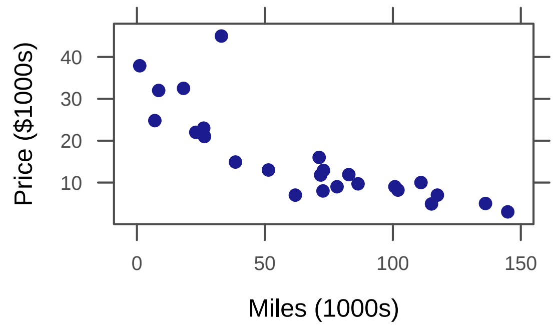



```
SE <- sd(~M, data = BootE)
SE

[1] 2.351

stat - 2 * SE

     M
-1.702

stat + 2 * SE

    M
7.702
```

Example 3.26

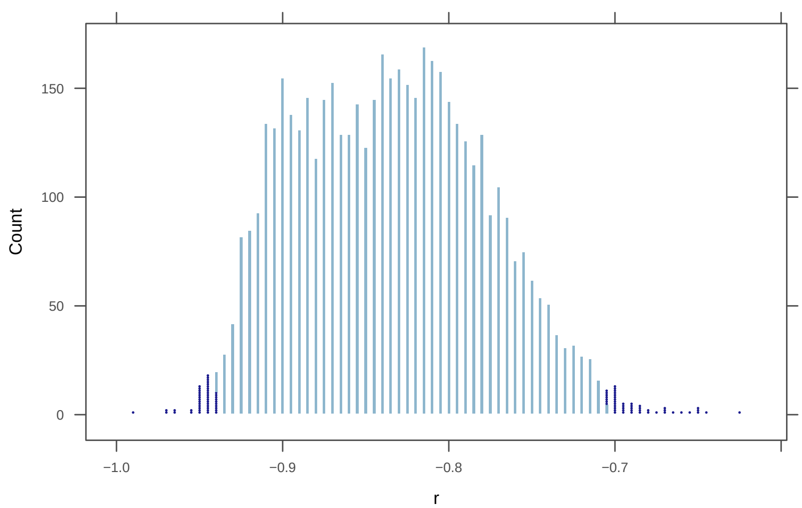
```
head(MustangPrice, 3)

  Age Miles Price
1   6   8.5  32.0
2   7  33.0  45.0
3   9  82.8  11.9

xyplot(Price ~ Miles, ylab = "Price ($1000s)", xlab = "Miles (1000s)", data = MustangPrice)
cor(Price ~ Miles, data = MustangPrice)

[1] -0.8246
```

 ©2014




```
BootM <- do(5000) * cor(Price ~ Miles, data = resample((MustangPrice)))
head(BootM, 3)


   result
1 -0.7622
2 -0.7888
3 -0.8175
```

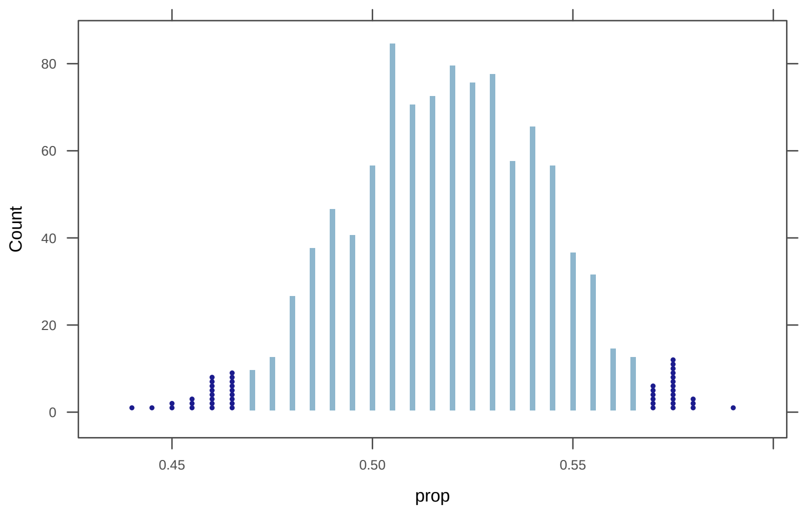
```
cdata(0.98, result, data = BootM)


      low        hi central.p
  -0.9391   -0.7073    0.9800


dotPlot(~result, width = 0.005, groups = (-0.9394 <= result & result <= -0.706), xlab = "r",
    data = BootM)
```



Another Look at the Effect of Sample Size

Example 3.27

```
BootP400 <- do(1000) * rflip(400, 0.52)
head(BootP400, 3)


    n heads tails   prop
1 400   202   198 0.5050
2 400   210   190 0.5250
3 400   207   193 0.5175


cdata(0.95, prop, data = BootP400)


      low        hi central.p
   0.4724    0.5651    0.9500


dotPlot(~prop, width = 0.005, groups = (0.47 <= prop & prop <= 0.5675), data = BootP400)
```

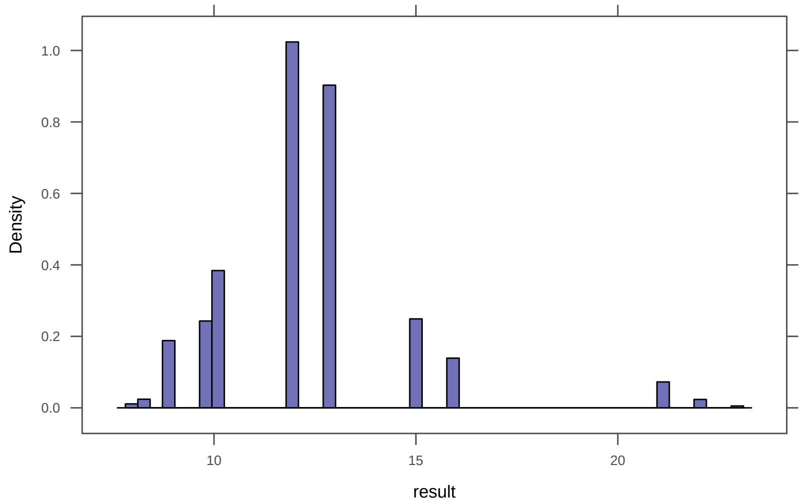


One Caution on Constructing Bootstrap Confidence Intervals

Example 3.28

```
median(~Price, data = MustangPrice)


[1] 11.9


Boot.Mustang <- do(5000) * median(~Price, data = resample(MustangPrice))
head(Boot.Mustang, 3)


  result
1   14.9
2   10.0
3    9.0


histogram(~result, data = Boot.Mustang, n = 50)
```


©2014



This time the histogram does not have the desired shape. There are two problems:

1. The distribution is not symmetric. (It is right skewed.)
2. The distribution has spikes and gaps.

   Since the median must be an element of the sample when the sample size is 25, there are only 25 possible values for the median (and some of these are *very* unlikely.

Since the bootstrap distribution does not look like a normal distribution (bell-shaped, symmetric), we cannot safely use our methods for creating a confidence interval.

4

Hypothesis Tests

4.1 Introducing Hypothesis Tests

The 4-step outline

The following 4-step outline is a useful way to organize the ideas of hypothesis testing.

1. State the Null and Alternative Hypotheses
2. Compute the Test Statistic
   The test statistic is a number that summarizes the evidence
3. Determine the p-value (from the Randomization Distribution)
4. Draw a conclusion

4.2 Measuring Evidence with P-values

Randomization distributions are a bit like bootstrap distributions except that instead of resampling from our sample (in an attempt to approximate resampling from the population), we need to sample from a situation in which our null hypothesis is true.

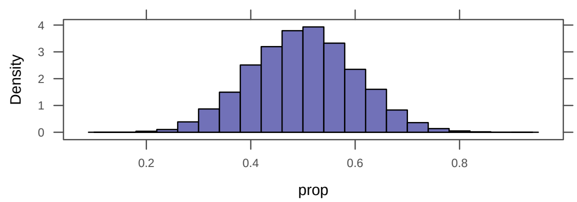
P-values from Randomization Distributions

Example 4.13

Testing one proportion.

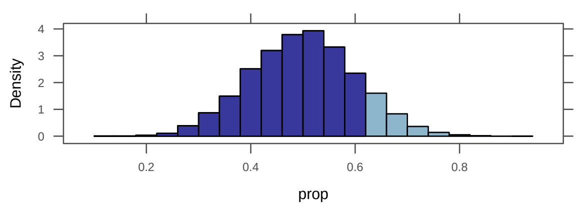
1. H_0: $p = 0.5$; H_a: $p > 0.5$
2. Test statistic: $\hat{p} = 16/25$ (the sample proportion)
3. We can simulate a world in which $p = 0.5$ using `rflip()`:

```
Randomization.Match <- do(10000) * rflip(25, 0.5)  # 25 because n=25
head(Randomization.Match)

   n heads tails prop
1 25     9    16 0.36
2 25    15    10 0.60
3 25    13    12 0.52
4 25    14    11 0.56
5 25    14    11 0.56
6 25    13    12 0.52

histogram(~prop, width = 0.04, data = Randomization.Match)
```



Here we find the proportion of the simulations which resulted in 16 or more matches out of 25, or 0.64 or greater, for the p-value.

```
prop(~(prop >= 0.64), data = Randomization.Match)  # 16/25

TRUE
0.12

histogram(~prop, width = 0.04, groups = (prop >= 0.64), data = Randomization.Match)
```

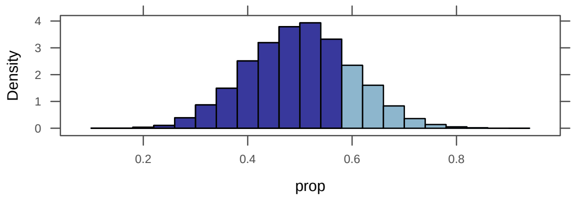


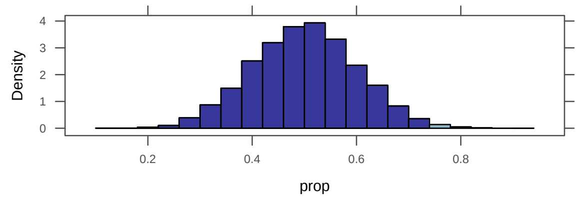
Example 4.15

```
prop(~(prop >= 0.6), data = Randomization.Match)  # 15/25

  TRUE
0.2139

prop(~(prop >= 0.76), data = Randomization.Match)  # 19/25

  TRUE
0.0083

histogram(~prop, width = 0.04, groups = (prop >= 0.6), data = Randomization.Match)
histogram(~prop, width = 0.04, groups = (prop >= 0.76), data = Randomization.Match)
```

Example4.15

 ©2014





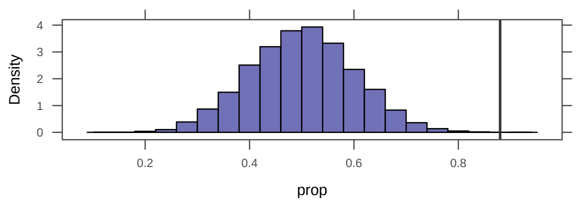
Example 4.16

```
prop(~(prop >= 0.88), data = Randomization.Match)  # 22/25
```

Example4.16

```
 TRUE
1e-04
```

```
histogram(~prop, width = 0.04, v = c(0.88), data = Randomization.Match)
```



Example 4.18

Testing two means.

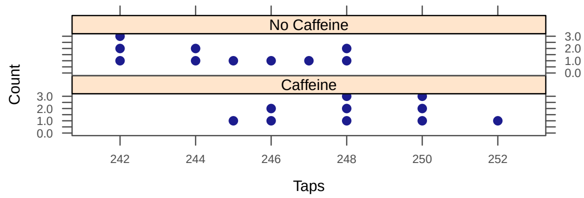
```
mean(Taps ~ Group, data = CaffeineTaps)
```

```
   Caffeine No Caffeine
      248.3       244.8
```

```
diff(mean(Taps ~ Group, data = CaffeineTaps))
```

```
No Caffeine
       -3.5
```

```
dotPlot(~Taps | Group, layout = c(1, 2), width = 1, cex = 0.1, data = CaffeineTaps)
```



1. H_0: $\mu_1 = \mu_2$; H_a: $\mu_1 > \mu_2$
2. Test statistic: $\bar{x}_1 - \bar{x}_2 = 3.5$ (the difference in sample means)
3. We simulate a world in which $\mu_1 = \mu_2$ or $\mu_1 - \mu_2 = 0$:

```
Randomization.Caff <- do (1000) * diff( mean( Taps ~ shuffle(Group), data=CaffeineTaps ) )
head(Randomization.Caff,3)
histogram(~No Caffeine, data=Randomization.Caff)
```

```
prop( ~ (No Caffeine > 3.5), data=Randomization.Temp )
```

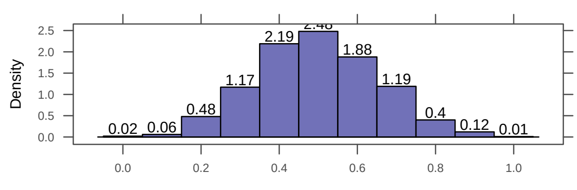
P-values and the Alternative Hypothesis

Example 4.19

Testing one proportion.

1. H_0: $p = 0.5$; H_a: $p > 0.5$
2. Test statistic: $\hat{p}$ (the sample proportion)
3. We simulate a world in which $p = 0.5$:

```
RandomizationDist <- do(1000) * rflip(10, 0.5)  # 10 because n=10
head(RandomizationDist)

   n heads tails prop
1 10     2     8  0.2
2 10     2     8  0.2
3 10     2     8  0.2
4 10     7     3  0.7
5 10     4     6  0.4
6 10     5     5  0.5

histogram(~prop, label = TRUE, data = RandomizationDist)
```



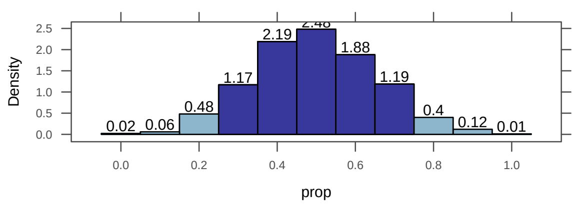
```
Example4.19
prop(~(prop >= 0.8), data = RandomizationDist)

 TRUE
0.053

prop(~(prop >= 0.6), data = RandomizationDist)

TRUE
0.36

prop(~(prop >= 0.4), data = RandomizationDist)

 TRUE
0.827
```

Example 4.20

Testing one proportion.

1. H_0: $p = 0.5$; H_a: $p \neq 0.5$
2. Test statistic: $\hat{p} = 0.8$ (the sample proportion)
3. We simulate a world in which $p = 0.5$:

```
Example4.20
prop(~(prop >= 0.8), data = RandomizationDist)

 TRUE
0.053

prop(~(prop <= 0.2), data = RandomizationDist)

 TRUE
0.056

histogram(~prop, label = TRUE, groups = (prop <= 0.2 | prop >= 0.8), data = RandomizationDist)
```




```
# a 2-sided p-value is the sum of the values above
prop(~(prop <= 0.2 | prop >= 0.8), data = RandomizationDist)

 TRUE
0.109
```

```
# We can also approximate the p-value by doubling one side
2 * prop(~prop >= 0.8, data = RandomizationDist)

 TRUE
0.106
```

4.3 Determining Statisical Significance

Less Formal Statistical Decisions

Example 4.27

Testing two means.

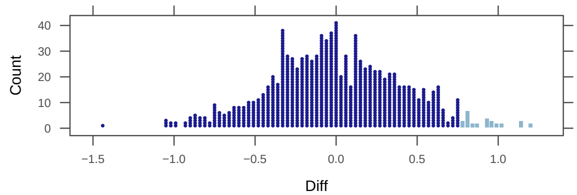
Example4.27

```
head(Smiles)

  Leniency Group
1      7.0 smile
2      3.0 smile
3      6.0 smile
4      4.5 smile
5      3.5 smile
6      4.0 smile

mean(Leniency ~ Group, data = Smiles)

neutral   smile
  4.118   4.912

diff(mean(Leniency ~ Group, data = Smiles))

 smile
0.7941
```

1. H_0: $\mu_1 = \mu_2$; H_a: $\mu_1 \neq \mu_2$
2. Test statistic: $\bar{x}_1 - \bar{x}_2 = 0.79$ (the difference in sample means)
3. We simulate a world in which $\mu_1 = \mu_2$:

```
Randomization.Smiles <- do(1000) * diff(mean(Leniency ~ shuffle(Group), data = Smiles))
head(Randomization.Smiles, 3)

    smile
1  0.1765
2 -0.1471
3 -0.1176

dotPlot(~smile, width = 0.03, cex = 0.5, groups = (smile >= 0.79), xlab = "Diff", data = Randomization.Smiles)
```

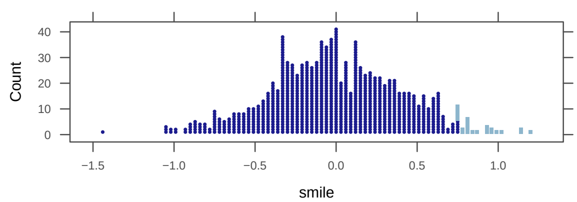

©2014




```
prop(~(smile <= -0.79 | smile >= 0.79), data = Randomization.Smiles)

 TRUE
0.049

2 * prop(~smile >= 0.79, data = Randomization.Smiles)

TRUE
0.04
```

Now we find the p-value to test a difference of 0.76:

```
prop(~(smile <= -0.76 | smile >= 0.76), data = Randomization.Smiles)

TRUE
0.06

2 * prop(~smile >= 0.76, data = Randomization.Smiles)

 TRUE
0.052

dotPlot(~smile, width = 0.03, cex = 0.5, groups = (smile >= 0.76), data = Randomization.Smiles)
```



4.4 Creating Randomization Distributions

In order to use these methods to estimate a p-value, we must be able to generate a randomization distribution. In the case of a test with null hypothesis claiming that a proportion has a particular value (e.g, H_O: $p = 0.5$), this is pretty easy. If the population has proportion 0.50, we can simulate sampling from that proportion by flipping a fair coin. If the proportion is some value other than 0.50, we simply flip a coin that has the appropriate probability of resulting in heads. So the general template for creating such a randomization distribution is

```
do(1000) * rflip(n, hypothesized_proportion)
```

©2014

where n is the size of the original sample.

In other situations, it can be more challenging to create a randomization distribution because the null hypothesis does not directly specify all of the information needed to simulate samples.

- H_O: $p_1 = p_2$

  This would be simple *if* we new the value of p_1 and p_2 (we could use `rflip()` twice, once for each group),

- H_O: μ = some number

  Just knowing the mean does not tell us enough about the distribution. We need to know about its shape. (We might need to know the standard deviation, for example, or whether the distribution is skewed.)

- H_O: $\mu_1 \neq \mu_2$ some number.

  Now we don't know the common mean and we don't know the things mentioned in the previous example either.

So how do we come up with randomization distribution?

The main criteria to consider when creating randomization samples for a statistical test are:

- Be consistent with the null hypothesis.

  If we don't do this, we won't be testing our null hypothesis.

- Use the data in the original sample.

  With luck, the original data will shed light on some aspects of the distribution that are not determined by null hypothesis.

- Reflect the way the original data were collected.

Randomization Test for a Difference in Proportions: Cocaine Addiction

Creating some data

Data 4.7 in the text describes some data that are not in a data frame. This often happens when a data set has only categorical variables because a simple table completely describes the distributions involved. Here's the table from the book:[1]

	Relapse	No Relapse
Lithium	18	6
Placebo	20	4

Here's one way to create the data in R:

cocaine-data

```
Cocaine <- rbind(
  do(18) * data.frame( treatment = "Lithium",     response="Relapse"),
  do(6)  * data.frame( treatment = "Lithium",     response="No Relapse"),
```

[1]The book includes data on an additional treatment group which we are omitting here.

©2014

```
  do(20) * data.frame( treatment = "Placebo",     response="Relapse"),
  do(4)  * data.frame( treatment = "Placebo",     response="No Relapse")
  )
```

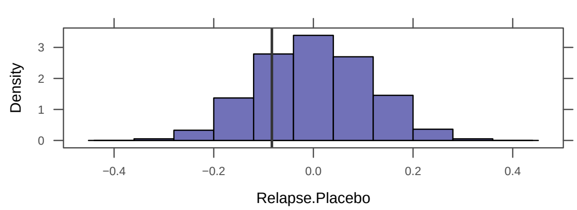
Example 4.29

Testing two proportions.

```
tally(response ~ treatment, data = Cocaine)


            treatment
response     Lithium Placebo
  Relapse      0.7500  0.8333
  No Relapse   0.2500  0.1667


prop(response ~ treatment, data = Cocaine)


Relapse.Lithium Relapse.Placebo
         0.7500          0.8333


diff(prop(response ~ treatment, data = Cocaine))


Relapse.Placebo
        0.08333
```

1. H_0: $p_1 = p_2$; H_a: $p_1 < p_2$
2. Test statistic: $\hat{p}_1 = \hat{p}_2$ (the difference in sample proportions)
3. We simulate a world in which $p_1 = p_2$ or $p_1 - p_2 = 0$:

```
Randomization.Coc <- do(5000) * diff(prop(response ~ shuffle(treatment), data = Cocaine))
head(Randomization.Coc)

  Relapse.Placebo
1         0.16667
2         0.08333
3        -0.25000
4         0.00000
5        -0.08333
6         0.08333

histogram(~Relapse.Placebo, data = Randomization.Coc, v = c(-0.0833), width = 0.08)
```



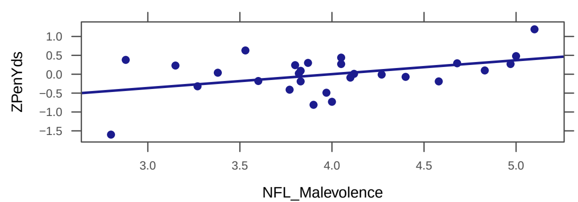
```
prop(~(Relapse.Placebo < -0.0833), data = Randomization.Coc)

  TRUE
0.3634
```

Randomization Test for a Correlation: Malevolent Uniforms and Penalties

Example 4.31

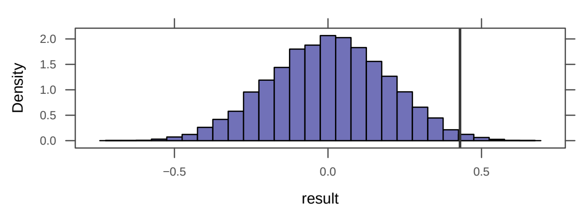
Testing correlation.

```
xyplot(ZPenYds ~ NFL_Malevolence, type = c("p", "r"), data = MalevolentUniformsNFL)
cor(ZPenYds ~ NFL_Malevolence, data = MalevolentUniformsNFL)

[1] 0.4298
```



1. H_0: $\rho = 0$; H_a: $\rho > 0$
2. Test statistic: $r = 0.43$ (the sample correlation)
3. We simulate a world in which $\rho = 0$:

```
Randomization.Mal <- do(10000) * cor(NFL_Malevolence ~ shuffle(ZPenYds), data = MalevolentUniformsNFL)
head(Randomization.Mal)

     result
1 -0.239727
2  0.000194
3  0.069772
4 -0.159381
5  0.210734
6  0.097180

histogram(~result, v = c(0.43), width = 0.05, data = Randomization.Mal)
```



 ©2014

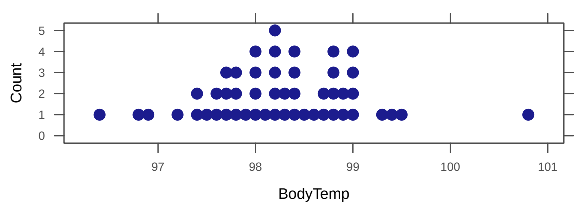
```
prop(~(result > 0.43), data = Randomization.Mal)

  TRUE
0.0096
```

Randomization Test for a Mean: Body Temperature

Example 4.33

Testing one mean.

```
mean(~BodyTemp, data = BodyTemp50)

[1] 98.26

dotPlot(~BodyTemp, v = c(98.26), width = 0.1, cex = 0.2, data = BodyTemp50)
```



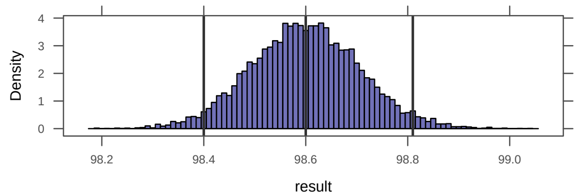
1. H_0: $\mu = 98.6$; H_a: $\mu \neq 98.6$
2. Test statistic: $\bar{x} = 98.26$ (the sample mean)
   Notice that the test statistic differs a bit from 98.6

   ```
   98.6 - mean(~BodyTemp, data = BodyTemp50)

   [1] 0.34
   ```

   But might this just be random variation? We need a randomization distribution to compare against.
3. If we resample, the mean will not be 98.6. But we shift the distribution a bit, then we will have the desired mean while preserving the shape of the distribution indicated by our sample. We simulate a world in which $\mu = 98.6$:

   ```
   Randomization.Temp <- do(10000) * (mean(~BodyTemp, data = resample(BodyTemp50)) + 0.34)
   head(Randomization.Temp, 3)

     result
   1  98.62
   2  98.62
   3  98.73

   mean(~result, data = Randomization.Temp)

   [1] 98.6
   ```

```
cdata(0.95, result, data = Randomization.Temp)

     low        hi central.p
   98.39     98.81      0.95

histogram(~result, width = 0.01, v = c(98.4, 98.6, 98.81), data = Randomization.Temp)
```



From this we can estimate the p-value:

```
prop(~abs(result - 98.6) > 0.34, data = Randomization.Temp)

  TRUE
0.0022
```

How do we interpret this (estimated) p-value of 0? Is it impossible to have a sample mean so far from 98.6 if the true population mean is 98.6? No. This merely means that we didn't see any such cases *in our 10000 randomization samples*. We might estimate the p-value as $p < 0.001$. Generally, to more accurately estimate small p-values, we must use many more randomization samples.

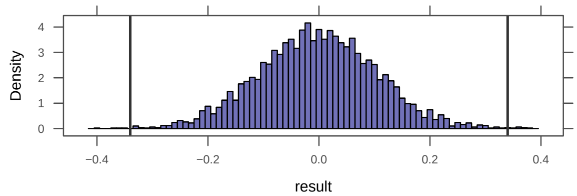
A different approach

An equivalent way to do the preceding test is based on a different way of expressing our hypotheses.

1. H_0: $\mu - 98.6 = 0$; H_a: $\mu - 98.6 \neq 0$
2. Test statistic: $\bar{x} - 98.6 = -0.34$
3. We we create a randomization distribution centered at $\mu - 98.6 = 0$:

```
Randomization.Temp2 <- do(5000) * (mean(~BodyTemp, data = resample(BodyTemp50)) - 98.26)
head(Randomization.Temp2, 3)

  result
1 -0.204
2 -0.144
3 -0.084

mean(~result, data = Randomization.Temp2)

[1] -0.002135

histogram(~result, width = 0.01, v = c(0.34, -0.34), data = Randomization.Temp2)
```

 ©2014



From this we can estimate the p-value:

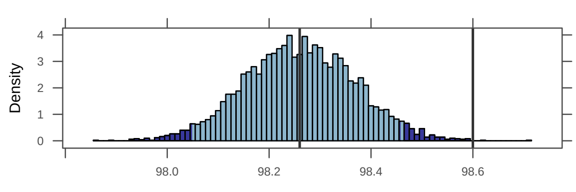
```
prop(~abs(result) > 0.34, data = Randomization.Temp2)

  TRUE
0.0024
```

Often there are multiple ways to express the same hypothesis test.

4.5 Confidence Intervals and Hypothesis Tests

If your randomization distribution is centered at the wrong value, then it isn't simulating a world in which the null hypothesis is true. This would happen, for example, if we got confused about randomization vs. bootstrapping.

```
Boot.Temp <- do(5000) * mean(~BodyTemp, data = resample(BodyTemp50))
head(Boot.Temp, 3)


  result
1  98.33
2  98.21
3  98.34


mean(~result, data = Boot.Temp)


[1] 98.26


cdata(0.95, result, data = Boot.Temp)


      low       hi central.p
    98.05    98.47      0.95


histogram(~result, width = 0.01, v = c(98.26, 98.6), groups = (98.05 <= result & result <=
    98.46), data = Boot.Temp)
```

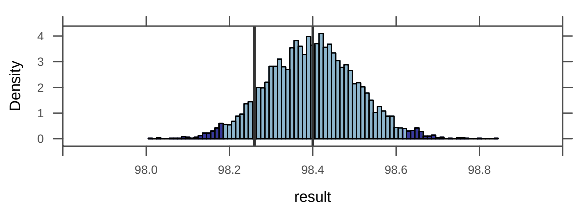


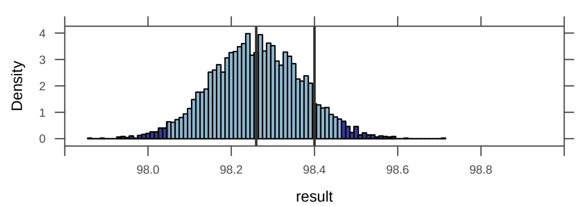
Notice that the distribution is now centered at our test statistic instead of at the value from the null hypothesis.

Example 4.35

1. H_0: $\mu = 98.4$; H_a: $\mu \neq 98.4$
2. Test statistic: $\bar{x} = 98.26$ (the sample mean)
3. We simulate a world in which $\mu = 98.4$:

```
Randomization.Temp3 <- do(5000) * (mean(~BodyTemp, data = resample(BodyTemp50)) + 0.14)
head(Randomization.Temp3, 3)

  result
1  98.29
2  98.51
3  98.49

mean(~result, data = Randomization.Temp3)

[1] 98.4

cdata(0.95, result, data = Randomization.Temp3)

      low        hi central.p
    98.19     98.61      0.95

histogram(~result, width = 0.01, v = c(98.26, 98.4), groups = (98.19 <= result & result <=
    98.62), xlim = c(97.8, 99), data = Randomization.Temp3)  # randomization
histogram(~result, width = 0.01, v = c(98.26, 98.4), groups = (98.05 <= result & result <=
    98.46), xlim = c(97.8, 99), data = Boot.Temp)  # bootstrap
```





Bibliography

Index