Teaching Statistics with R

Randall Pruim Nicholas J. Horton Daniel Kaplan

USCOTS workshop (May 17-19, 2011)

Contents

About These Notes

These materials were prepared for a workshop entitled *Teaching Statistics Using R* prior to the 2011 United States Conference on Teaching Statistics. We organized this workshop to help instructors integrate R (as well as some related technologies) into their statistics courses at all levels. We are presenting an approach to teaching introductory and intermediate statistics courses that is tightly coupled with computing generally and with R in particular. This flows from the philosophy outlined by Nolan and Temple Lang [NT10].

The activities and examples in the workshop are intended to highlight a modern approach to statistical education that focuses on modeling, resampling based inference, and multivariate graphical techniques. A secondary goal is to facilitate computing with data through use of small simulation studies, data scraping from the internet and appropriate statistical analysis workflow.

During this workshop, we are planning to introduce multiple activities, some appropriate for an introductory course, others suitable for higher levels, that demonstrate key concepts in statistics and modeling. We've crafted these notes partly to accompany our presentations, but primarily as a reference to support your teaching using R. We've included far more than we could possibly cover in 2 days, but hope to be able to provide some exposure and motivation for material and approaches that may differ from what you are doing at present, while also supporting the core material of a traditional course.

CAUTION!
You WILL find bugs both in this document and in our code. Please let us know when you encounter them so we can call in the exterminators.

A Work in Progress

Consider these notes a work in progress. We hope to continue improving them over the summer of 2011 with the goal of making a stable version available sometime in August 2011. We appreciate any feedback you are willing to share as we continue to work on these materials and the accompanying `mosaic` package. Drop us an email at `pis@mosaic.org` with any comments, suggestions, corrections, etc.

SUGGESTION BOX
Sometimes we will mark places where we would especially like feed back with one of these suggestion boxes. But we won't do that everywhere we want feedback or there won't be room for anything else.

Updated versions of this document will be posted from time to time at `http://mosaic-web.org`.

What's Ours Is Yours – To a Point

the work) if you attribute our work. More detailed information about the licensing is available at this web page: `http://www.mosaic-web.org/teachingRlicense.html`.

Two Audiences

The primary audience for these materials is instructors of statistics at the college or university level. A secondary audience is the students these instructors teach. Some of the sections, examples, and exercises are written with one or the other of these audiences more clearly at the forefront. This means that

1. Some of the materials can be used essentially as is with students.
2. Some of the materials aim to equip instructors to develop their own expertise in R and their own teaching materials.

Although the distinction can get blurry, and what works "as is" in one setting may not work "as is" in another, we'll try to indicate which parts of this book fit into each category as we go along.

R and R Packages

R can be obtained from `http://cran.r-project.org/`. Download and installation are pretty straightforward for Mac, PC, or linux machines.

In addition to R, we will make use of several packages that need to be installed and loaded separately. The `mosaic` and `Hmisc` packages will be use throughout. Other packages appear from time to time, including

- `fastR`
- `abd`
- `Zillow`
- `twitteR`
- `vcd`

We also make use of `lattice` and `grid` which are installed with R but must be loaded before use.

There are also a few things (mainly those using `manipulate()`) that require the RStudio interface to R. RStudio is available from `http://www.rstudio.org/`.

RStudio can be installed as a desktop (laptop) application or as a server application that is accessible to others via the Internet. We wish to thank the RStudio team for providing all workshop participants with accounts on their servers.

Marginal Notes

Marginal notes appear here and there. Sometimes these are side comments that we wanted to say, but didn't want to interrupt the flow to mention. These may describe more advanced features of the language or make suggestions about how to implement things in the classroom. Some are warnings to help you avoid common pitfalls. Still others contain requests for feedback.

DIGGING DEEPER
Many marginal notes will look like this one.

CAUTION!
But warnings are set differently to make sure they catch your attention.

SUGGESTION BOX
So, do you like having marginal notes in these notes?

DIGGING DEEPER
If you know LaTeX as well as R, then Sweave provide a nice solution for mixing the two.

Document Creation

This document was created May 17, 2011, using Sweave and R version 2.13.0 (2011-04-13)

Project MOSAIC

Project MOSAIC is a community of educators working to develop new ways to introduce mathematics, statistics, computation, and modeling to students in colleges and universities.

The purpose of the MOSAIC project is to help us share ideas and resources to improve teaching, and to develop a curricular and assessment infrastructure to support the dissemination and evaluation of these ideas. Our goal is to provide a broader approach to quantitative studies that provides better support for work in science and technology. The focus of the project is to tie together better diverse aspects of quantitative work that students in science, technology, and engineering will need in their professional lives, but which are today usually taught in isolation, if at all.

In particular, we focus on:

Modeling The ability to create, manipulate and investigate useful and informative mathematical representations of a real-world situations.

Statistics The analysis of variability that draws on our ability to quantify uncertainty and to draw logical inferences from observations and experiment.

Computation The capacity to think algorithmically, to manage data on large scales, to visualize and interact with models, and to automate tasks for efficiency, accuracy, and reproducibility.

Calculus The traditional mathematical entry point for college and university students and a subject that still has the potential to provide important insights to today's students.

Drawing on support from the US National Science Foundation (NSF DUE-0920350), Project MOSAIC supports a number of initiatives to help achieve these goals, including:

Faculty development and training opportunities, such as the USCOTS 2011 workshop and our 2010 gathering at the Institute for Mathematics and its Applications.

M-casts, a series of regularly scheduled seminars, delivered via the Internet, that provide a forum for instructors to share their insights and innovations and to develop collaborations to refine and develop them. A schedule of future M-casts and recordings of past M-casts are available at the Project MOSAIC web site, `http://mosaic-web.org`.

The development of a "concept inventory" to support teaching modeling. It is somewhat rare in today's curriculum for modeling to be taught. College and university catalogs are filled with descriptions of courses in statistics, computation, and calculus. There are many textbooks in these areas and the most new faculty teaching statistics, computation, and calculus have a solid idea of what should be included. But modeling is different. It's generally recognized as important, but few if instructors have a clear view of the essential concepts.

The construction of syllabi and materials for courses that teach the MOSAIC topics in a better integrated way. Such courses and materials might be wholly new constructions, or they might be incremental modifications of existing resources that draw on the connections between the MOSAIC topics.

We welcome and encourage your participation in all of these initiatives.

1 Getting Started: The First Week With R

1.1 Getting Students Familiar with R

1.1.1 Strategies

1. Start right away.

   Do something with R on day 1. Do something else on day 2. Have students do something by the end of week 1 at the latest.

2. Illustrate frequently.

   Have R running every class period and use it as needed throughout the course so students can see what R does. Preview topics by showing before asking students to do things.

3. Teach R as a programming language. (But don't overdo it.)

   There is a bit of syntax to learn – so teach it explicitly.

   - Capitalization (and spelling) matter
   - Explain carefully (and repeatedly) the syntax of functions.
   - Every object in R has a type (class). Ask frequently: *What type of thing is this?*
   - Get students to think about what arguments are needed for functions by asking *What does this function need to know to do its job?*

   Give more language details in higher level courses

   - More about R classes
   - User-defined functions
   - Control structures

4. "Less volume, more creativity." [Mike McCarthy, head coach, Green Bay Packers]

   Use a few methods frequently and students will learn how to use them well, flexibly, even creatively.

   Focus on a small number of data types: numerical vectors, character strings, factors, and data frames.

   Not everything needs to be introduced from first principles. For instance, categorical variables are easily enough understood by putting together simple concepts about character strings and vectors.

5. Find a way to have computers available for tests.

   It makes the test match the rest of the course and is a great motivator for students to learn R. It also changes what you can ask for and about on tests.

   Randy began doing this when his students asked him if there was a way to use computers during the test "*since that's how we do all the homework.*" He has students bring laptops to class. Nick has both in-class (without computer) and out-of-class (take home) components to his assessment.

6. Rethink your course.

   If you have taught computer-free or computer-light courses in the past, you may need to rethink some things. With ubiquitous computing, some things disappear from your course:

   - Reading statistical tables.

     Does anyone still consult a table for values of sin, or log? All of us have sworn off the use of tabulations of critical values of distributions (since none of us use them in our professional work, why would we teach this to students?)
   - "Computational formulas".

     Replace them with computation. Teach only the most intuitive formulas. Focus on how they lead to intuition and understanding, *not* computation.
   - (Most) hand calculations.

   At the same time, other things become possible that were not before:

   - Large data sets
   - Beautiful plots
   - Simulation/randomization/resampling based methods
   - Quick computations
   - Increased focus on concepts rather than calculations

   Get your students to think that using the computer is just part of how statistics is done, rather than an add-on.

7. Anticipate computationally challenged students, but don't give in.

   Some students pick up R very easily. In every course there will be a few students who struggle. Be prepared to help them, but don't spend time listening to their complaints. Focus on diagnosing what they don't know and how to help them "get it".

   Tell students to copy and paste R code and error messages into email when they have trouble. When you reply, explain how the error message helped you diagnose their problem and help them generalize your solution to other situations.

TEACHING TIP
Tell your students to copy and paste error messages into email rather than describe them vaguely. It's a big time saver for both of you.

1.1.2 Tactics

1. Introduce Graphics Early.

   Do graphics very early, so that students see that they can get impressive output from simple commands. Try to break away from their prior expectation that there is a "steep learning curve."

   Accept the defaults – don't worry about the niceties (good labels, nice breaks on histograms, colors) too early. Let them become comfortable with the basic graphics commands and then play (make sure it feels like play!) with fancying things up.

   Keep in mind that just because the graphs are easy to make on the computer doesn't mean your students understand how to read the graphs. Use examples that will help students develop good habits for visualizing data. Remember:

Students must learn to see before they can see to learn. – R. Pruim

2. Introduce Sampling and Randomization Early.

   Since sampling drives much of the logic of statistics, introduce the idea of a random sample very early, and have students construct their own random samples. The phenomenon of a sampling distribution can be introduced in an intuitive way, setting it up as a topic for later discussion and analysis.

1.2 Examples for Early in the Course

The remainder of this chapter has some of our favorite activities for early in the course.

1.2.1 Coins and Cups: The Lady Tasting Tea

This section is a slightly modified version of a handout R. Pruim has given Intro Stats students on Day 1 after going through the activity as a class discussion.

There is a famous story about a lady who claimed that tea with milk tasted different depending on whether the milk was added to the tea or the tea added to the milk. The story is famous because of the setting in which she made this claim. She was attending a party in Cambridge, England, in the 1920s. Also in attendance were a number of university dons and their wives. The scientists in attendance scoffed at the woman and her claim. What, after all, could be the difference?

All the scientists but one, that is. Rather than simply dismiss the woman's claim, he proposed that they decide how one should *test* the claim. The tenor of the conversation changed at this suggestion, and the scientists began to discuss how the claim should be tested. Within a few minutes cups of tea with milk had been prepared and presented to the woman for tasting.

At this point, you may be wondering who the innovative scientist was and what the results of the experiment were. The scientist was R. A. Fisher, who first described this situation as a pedagogical example in his 1925 book on statistical methodology [Fis25]. Fisher developed statistical methods that are among the most important and widely used methods to this day, and most of his applications were biological.

You might also be curious about how the experiment came out. How many cups of tea were prepared? How many did the woman correctly identify? What was the conclusion?

Fisher never says. In his book he is interested in the method, not the particular results. But let's suppose we decide to test the lady with ten cups of tea. We'll flip a coin to decide which way to prepare the cups. If we flip a head, we will pour the milk in first; if tails, we put the tea in first. Then we present the ten cups to the lady and have her state which ones she thinks were prepared each way.

It is easy to give her a score (9 out of 10, or 7 out of 10, or whatever it happens to be). It is trickier to figure out what to do with her score. Even if she is just guessing and has no idea, she could get lucky and get quite a few correct – maybe even all 10. But how likely is that?

Let's try an experiment. I'll flip 10 coins. You guess which are heads and which are tails, and we'll see how you do.

⋮

Comparing with your classmates, we will undoubtedly see that some of you did better and others worse.

Now let's suppose the lady gets 9 out of 10 correct. That's not perfect, but it is better than we would expect for someone who was just guessing. On the other hand, it is not impossible to get 9 out of 10 just by guessing. So here is Fisher's great idea: Let's figure out how hard it is to get 9 out of 10 by guessing. If it's not so hard to do, then perhaps that's just what happened, so we won't be too impressed with the lady's tea tasting ability. On the other hand, if it is really unusual to get 9 out of 10 correct by guessing, then we will have some evidence that she must be able to tell something.

But how do we figure out how unusual it is to get 9 out of 10 just by guessing? We'll learn another method later, but for now, let's just flip a bunch of coins and keep track. If the lady is just guessing, she might as well be flipping a coin.

So here's the plan. We'll flip 10 coins. We'll call the heads correct guesses and the tails incorrect guesses. Then we'll flip 10 more coins, and 10 more, and 10 more, and .... That would get pretty tedious. Fortunately, computers are good at tedious things, so we'll let the computer do the flipping for us.

The `rflip()` function can flip one coin

```
> require(mosaic)
> rflip()

Flipping 1 coins [ Prob(Heads) = 0.5 ] ...

T

Result: 0 heads.
```

or a number of coins

```
> rflip(10)

Flipping 10 coins [ Prob(Heads) = 0.5 ] ...

T H H T H H T H T T

Result: 5 heads.
```

Typing `rflip(10)` a bunch of times is almost as tedious as flipping all those coins. But it is not too hard to tell R to `do()` this a bunch of times.

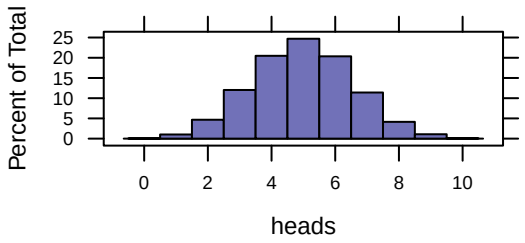
```
> do(3) * rflip(10)

   n heads tails
1 10     6     4
2 10     6     4
3 10     6     4
```

Let's get R to `do()` it for us 10,000 times and make a table of the results.

```
> random.ladies <- do(10000) * rflip(10)

> table(random.ladies$heads)

   0    1    2    3    4    5    6    7    8    9   10
   5  102  467 1203 2048 2470 2035 1140  415  108    7

> perctable(random.ladies$heads)     # display table using percentages

    0     1     2     3     4     5     6     7     8     9    10
 0.05  1.02  4.67 12.03 20.48 24.70 20.35 11.40  4.15  1.08  0.07
```

We can display this table graphically using a plot called a **histogram**.


```
> histogram(~ heads, random.ladies,
         breaks=-.5 + (0:11)
         )
```



We have control of the histogram display by specifying the breaks from -.5 to 10.5:

```
> -.5 + (0:11)
 [1] -0.5  0.5  1.5  2.5  3.5  4.5  5.5  6.5  7.5  8.5  9.5 10.5
```

You might be surprised to see that the number of correct guesses is exactly 5 (half of the 10 tries) only 25% of the time. But most of the results are quite close to 5 correct. 67% of the results are 4, 5, or 6, for example. And 90% of the results are between 3 and 7 (inclusive). But getting 8 correct is a bit unusual, and getting 9 or 10 correct is even more unusual.

So what do we conclude? It is possible that the lady could get 9 or 10 correct just by guessing, but it is not very likely (it only happened in about 1.2% of our simulations). So *one of two things must be true*:

- The lady got unusually "lucky", or
- The lady is not just guessing.

Although Fisher did not say how the experiment came out, others have reported that the lady correctly identified all 10 cups! [Sal01]

A different design

Suppose instead that we prepare five cups each way (and that the woman tasting knows this). We give her five cards labeled "milk first", and she must place them next to the cups that had the milked poured first. How does this design change things?

```
> results <- do(10000) * table(sample( c('M','M','M','M','M','T','T','T','T','T'), 5))
> table(results$M) / 10000
     1      2      3      4      5
0.0993 0.3965 0.3928 0.1028 0.0042
```

2 An Introduction to R

This is a lightly modified version of a handout RJP used with his Intro Stats students Spring 2011. Aside from the occasional comment to instructors, this chapter could be used essentially as is with students.

2.1 Welcome to R and RStudio

R is a system for statistical computation and graphics. We use R for several reasons:

1. R is open-source and freely available for Mac, PC, and Linux machines. This means that there is no restriction on having to license a particular software program, or have students work in a specific lab that has been outfitted with the technology of choice.
2. R is user-extensible and user extensions can easily be made available to others.
3. R is commercial quality. It is the package of choice for many statisticians and those who use statistics frequently.
4. R is becoming very popular with statisticians and scientists, especially in certain sub-disciplines, like genetics. Articles in research journals such as *Science* often include links to the R code used for the analysis and graphics presented.
5. R is very powerful. Furthermore, it is gaining new features every day. New statistical methods are often available first in R.

RStudio provides access to R in a web browser. This has some additional advantages: no installation is required, the interface has some additional user-friendly components, and work begun on one machine can be picked up seamlessly later on somewhere else.

The URL for the RStudio beta server is

`http://beta.rstudio.org/`

It is also possible to download RStudio server and set up your own server or RStudio desktop for stand-alone processing.

Once you have logged in to an RStudio server, you will see something like Figure 2.1.

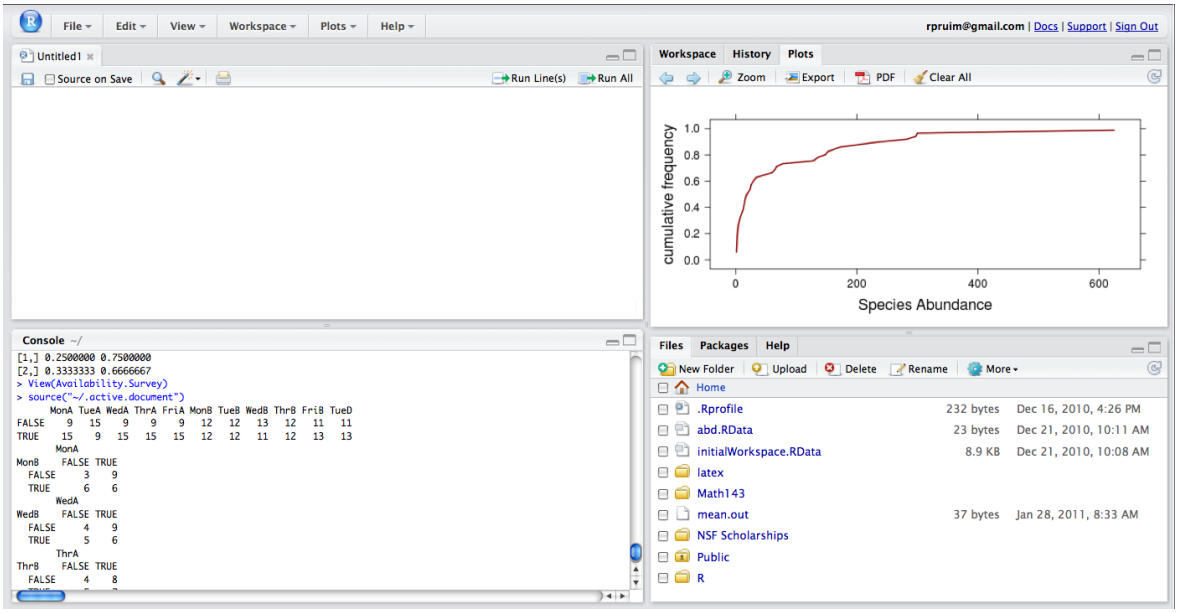


Figure 2.1: Welcome to RStudio.

Notice that RStudio divides its world into four panels. Several of the panels are further subdivided into multiple tabs. The console panel is where we type commands that R will execute.

2.2 Using R as a Calculator

R can be used as a calculator. Try typing the following commands in the console panel.

```
> 5 + 3
[1] 8
> 15.3 * 23.4
[1] 358
> sqrt(16)
[1] 4
```

Teaching Tip
It's probably best to settle on using one or the other of the right-to-left assignment operators rather than to switch back and forth. The authors of this document have different preferences.

You can save values to named variables for later reuse.

```
> product = 15.3 * 23.4       # save result
> product                     # show the result
[1] 358
> product <- 15.3 * 23.4      # <- is assignment operator, same as =
> product
[1] 358
> 15.3 * 23.4 -> newproduct   # -> assigns to the right
> newproduct
[1] 358
```

Once variables are defined, they can be referenced with other operators and functions.

```
> .5 * product                # half of the product
```

```
[1] 179
> log(product)                # (natural) log of the product
[1] 5.88
> log10(product)              # base 10 log of the product
[1] 2.55
> log(product,base=2)         # base 2 log of the product
[1] 8.48
```

The semi-colon can be used to place multiple commands on one line. One frequent use of this is to save and print a value all in one go:

```
> 15.3 * 23.4 -> product; product    # save result and show it
[1] 358
```

Four Things to Know About R

1. R is case-sensitive

   If you mis-capitalize something in R it won't do what you want.

2. Functions in R use the following syntax:

   ```
   > functionname( argument1, argument2, ... )
   ```

   - The arguments are always *surrounded by (round) parentheses* and *separated by commas*.

     Some functions (like `data()`) have no required arguments, but you still need the parentheses.

   - If you type a function name without the parentheses, you will see the *code* for that function (this probably isn't what you want at this point).

3. TAB completion and arrows can improve typing speed and accuracy.

   If you begin a command and hit the TAB key, R will show you a list of possible ways to complete the command. If you hit TAB after the opening parenthesis of a function, it will show you the list of arguments it expects. The up and down arrows can be used to retrieve past commands.

4. If you see a + prompt, it means R is waiting for more input.

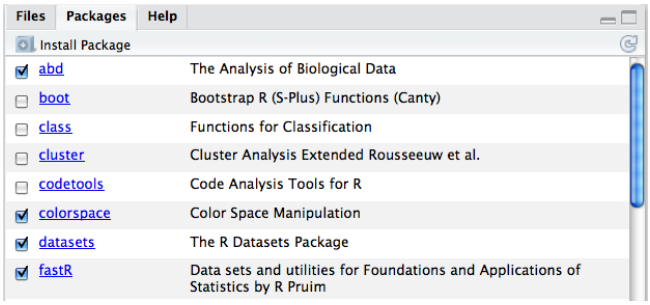
   Often this means that you have forgotten a closing parenthesis or made some other syntax error. If you have messed up and just want to get back to the normal plot, hit the escape key and start the command fresh.

TEACHING TIP
To help students get the hang of function arguments, ask them *What information does the computer need to compute this?*

CAUTION!
Your students will sometimes find themselves in a syntactic hole from which they cannot dig out. Teach them about the ESC key early.

2.3 R Packages

In addition to its core features, R provides many more features through a (large) number of packages. To use a package, it must be installed (one time), and loaded (each session). A number of packages are already available in RStudio. The Packages tab in RStudio will show you the list of installed packages and indicate which of these are loaded.



Here are some packages we will use frequently:

- `lattice` (for graphics; this will always be installed in R)
- `mosaic` (for teaching statistics; this will need to be installed in R from CRAN, see section 2.8.3)
- `Hmisc` (a package with some nice utilities; available on CRAN)

There are others that we use from time to time as well. You should install `Hmisc` and `mosaic` the first time you use R. Once a package is installed, it is available to be loaded in the current or any future session. You can install these packages by clicking on the "Install Package" button in RStudio and following the directions or by using the following command:

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

Once these are installed, you can load them by checking the box in the Packages tab or by using `require()` (or `library()`)

```
> require(lattice)          # could also use library(lattice)
> require(Hmisc)            # do this one before mosaic
> require(mosaic)           # do this one after Hmisc
```

2.4 Getting Help

If something doesn't go quite right, or if you can't remember something, it's good to know where to turn for help. In addition to asking your friends and neighbors, you can use the R help system.

2.4.1 ?

To get help on a specific function or data set, simply precede its name with a `?`:

```
> ?col.whitebg()
```

2.4.2 `apropos()`

If you don't know the exact name of a function, you can give part of the name and R will find all functions that match. Quotation marks are mandatory here.

```
> apropos('hist')             # must include quotes.  single or double.
```



```
 [1] "event.history"              "hist"
 [3] "hist.data.frame"            "hist.default"
 [5] "hist.FD"                    "hist.scott"
 [7] "histbackback"               "histochart"
 [9] "histogram"                  "histogram"
[11] "history"                    "histSpike"
[13] "ldahist"                    "loadhistory"
[15] "panel.histogram"            "panel.xhistogram"
[17] "pmfhistogram"               "prepanel.default.histogram"
[19] "savehistory"                "truehist"
[21] "xhistogram"                 "xhistogram"
```

2.4.3 `??` and `help.search()`

If that fails, you can do a broader search using `??` or `help.search()`, which will find matches not only in the names of functions and data sets, but also in the documentation for them. Quotations marks are optional here.

```
> ??histogram                  # any of these will work
> ??"histogram"
> ??'histogram'
> help.search('histogram')
```

2.4.4 Examples and Demos

Many functions and data sets in R include example code demonstrating typical uses. For example,

```
> example(histogram)
```

Not all package authors are equally skilled at creating examples. Some of the examples are next to useless, others are excellent.

will generate a number of example plots (and provide you with the commands used to create them). Examples such as this are intended to help you learn how specific R functions work. These examples also appear at the end of the documentation for functions and data sets.

The `mosaic` package (and some other packages as well) also includes demos. Demos are bits of R code that can be executed using the `demo()` command with the name of the demo.

To see how demos work, give this a try:

```
> demo(histogram)
```

Demos are intended to illustrate a concept, a method, or some such thing, and are independent of any particular function or data set.

You can get a list of available demos using

```
> demo()                     # all demos
> demo(package='mosaic')     # just demos from mosaic package
```

2.5 Data

2.5.1 Data in Packages

Many packages contain data sets. You can see a list of all data sets in all loaded packages using

```
> data()
```

Typically (provide the author of the package allowed for lazy loading of data) you can use data sets by simply typing their names. But if you have already used that name for something or need to refresh the data after making some changes you no longer want, you can explicitly load the data using the `data()` function with the name of the data set you want.

```
> data(iris)
```

2.5.2 Data Frames

Data sets are usually stored in a special structure called a **data frame**.

Data frames have a 2-dimensional structure.

- 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).

We'll talk later about how to get your own data into R. For now we'll use some data that comes with R and is all ready for you to use. The `iris` data frame contains 5 **variables** measured for each of 150 iris plants (the observational units). The `iris` data set is included with the default R installation. (Technically, it is located in a package called `datasets` which is always available.)

There are several ways we can get some idea about what is in the `iris` data frame.

```
> str(iris)
```

```
'data.frame':        150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
> summary(iris)
```

```
  Sepal.Length   Sepal.Width    Petal.Length   Petal.Width          Species
 Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1   setosa    :50
 1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3   versicolor:50
 Median :5.80   Median :3.00   Median :4.35   Median :1.3   virginica :50
 Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
 3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
 Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5
```

```
> head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

In interactive mode, you can also try

```
> View(iris)
```

to see the data or

```
> ?iris
```

to get the documentation about for the data set.

Access to an individual variable in a data frame uses the `$` operator in the following syntax:

```
> dataframe$variable
```

or

```
> with(dataframe, variable)
```

For example, either of

```
> iris$Sepal.Length
```

or

```
> with(iris, Sepal.Length)
```

shows the contents of the `Sepal.Length` variable in the following format.

```
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1
 [21] 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1
 [41] 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2
 [61] 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7
 [81] 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7
[101] 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0
[121] 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9
[141] 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

But this isn't very useful for a large data set. We would prefer to compute numerical or graphical summaries. We'll do that shortly.

The `attach()` function in R can be used to make objects within dataframes accessible in R with fewer keystrokes, but we strongly discourage its use, as it often leads to name conflicts. The Google R Style Guide (`http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html`) echoes this advice, stating that *The possibilities for creating errors when using attach are numerous. Avoid it.* It is far better to directly access variables using the `\$` syntax or to use the `with()` function.

CAUTION!
Avoid the use of `attach()`.

2.5.3 Using Your Own Data

RStudio will help you import your own data. To do so use the "Import Dataset" button in the Workspace tab. You can load data from text files, from the web, or from google spreadsheets.

From Google. The easiest of these is the Google spreadsheets option: Just click, select your spreadsheet, choose a name, and you're done.

From Excel, you need to follow a 3-step process:

1. Save your Excel worksheet as a csv (comma separated value) file.
2. Upload (in the Files tab) your csv file to the server, where you can create folders and store files in your personal account. (To share things with others: Put the files in your Public folder. Read the`AboutPublic.txt` file in that folder for directions.)
3. Now import "from a text file" in the Workspace tab.

In either case, be sure to do the following:

- Choose good variables names.
- Put your variables names in the first row.
- Use each subsequent row for one observational unit.
- Give the resulting data frame a good name.

LOAD()
`load()` is used for opening files that store R objects in 'native' format.

If you are not using RStudio, you can read files using `read.csv()` or `read.table()` (for white space delimited files). The `mosaic()` package includes a function called `read.file()` that uses slightly different default settings and infers whether it should use `read.csv()`, `read.table()`, or `load()` based on the file name.

Each of these functions also accepts a URL in place of a file name, which provides an easy way to distribute data via the Internet:

```
> births <- read.table('http://www.calvin.edu/~rpruim/data/births.txt', header=TRUE)
> head(births)        # number of live births in the US each day of 1978.
```

```
    date births datenum dayofyear
1 1/1/78   7701    6575         1
2 1/2/78   7527    6576         2
3 1/3/78   8825    6577         3
4 1/4/78   8859    6578         4
5 1/5/78   9043    6579         5
6 1/6/78   9208    6580         6
```

The `mosaic` package provides `read.file()` which attempts to infer the file type from its extension and then applies `read.csv()`, `read.table()`, or `load()`.

```
> births <- read.file('http://www.calvin.edu/~rpruim/data/births.txt')
```

It also sets a number of defaults, including `header=TRUE`.

2.5.4 Putting Data Into a Package

It is not that difficult to take a collection of csv files (a format available for many books) and put them all into a package. The `abd` package contains data sets from *The Analysis of Biological Data*, for example. Kevin Middleton and Randall Pruim contacted the authors and obtained permission to build and disseminate this package.

The `findData()` function in `abd` makes it easy to map examples and exercises in that book to data frame names in the `abd` package.

```
> findData('human')         # all data sets with 'human' in the name
```

```
              name chapter    type number sub
24 HumanGeneLengths       4 Example      1
43    HumanBodyTemp      11 Example      3
```

```
> findData(2)               # all data sets in chapter 2
```

```
                  name chapter    type number sub
1           TeenDeaths       2 Example      1   a
2          DesertBirds       2 Example      1   b
3       SockeyeFemales       2 Example      1   c
4      GreatTitMalaria       2 Example      3
5           Hemoglobin       2 Example      4
```

```
6               Guppies       2 Example      5   a
7                 Lynx        2 Example      5   b
8     EndangeredSpecies       2 Problem      6
9       ShuttleDisaster       2 Problem     10
10          Convictions       2 Problem     16
11 ConvictionsAndIncome       2 Problem     17
12            Fireflies       2 Problem     18
13     NeotropicalTrees       2 Problem     23
```

For information on how to create such packages, consult the *Writing R Extensions* manual on CRAN.

2.6 Summarizing Data

2.6.1 A Few Numerical Summaries

R includes functions that compute a wide range of numerical and graphical summaries. Most of the numerical summaries already familiar to you have obvious names. Here are a few examples.

```
> with(iris, mean(Sepal.Length))                  # or mean(iris$Sepal.Length)
[1] 5.84
> with(iris, median(Sepal.Length))                # or median(iris$Sepal.Length)
[1] 5.8
> with(iris, sd(Sepal.Length))                    # or sd(iris$Sepal.Length)
[1] 0.828
> with(iris, quantile(Sepal.Length))              # or quantile(iris$Sepal.Length)
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
> with(iris, IQR(Sepal.Length))                   # or IQR(iris$Sepal.Length)
[1] 1.3
> IQR(iris$Sepal.Length)
[1] 1.3
```

The `favstats()` function in the `mosaic` package computes several numerical summaries all at once.

```
> require(mosaic)               # if you haven't already loaded the mosaic package
> favstats(iris$Sepal.Length)
 min  Q1 median  Q3 max mean    sd   n missing
 4.3 5.1    5.8 6.4 7.9 5.84 0.828 150       0
```

Here's something a little fancier.

```
> require(Hmisc)
> summary(Sepal.Length ~ Species, data=iris, fun=favstats)
Sepal.Length    N=150

+-------+----------+---+----+----+-------+---+----+-----+-----+---+--------+
|       |          |N  |min |Q1  |median |Q3 |max |mean |sd   |n  |missing |
+-------+----------+---+----+----+-------+---+----+-----+-----+---+--------+
|Species|setosa    | 50|4.3 |4.80|5.0    |5.2|5.8 |5.01 |0.352| 50|0       |
|       |versicolor| 50|4.9 |5.60|5.9    |6.3|7.0 |5.94 |0.516| 50|0       |
|       |virginica | 50|4.9 |6.23|6.5    |6.9|7.9 |6.59 |0.636| 50|0       |
+-------+----------+---+----+----+-------+---+----+-----+-----+---+--------+
|Overall|          |150|4.3 |5.10|5.8    |6.4|7.9 |5.84 |0.828|150|0       |
+-------+----------+---+----+----+-------+---+----+-----+-----+---+--------+
```

(Note: This requires installation of the `Hmisc` package.)

2.6.2 Lattice Graphics

There are several ways to make graphs in R. One approach is a system called `lattice` graphics. The first step for using `lattice` is to load the `lattice` package using the check box in the Packages tab or using the following command:

```
> require(lattice)
```

`lattice` plots make use of a **formula interface**:

```
> plotname( y ~ x | z, data=dataname, groups=grouping_variable, ...)
```

- Here are the names of several `lattice` plots:
  - `dotPlot` (notice the capital P; `dotplot()` does something different) [1]
  - `histogram` (for histograms)
  - `bwplot` (for boxplots)
  - `xyplot` (for scatter plots)
  - `qqmath` (for quantile-quantile plots)
- `x` is the name of the variable that is plotted along the horizontal (x) axis.
- `y` is the name of the variable that is plotted along the vertical (y) axis. (For some plots, this slot is empty because R computes these values from the values of `x`.)
- `z` is a conditioning variable used to split the plot into multiple subplots called **panels**.
- `grouping_variable` is used to display different groups differently (different colors or symbols, for example) within the same panel.
- `...` There are many additional arguments to these functions that let you control just how the plots look. (But we'll focus on the basics for now.)

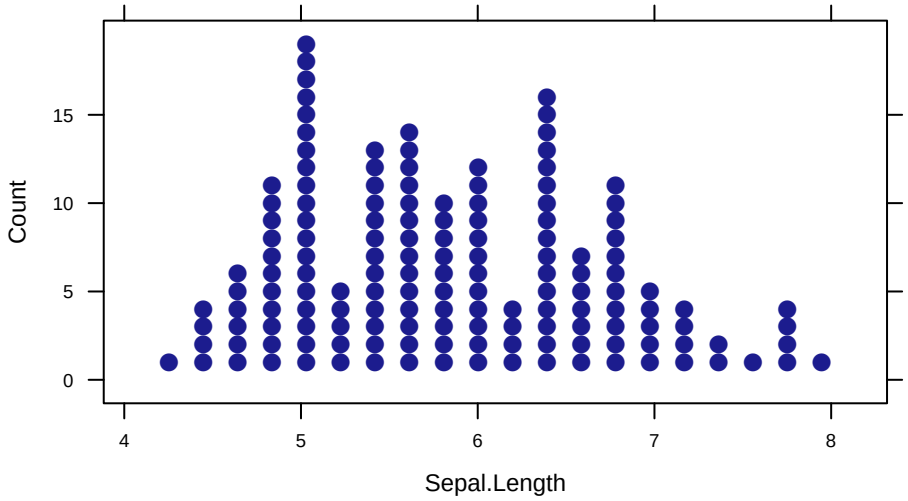
2.6.3 Dot plots: `dotPlot()`

A dot plot represents each value of a quantitative variable with a dot. The values are rounded a bit so that the dots line up neatly, and dots are stacked up into little towers when the data values cluster near each other.

For a dot plot, the `y` component of the formula is empty since we let R calculate that for us.

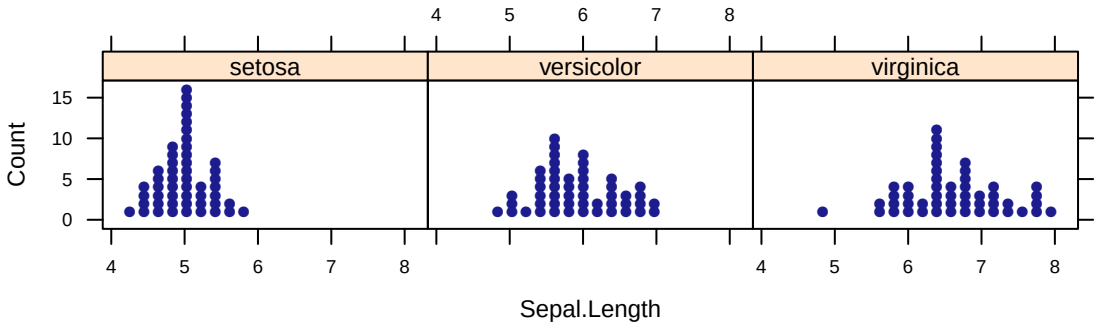
```
> dotPlot(~ Sepal.Length, data=iris, n=20)   # n=20 gives us approximately 20 towers
```

[1] `dotPlot()` is in the `mosaic` package and was created because `dotplot()` in the `lattice` package makes something different from what we call a dot plot.



We can use a conditional variable to give us separate dot plots for each species.

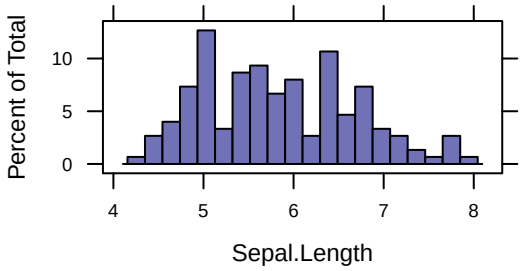
```
> dotPlot(~ Sepal.Length | Species, data=iris, n=20)
```



2.6.4 Histograms: `histogram()`

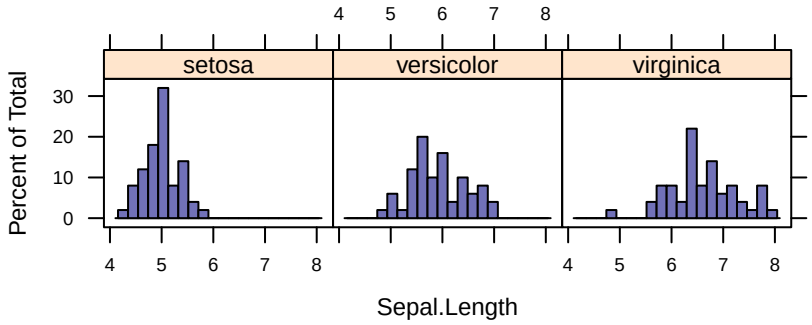
Histograms are a lot like dot plots, but the towers of dots are replaced by a vertical bar. Again the `y` component of the formula is empty since we let R compute the heights of the bars for us.

```
> histogram(~ Sepal.Length, data=iris, n=20)       # n= 20 gives approx. 20 bars
```



We can use a conditional variable to give us separate histograms for each species.

```
> histogram(~ Sepal.Length | Species, data=iris, n=20)
```

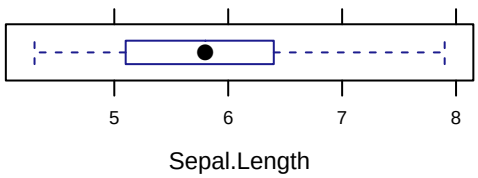


In lattice lingo, the three subplots are called panels and the labels at the top are called strips. (Strips can be placed on the left side if you prefer.)

2.6.5 Boxplots: `bwplot()`

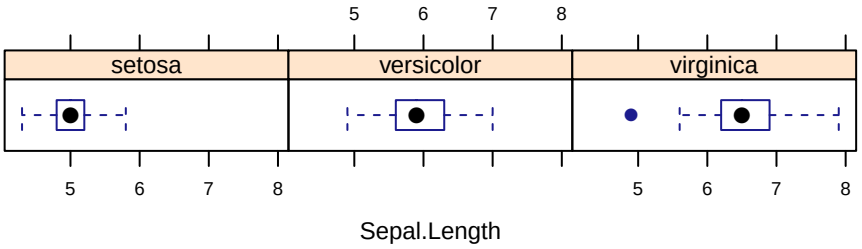
Boxplots are made pretty much the same way as histograms:

```
> bwplot(~ Sepal.Length, data=iris)
```



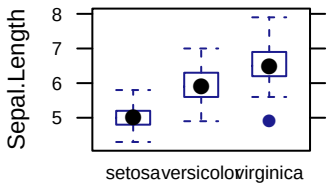
We can use conditioning as we did for histograms:

```
> bwplot(~ Sepal.Length | Species, data=iris)
```



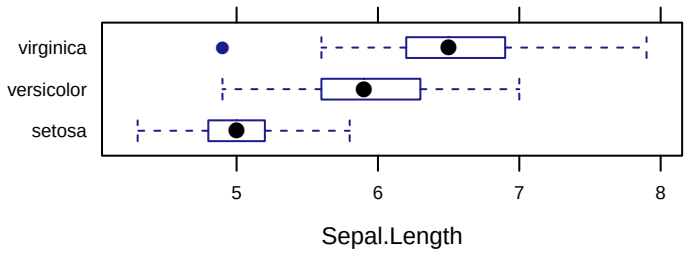
But there are betters ways to do this.

```
> bwplot(Sepal.Length ~ Species, data=iris)
```



This is better, but the species names run into each other. We could fix that run-together text by using abbreviated names or rotating the labels 45 or 90 degrees. Instead of those solutions, we can also just reverse the roles of the horizontal and vertical axes.

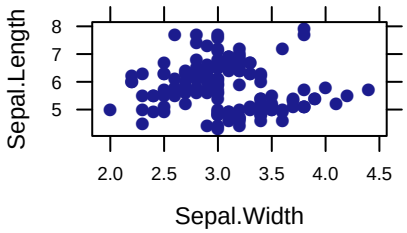
```
> bwplot(Species ~ Sepal.Length, data=iris)
```



2.6.6 Scatterplots: `xyplot()`

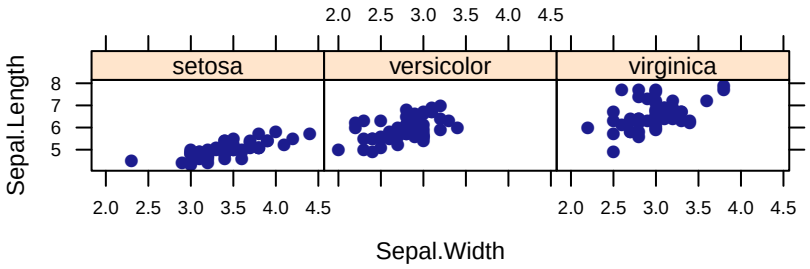
Scatterplots are made with `xyplot()`. The formula interface is very natural for this. Just remember that the "y variable" comes first. (Its label is also farther left on the plot, if that helps you remember.)

```
> xyplot(Sepal.Length ~ Sepal.Width, data=iris)
```



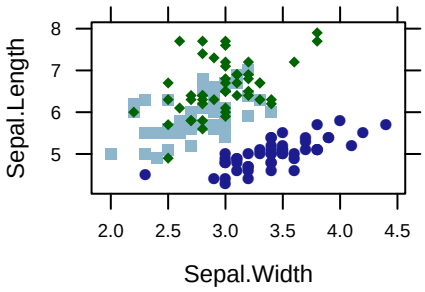
Again, we can use conditioning to make a panel for each species.

```
> xyplot(Sepal.Length ~ Sepal.Width | Species, data=iris)
```



Even better (for this example), we can use the `groups` argument to indicate the different species using different symbols on the same panel.

```
> xyplot(Sepal.Length ~ Sepal.Width, groups=Species, data=iris)
```



2.6.7 Saving Your Plots

There are several ways to save plots, but the easiest is probably the following:

1. In the Plots tab, click the "Export" button.
2. Copy the image to the clipboard using right click.

3. Go to your Word document and paste in the image.

4. Resize or reposition your image in Word as needed.

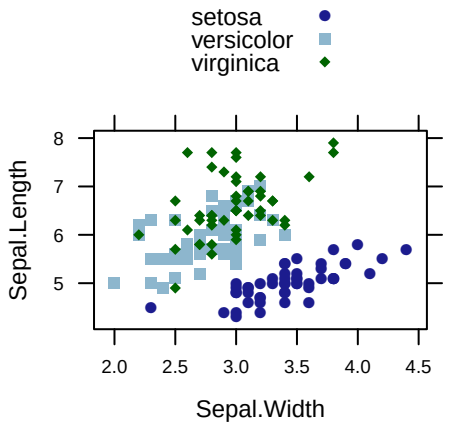
2.6.8 A Few Bells and Whistles

There are lots of arguments that control how these plots look. Here are just a few examples.

auto.key

It would be useful to have a legend for the previous plot. `auto.key=TRUE` turns on a simple legend. (There are ways to have more control, if you need it.)

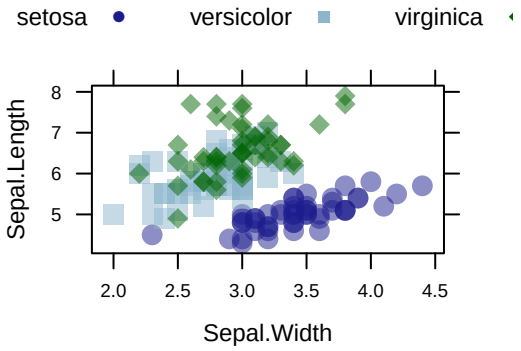
```
> xyplot(Sepal.Length ~ Sepal.Width, groups=Species, data=iris,
         auto.key=TRUE)
```



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.

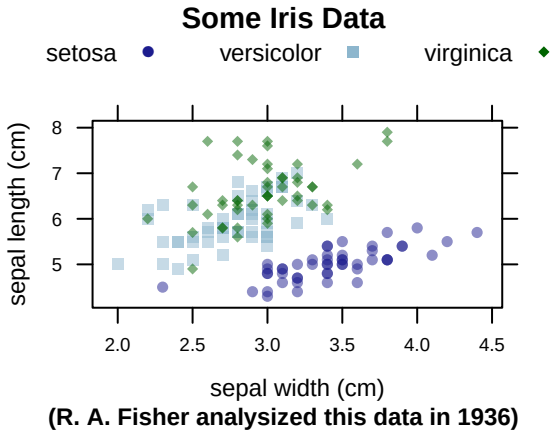
```
> xyplot(Sepal.Length ~ Sepal.Width, groups=Species, data=iris,
         auto.key=list(columns=3),
         alpha=.5,
         cex=1.3)
```



main, sub, xlab, ylab

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

```
> 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=.5,
         auto.key=list(columns=3))
```



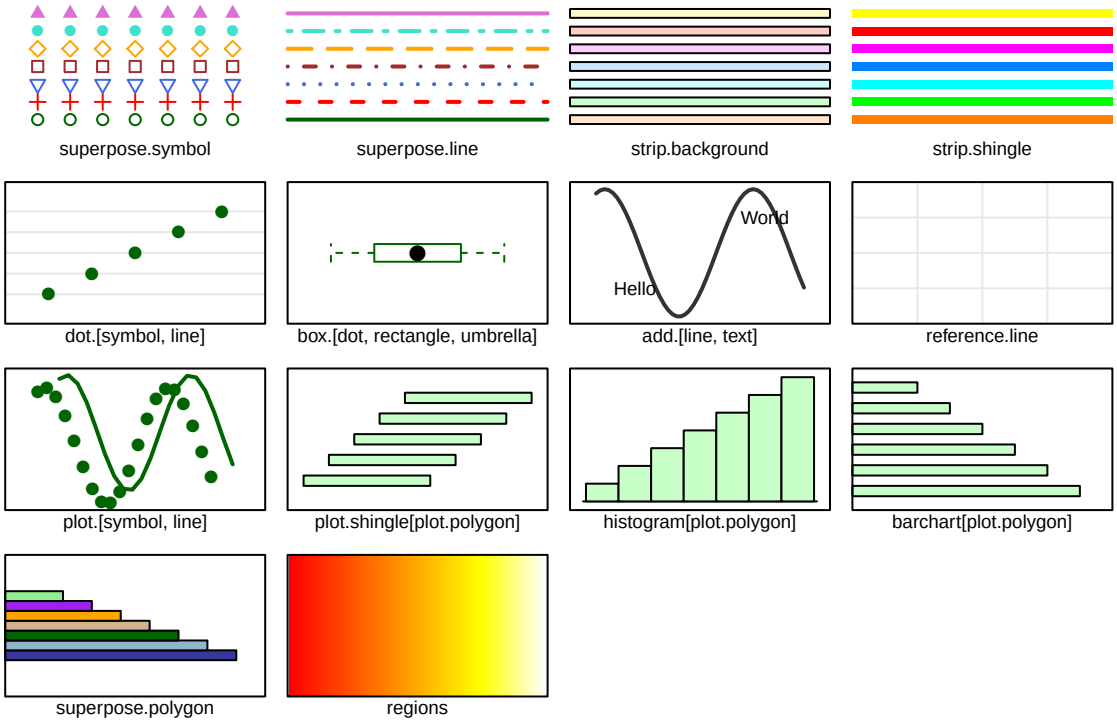
trellis.par.set()

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

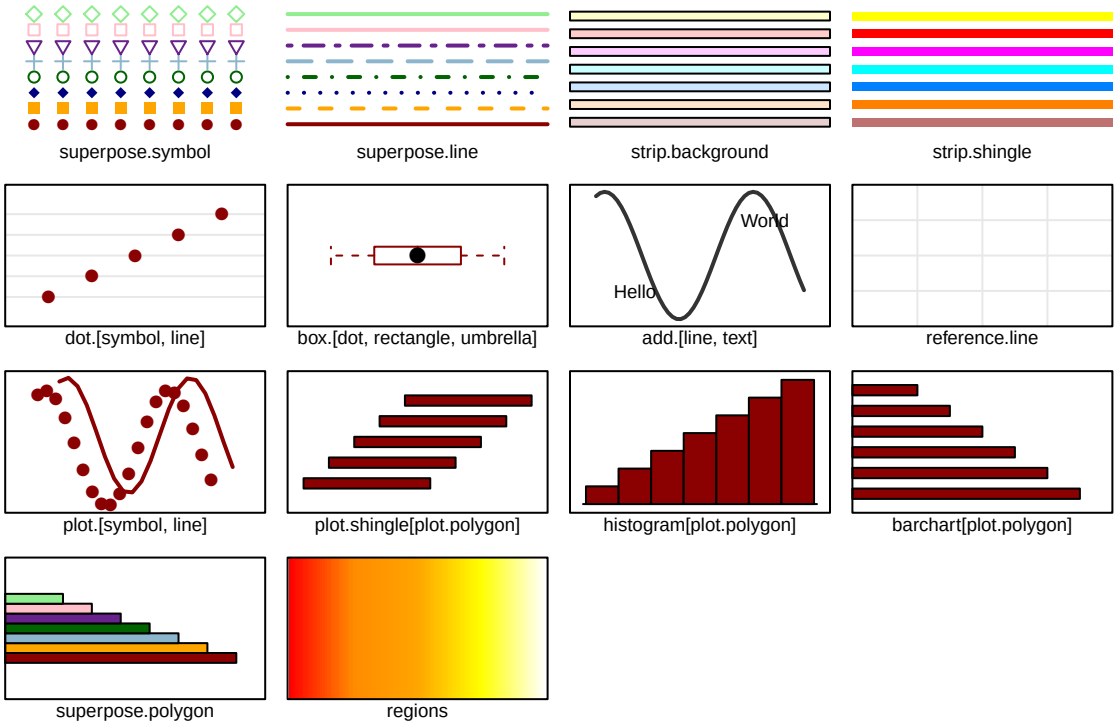
```
> trellis.par.set(fontsize=list(text=7))    # base size for text is 7 point
```

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.

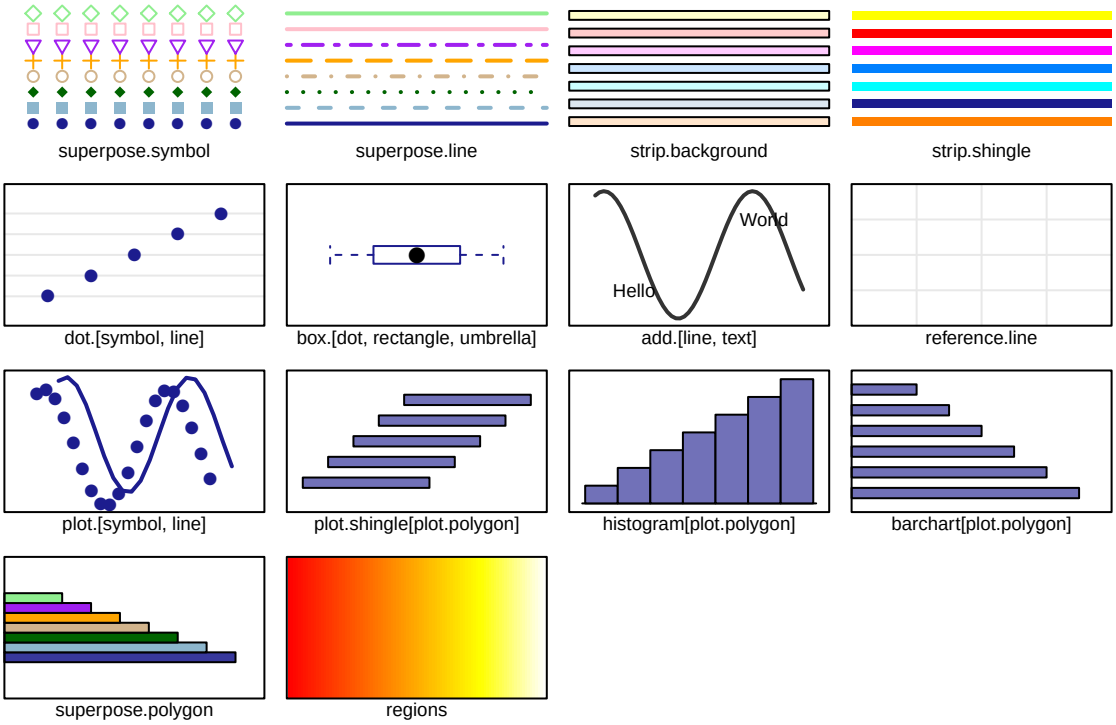
```
> trellis.par.set(theme=col.whitebg())      # a theme in the lattice package
> show.settings()
```




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



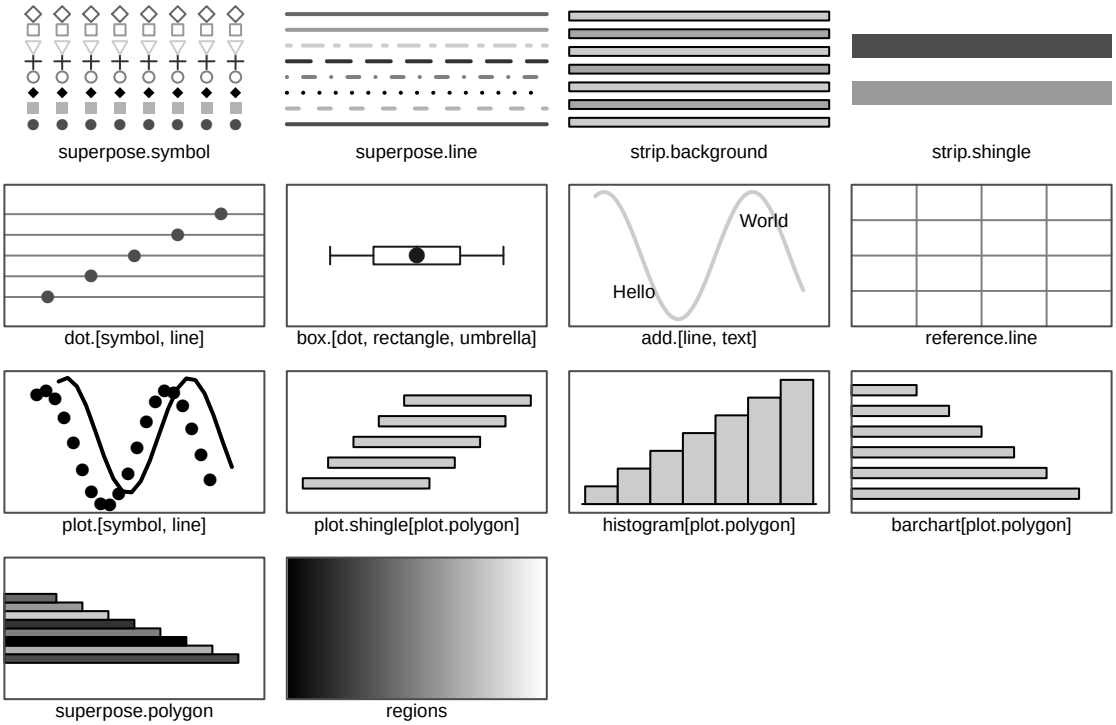

```
> trellis.par.set(theme=col.mosaic())       # a theme in the mosaic package
> show.settings()
```



SUGGESTION BOX
Do you have a great eye for colors? Help us design other lattice themes.

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

DIGGING DEEPER
The RColorBrewer package provides several palettes of colors that are highly distinguishable and aesthetically pleasing.




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

2.6.9 Tabulating Categorical Data

The Current Population Survey (CPS) is used to supplement census information between census years. These CPS data frame consist of a random sample of persons from the CPS, with information on wages and other characteristics of the workers, including sex, number of years of education, years of work experience, occupational status, region of residence and union membership.

```
> head(CPS,3)
```

```
  wage educ race sex hispanic south married exper union age sector
1  9.0   10    W   M       NH    NS Married    27   Not  43  const
2  5.5   12    W   M       NH    NS Married    20   Not  38  sales
3  3.8   12    W   F       NH    NS  Single     4   Not  22  sales
```

Making Frequency and Contingency Tables with `xtabs()`

Categorical variables are often summarized in a table. R can make a table for a categorical variable using `xtabs()`.

```
> xtabs(~ race, CPS)
```

```
race
 NW   W
 67 467
```

```
> xtabs(~ sector, CPS)
```

```
sector
clerical    const    manag    manuf    other     prof    sales  service
      97       20       55       68       68      105       38       83
```

Alternatively, we can use `table()`, `proptable()`, and `perctable()` to make tables of counts, proportions, or percentages.

```
> with(CPS, table(race))
```

```
race
 NW   W
 67 467
```

```
> with(CPS, proptable(sector))
```

```
sector
clerical    const    manag    manuf    other     prof    sales  service
  0.1816   0.0375   0.1030   0.1273   0.1273   0.1966   0.0712   0.1554
```

```
> with(CPS, perctable(sector))
```

```
sector
clerical    const    manag    manuf    other     prof    sales  service
   18.16     3.75    10.30    12.73    12.73    19.66     7.12    15.54
```

We can make a **cross-table** (also called a **contingency table** or a **two-way table**) summarizing this data with `xtabs()`. This is often a more useful view of data with two categorical variables.

```
> xtabs(~ race + sector, CPS)
```

```
    sector
race clerical const manag manuf other prof sales service
  NW       15     3     6    11     5    7     3      17
  W        82    17    49    57    63   98    35      66
```

Entering Tables by Hand

Because categorical data is so easy to summarize in a table, often the frequency or contingency tables are given instead. You can enter these tables manually as follows:

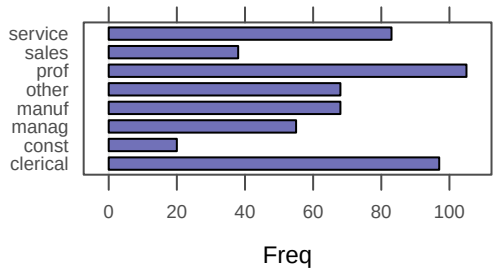
```
> myrace <- c( NW=67, W=467 )       # c for combine or concatenate
> myrace
 NW   W
 67 467

> mycrosstable <- rbind(               # bind row-wise
   NW = c(clerical=15, const=3, manag=6,  manuf=11, other=5,  prof=7, sales=3, service=17),
   W  = c(82,17,49,57,63,98,35,66)    # no need to repeat the column names
                         )
> mycrosstable
   clerical const manag manuf other prof sales service
NW       15     3     6    11     5    7     3      17
W        82    17    49    57    63   98    35      66
```

Replacing `rbind()` with `cbind()` will allow you to give the data column-wise instead.

2.6.10 Graphing Categorical Data

The `lattice` function `barchart()` can display these tables as barcharts.

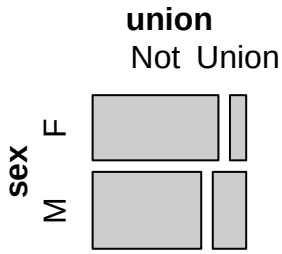
```
> barchart(xtabs(~ sector, CPS))
```




```
> barchart(xtabs(~ sector, CPS), horizontal=FALSE)  # vertical bars
```

Just as bar charts are used to display the distribution of one categorical variable, mosaic plots can do the same for cross tables. `mosaic()` (from the `vcd` package) is not a `lattice` plot, but it does use a similar formula interface.

```
> require(vcd)                          # load the visualizing categorical data package
> mosaic(~ sex + union, CPS)
```



Alternatively, we can send `mosaic()` the output of `xtabs()`:

> **Caution!**
> The `mosaic()` function has nothing to do with the `mosaic` package, they just happen to share the same name.

```
> mosaic(xtabs(~ sex + union, CPS))  # non-whites are more likely to be unionized
```

Or we can send our own hand-made table (although the output isn't quite as nice without some extra effort we won't discuss just now):

> Neither `mosaic()` nor the similar `mosaicplot()` are as clever as one could hope. In particular, without some extra customization, both tend to look bad if the levels of the variables have long names. `mosaic()` plots also always stay square.

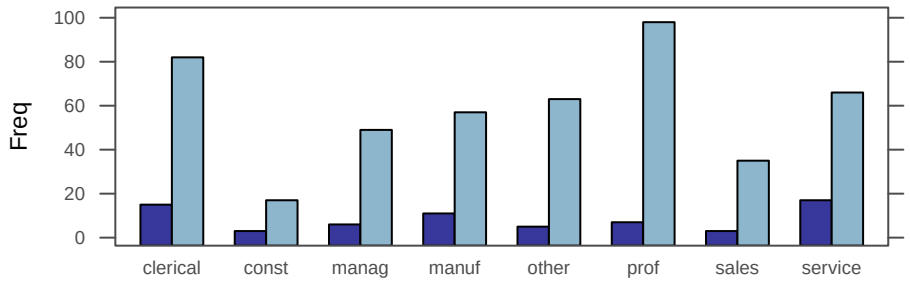
```
> mosaic(mycrosstable)
```

Barcharts can also be used to display two-way tables. First we convert the cross-table to a data frame. Then we can use this data frame for plotting.

```
> cps <- as.data.frame(xtabs(~ sector + race, data=CPS)); cps
```

```
     sector race Freq
1  clerical   NW   15
2     const   NW    3
3     manag   NW    6
4     manuf   NW   11
5     other   NW    5
6      prof   NW    7
7     sales   NW    3
8   service   NW   17
9  clerical    W   82
10    const    W   17
11    manag    W   49
12    manuf    W   57
13    other    W   63
14     prof    W   98
15    sales    W   35
16  service    W   66
```

```
> barchart(Freq ~ sector, groups=race, data=cps)
```



2.7 Additional Notes on R Syntax

2.7.1 Text and Quotation Marks

For the most part, text in R must be enclosed in either single or double quotations. It usually doesn't matter which you use, unless you want one or the other type of quotation mark *inside* your text. Then you should use the other type of quotation mark to mark the beginning and the end.

```
> text1 <- "Mary didn't come"           # apostrophe inside requires double quotes around text
> text2 <- 'Do you use "scare quotes"?'  # this time we flip things around
```

If you omit quotes, you will often see error messages telling you that R can't find an object because R will look for a function, data set or other object with that name instead of treating your text as text.

```
> text3 <- blah
Error: object 'blah' not found
```

2.7.2 Functions

Functions in R use the following syntax:

```
> functionname( argument1, argument2, ... )
```

- The arguments are always *surrounded by (round) parentheses* and *separated by commas.*
  - Some functions (like `col.whitebg()`) have no arguments, but you still need the parentheses.
- Most arguments have names, but you don't need to use the names *if you give the arguments in the correct order.*

  If you use names, you can give the arguments out of order. The following do the same thing,

  ```
  > xyplot(Sepal.Length ~ Sepal.Width, data=iris, groups=Species)
  > xyplot(Sepal.Length ~ Sepal.Width, iris, groups=Species)
  > xyplot(Sepal.Length ~ Sepal.Width, groups=Species, iris)
  ```

  But these do not work

  ```
  > xyplot(Sepal.Length ~ Sepal.Width, Species, iris)
  > xyplot(Sepal.Length ~ Sepal.Width, iris, Species)
  ```

  The first fails because the second argument is `data`, so `iris` needs to be in the second position if it is not named. The second fails because `groups` is not the third argument. (There are many other arguments between `data` and `groups` .) The documentation for functions shows the correct order of arguments.

- Typically, we will not use names for the first argument or two (these tend to be very important arguments that have to be there) but will use names for the rest (these are often optional arguments that can be present or not, depending on whether we want the default behavior or something special).

2.8 Installing R

2.8.1 RStudio in the cloud

Our primary version of R will be the online version of RStudio. You should have an RStudio account at `http://beta.rstudio.org/` using your Gmail address. RStudio is a brand new (and quite nice) interface to R that runs in a web browser. This has the advantage that you don't have to install or configure anything. Just login and you are good to go. Futhermore, RStudio will "remember" what you were doing so that each time you login (even on a different machine) you can pick up right where you left off. This is "R in the cloud" and works a bit like GoogleDocs for R.

If you find bugs or have suggestions for RStudio, let us know. It is in rapid development at the moment, and we can pass along your feedback to the developers.

This should be all you need for this course. But if you prefer to have a stand-alone version (because you study somewhere without an internet connection, or have data that can't be loaded into the cloud, for example), read on.

2.8.2 RStudio on Your Desktop/Laptop

There is also a stand-alone version of the RStudio environment that you can install on your desktop or laptop machine. This can be downloaded from `http://www.rstudio.org/`. This assumes that you have a version of R installed on your computer (see below for instructions to download this from CRAN).

2.8.3 Getting R from CRAN

CRAN is the Comprehensive R Archive Network (`http://cran.r-project.org/`). You can download free versions of R for PC, Mac, and Linux from CRAN. (If you use the RStudio stand-alone version, you also need to install R this way first.) All the instructions for downloading and installing are on CRAN. Just follow the appropriate instructions for your platform.

2.8.4 RStudio in the cloud on your own server

At present, we are using a beta version of RStudio running on their servers. It is also possible to install this on servers at your institution. This will almost certainly require a discussion with your administrators, but may be worthwhile to facilitate student use in this manner.

2.9 R Examples

The commands below are illustrated with the data sets `iris` and `CPS`. To apply these in other situations, you will need to substitute the name of your data frame and the variables in it.

`answer <- 42`	Store the value 42 in a variable named `answer`.
`log(123); log10(123); sqrt(123)`	Take natural logarithm, base 10 logarithm, or square root of 123.
`x <- c(1,2,3)`	Make a variable containing values 1, 2, and 3 (in that order).
`data(iris)`	(Re)load the data set `iris`.
`summary(iris$Sepal.Length)`	Summarize the distribution of the `Sepal.Length` variable in the `iris` data frame.
`summary(iris)`	Summarize each variable in the `iris` data frame.
`str(iris)`	A different way to summarize the `iris` data frame.
`head(iris)`	First few rows of the data frame `iris`.
`require(Hmisc)` `require(abd)`	Load packages. (This can also be done by checking boxes in the Packages tab.)
`summary(Sepal.Length~Species,data=iris,fun=favstats)`	Compute favorite statistics of `Sepal.Length` for each `Species`. [requires `Hmisc`]
`histogram(~Sepal.Length|Species, iris)`	Histogram of `Sepal.Length` conditioned on `Species`.
`bwplot(Sepal.Length~Species, iris)`	Boxplot of `Sepal.Length` conditioned on `Species`.
`xyplot(Sepal.Length~Sepal.Width|Species, iris)`	Scatterplot of `Sepal.Length` by `Sepal.Width` with separate panels for each `Species`.
`xtabs(~sector, CPS)`	Frequency table of the variable `sector`.
`barchart(xtabs(~sector, CPS))`	Make a barchart from the table.
`xtabs(~sector + race, CPS)`	Cross tabulation of `sector` and `race`.
`mosaic(~sector + race, CPS)`	Make a mosaic plot.
`xtData <- as.data.frame( xtabs(~sector + race, Trematodes) )`	Save cross table information as `xtData`.
`barchart(Freq~sector, data=xtData, groups=race)`	Use `xtData` to make a segmented bar chart.
`sum(x)`; `mean(x)`; `median(x)`; `var(x)`; `sd(x)`; `quantile(x)`	Sum, mean, median, variance, standard deviation, quantiles of `x`.

2.10 Exercises

2.1. Calculate the natural logarithm (log base e) and base 10 logarithm of 12,345.

What happens if you leave the comma in this number?

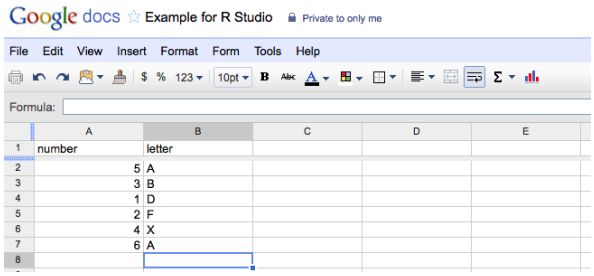
```
> log(12,345)
[1] 0.425
```

2.2. Install and load the `mosaic` package. Make sure `lattice` is also loaded (no need to install it, it is already installed).

Here are some other packages you may like to install as well. following packages:

- `Hmisc` (Frank Harrell's miscellaneous utilities),
- `vcd` (visualizing categorical data),
- `fastR` (*Foundations and Applications of Statistics*), and
- `abd` (*Analysis of Biological Data*).

2.3. Enter the following small data set in an Excel or Google spreadsheet and import the data into RStudio.



	A	B	C	D	E
1	number	letter			
2	5	A			
3	3	B			
4	1	D			
5	2	F			
6	4	X			
7	6	A			
8					

You can import directly from Google. From Excel, save the file as a csv and import that (as a text file) into RStudio. Name the data frame `JunkData`.

2.4. What is the average (mean) *width* of the sepals in the `iris` data set?

2.5. Determine the average (mean) sepal width for each of the three species in the `iris` data set.

2.6. The `Jordan8687` data set (in the `fastR` package) contains the number of points Michael Jordan scored in each game of the 1986–87 season.

a) Make a histogram of this data. Add an appropriate title.

b) How would you describe the shape of the distribution?

c) In approximately what percentage of his games, did Michael Jordan score less than 20 points? More than 50? (You may want to add `breaks=seq(0,70,by=5)` to your command to neaten up the bins.)

2.7. Cuckoos lay their eggs in the nests of other birds. Is the size of cuckoo eggs different in different host species nests? The `cuckoo` data set (in `fastR`) contains data from a study attempting to answer this question.

a) When were these data collected? (Use `?cuckoo` to get information about the data set.)

b) What are the units on the length measurements?

c) Make side-by-side boxplots of the length of the eggs by species.

d) Calculate the mean length of the eggs for each host species.

e) What do you think? Does it look like the size is differs among the different host species? Refer to your R output as you answer this question. (We'll learn formal methods to investigate this later in the semester.)

2.8. The `Utilities2` data set in the `mosaic` package contains a number of variables about the utilities bills at a residence in Minnesota over a number of years. Since the number of days in a billing cycle varies from month to month, variables like `gasbillpday` (`elecbillpday`, etc.) contain the gas bill (electric bill, etc.) divided by the number of days in the billing cycle.

a) Make a scatter plot of `gasbillpday` vs. `monthsSinceY2K` using the command

```
> xyplot(gasbillpday ~ monthsSinceY2K, data=Utilities2, type='l')   # the letter l
```

What pattern(s) do you see?

b) What does `type='l'` do? Make your plot with and without it. Which is easier to read in this situation?

c) What happens if we replace `type='l'` with `type='b'`?

d) Make a scatter plot of `gasbillpday` by `month`. What do you notice?

e) Make side-by-side boxplots of `gasbillpday` by `month` using the `Utilities2` data frame. What do you notice?

Your first try probably won't give you what you expect. The reason is that month is coded using numbers, so R treats it as numerical data. We want to treat it as categorical data. To do this in R use `factor(month)` in place of `month`. R calls categorical data a **factor**.

f) Make any other plot you like using this data. Include both a copy of your plot and a discussion of what you can learn from it.

2.9. The table below is from a study of nighttime lighting in infancy and eyesight (later in life).

	no myopia	myopia	high myopia
darkness	155	15	2
nightlight	153	72	7
full light	34	36	3

a) Recreate the table in RStudio.

b) What percent of the subjects slept with a nightlight as infants?

There are several ways to do this. You could use R as a calculator to do the arithmetic. You can save some typing if you use the function `prop.table()`. See `?prop.table` for documentation. If you just want row and column totals added to the table, see `mar_table()` in the `vcd` package.

c) Make a mosaic plot for this data. What does this plot reveal?

3 Getting Interactive with manipulate

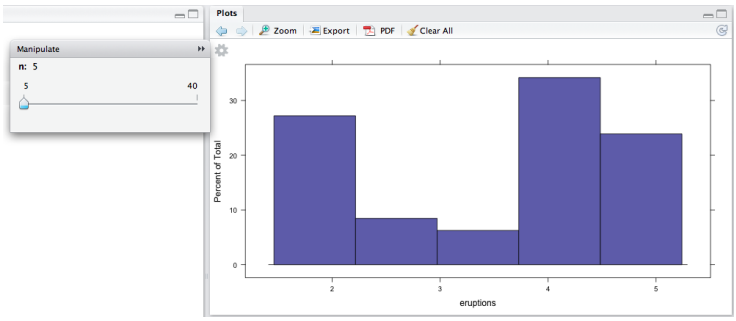
One very attractive feature of RStudio is the `manipulate()` function, which can allow the creation of a set of controls (such as `slider()`, `picker()` or `checkbox()`) that can be used to dynamically change values within the expression. When a value is changed using these controls, the expression is automatically re-executed and redrawn. This can be used to quickly prototype a number of activities and demos as part of a statistics lecture.

3.1 Simple Things

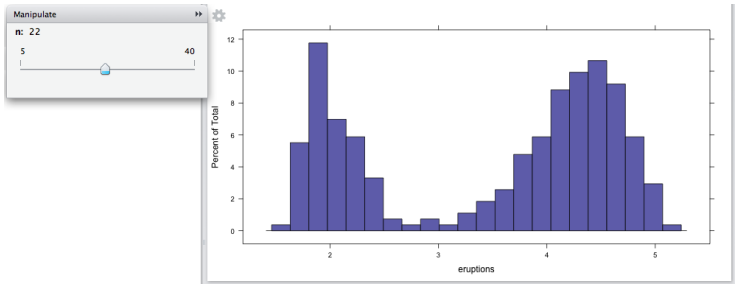
3.1.1 Sliders

```
> if(require(manipulate)) {
   manipulate(
         histogram( ~ eruptions, data=faithful, n=n),
         n = slider(5,40)
   )
 }
```

This generates a plot along with a slider ranging from 5 bins to 40.

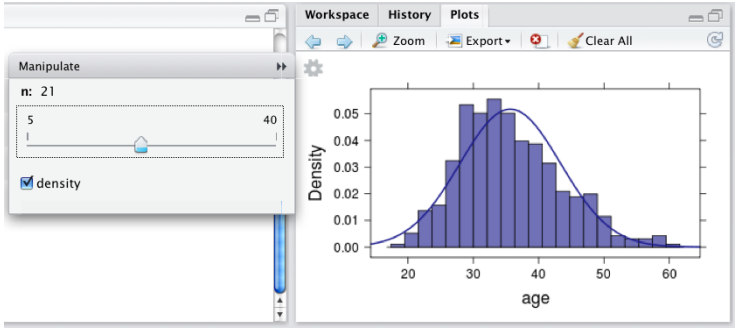


When the slider is changed, we see a clearer view of the eruptions of Old Faithful.



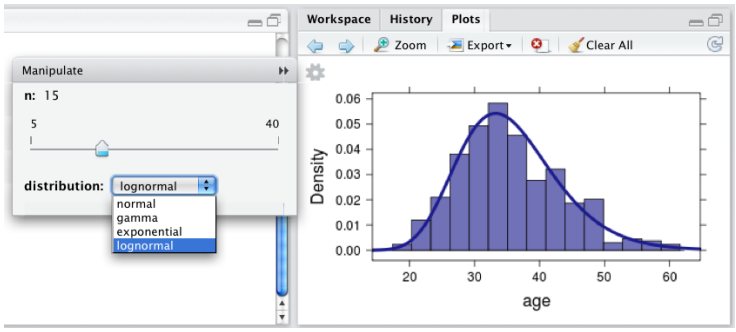
3.1.2 Check Boxes

```
> if(require(manipulate)) {
   manipulate(
         xhistogram( ~ age, data=HELP, n=n, density=density),
         n = slider(5,40),
         density = checkbox()
   )
 }
```



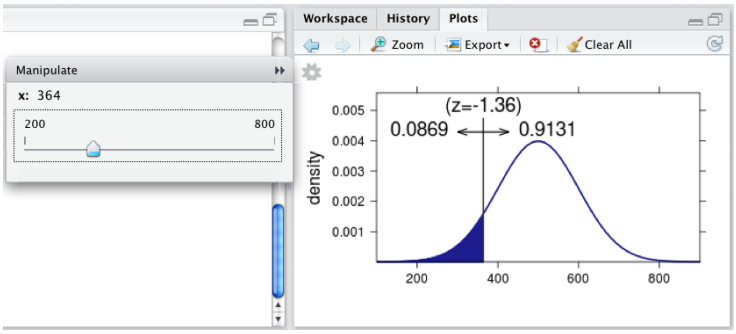
3.1.3 Drop-down Menus

```
> if(require(manipulate)) {
   manipulate(
         xhistogram( ~ age, data=HELP, n=n, fit=distribution, dlwd=4),
         n = slider(5,40),
         distribution =
                 picker('normal', 'gamma', 'exponential', 'lognormal',
                         label="distribution")
   )
 }
```



3.1.4 Visualizing Normal Distributions

```
> if(require(manipulate)) {
   manipulate( xpnorm( x, 500, 100 ), x = slider(200,800) )
 }
```



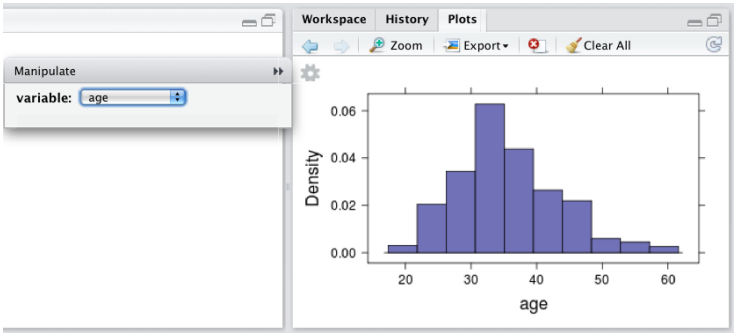
Exercises

3.1. The following code makes a scatterplot with separate symbols for each sex.

```
> xyplot(cesd ~ age, data=HELP, groups=sex)
```

Build a `manipulate` example that allows you to turn the grouping on and off with a checkbox.

3.2. Build a `manipulate` example that uses a picker to select from a number of variables to make a plot for. Here's an example with a histogram:



3.3. Design your own interactive demonstration idea and implement it using RStudio `manipulate` tools.

4

Multivariate Statistics – Early?

If you are of a certain age, you may remember the 1960s television show *Lost in Space*. One of the characters, the Robot, was often assigned guard duty. Robot would march back and forth, just like a soldier on guard duty.

But why? Soldiers are ordered to march back and forth so that they won't fall asleep; walking forces them to maintain a certain attention to their duty. Robot has no such need; his sensors and circuits can reliably detect intruders without the marching. Of course, the television viewers wouldn't know this about robots. Using the robot in the manner of a soldier was a way to introduce new technology to people with old mind-sets.

Now fast forward from the television of the 1960s to the classroom of the 21st century. Students have computers. They use statistical software packages to do calculations. But what calculations are they doing? Sample means, sample proportions, differences between means.

This is Robot walking back and forth. A way to use new technology with old mind-sets. But it's the professors, not the students, with the old mind-sets. The students are open to new things. They don't need to know that, once upon a time, it was a feat to get a machine to add up a column of numbers.

We professors were educated at a time when the tools for inverting a matrix were paper and pencil, when doing least squares problem involved a strange thing called a "pseudo-inverse" that you might learn in your fourth or fifth semester of university-level mathematics. But, now, least squares problems are no more difficult or time consuming to the human than square roots or addition. We just have to learn to use the tools in the right way.

Or, rather, we have to show our students what are the basic operations that are important for statistical reasoning in an age of modern computation. Not marching back and forth, like robot soldiers, computing sums of columns of numbers, but thinking about how to model the potentially rich interplay among multiple variables.

The standard approach to introductory statistics is based on a few simple operations: adding, squaring, and square-rooting to finding means and standard deviations; counting to find proportions and medians/quantiles. These operations are (supposed to be) familiar to students from high-school mathematics.

Here we'll examine the consequences of adding in a new basic operation that is not in the traditional high-school curriculum, but which is actually quite consistent with the "Common Core" curriculum being introduced by many U.S. states.[Rep10]

The operation is fitting multivariate linear models. Modern software makes it no more arduous than

finding a mean or standard deviation. Our emphasis here will be on how to introduce the conceptual foundations — using just high-school math and simple extensions largely specified already in the "Common Core" — that make the concept of modeling accessible.

There are three important pedagogical advantages to using multivariate linear models as a foundation for statistics:

1. It provides a logically coherent framework that unifies many of the disparate techniques found in introductory statistics books.
2. It allows the very important ideas of confounding and covariates to be introduced in an integrated way, showing students not only the perils of confounding, but what to do about it (and when you can't do anything).
3. It allows the examples used in classes to be drawn from a richer set of situations, and provides scope for students to express their creativity and insight. This can increase student motivation.

Of course, it's also important that modern statistical work is already and increasingly shaped by multivariate methods. For an example of how over the last few decades statistical methods used in the literature have shifted away from the curriculum of the traditional, non-multivariate introductory course, see the review of statistics practices in the *New England Journal of Medicine*. [HS05]

4.1 The Mathematical Foundations

A standard part of the high-school curriculum is the equation of a straight line: $y = mx + b$. Many students will recognize that m is a slope and b is the y-intercept.

It's helpful to move them a bit beyond this:

- Emphasize the "function" concept. A function, of course, is a relationship between an input and an output. Generalize this, so that students are comfortable with using names other than x as the input and y as the output. For example,

$$\text{height} = f(\text{age}) = 3\ \text{age} + 20$$

  .

- Introduce the idea of functions of non-numeric variables, for example:

$$\text{height} = g(\text{sex}) = \begin{cases} 67 & \text{for sex = female} \\ 71 & \text{for sex = male} \end{cases}$$

- Generalize the idea of a function to include having more than one input, for instance

$$\text{height} = h(\text{age}, \text{sex}) = -2\ \text{age} + \begin{cases} 67 & \text{for sex = female} \\ 71 & \text{for sex = male} \end{cases}$$

- De-program your students from thinking that there is one, and only one, correct formula for a relationship. Introduce the idea of a function as a description of a relationship and that there can be different descriptions of the same relationship, all of which can be right in some ways. Different descriptions can include some details and exclude others. Such descriptions are called "models." The above models $f(\text{age})$, and $g(\text{sex})$ and $h(\text{age}, \text{sex})$ give different outputs for the same inputs. For example, for a male of age 10, according to the models, the height is variously $f(10) = 50$ or $g(\text{male}) = 71$ or $h(10, \text{male}) = 51$.

- Many useful models don't give exactly the right output for every case. Not every 10-year old male has the same height. The difference between what a model says, and what the value is for an actual case, will not in general be zero. The "residual" is the difference between the actual value for a given 10-year old male, say Ralph, and what a given model says about 10-year old males in general. The residuals give important information about how good a model is.

4.2 The Language of Models

There is a language for defining models that is different from writing down an algebraic formula.

You have already seen some aspects of this notation in graphics commands, e.g. `height ~ sex`. You can read this in any of several ways:

- Break down `height` by `sex` (as in a boxplot)
- `height` versus `sex`
- Model `height` as a function of `sex`.

In this section, you'll see more complicated models, involving multiple variables. This makes it worthwhile to assign more precise labels to the different components of the modeling language, which has just a few components:

- The response variable. This is the output of the model function, `height` in the above examples.
- The explanatory variables. These are the inputs to the model function: `age` and `sex` in the above examples.
- Model terms. Examples of model terms are explanatory variables themselves and a special term called the "intercept." There are a few others, but we'll deal with them as we come to them.

(A more comprehensive introduction is given in Chapters 4 and 5 of [Kap09].)

As an example, consider this simple formula:

$$z = 7 + 3x + 4.5y.$$

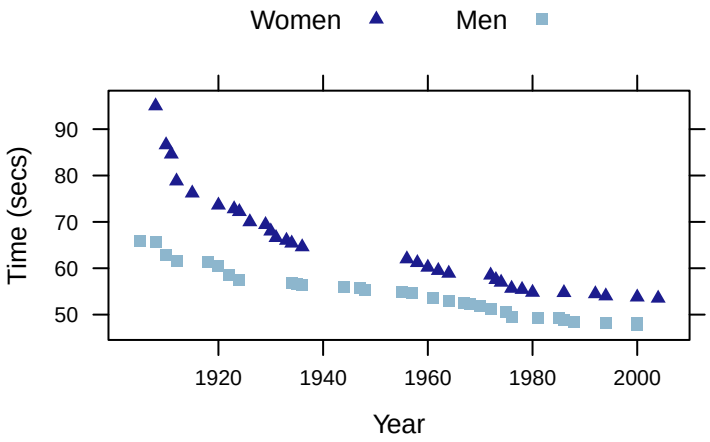
The corresponding model description consists of the response variable z, and three model terms: the explanatory variables x and y and the intercept term. In the modeling language, it would be written:

$$\mathrm{z} \sim 1 + \mathrm{x} + \mathrm{y}$$

Notice that instead of the $=$ sign, we are using the $\sim$ sign. Also, notice that the specific coefficients 7, 3, and 4.5 are missing. The model description is a kind of skeleton for describing the shape of the model. It's like saying: "We want a straight line model," rather than giving the complete specification of the formula.

The process of finding a specific formula to make a model match the pattern shown by data is called "fitting the model to the data."

To see how different model specifications correspond to different "shapes" of models, consider the history of world-record times in the 100-meter free-style swimming race.

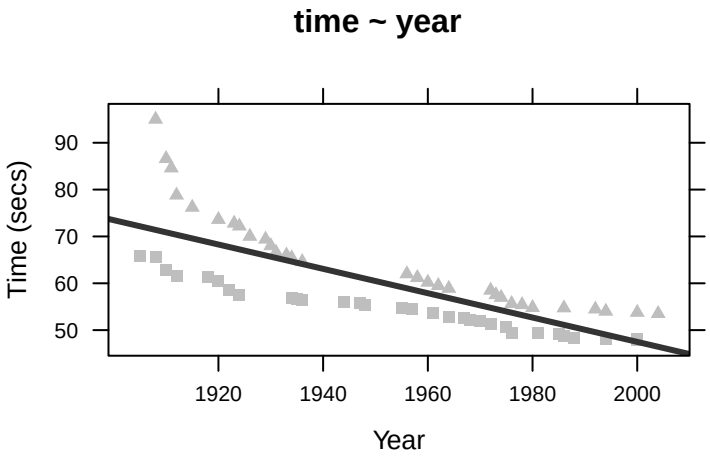


You can see the steady improvement in records over the decades from 1900 to the present. Men's times are somewhat faster than women's.

Now to build some models.

4.2.1 `Time` ∼ `1` + `Year`

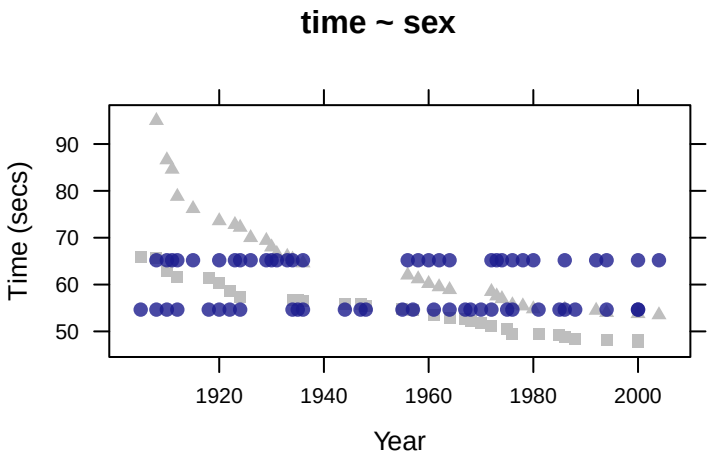
The model `time` ∼ 1 + `year` gives the familiar straight-line form: the intercept term (written simply as 1) and a term corresponding to the linear dependence on `year`.



This model captures some of the pattern evident in the data: that swimming times are improving (getting shorter) over the years. And it ignores other obvious features, for example the difference between men's and women's times, or the curvature reflecting that records are not improving as fast as they did in the early days.

4.2.2 Time ∼ Sex

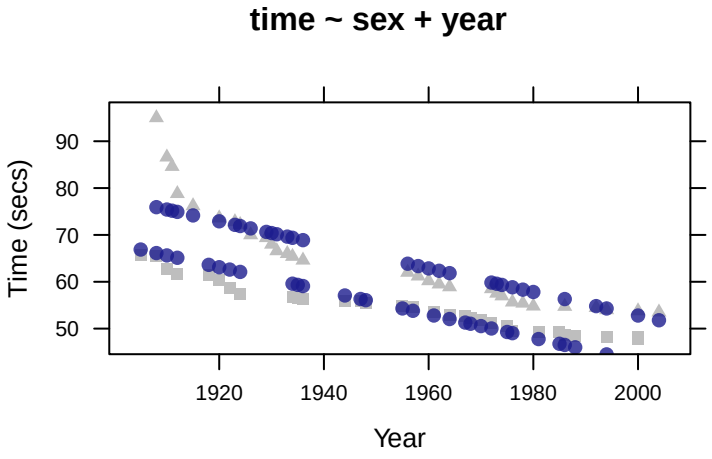
The model time ∼ sex breaks down the swimming times according to sex:



This model reflects the typical difference between men's and women's times. It's oblivious to the trend that records improve over the years. Why? Because the variable year was not included in the model.

4.2.3 Time ∼ Sex+Year

The record time evidently depends both on sex and year, so it's sensible to include both variables in the model

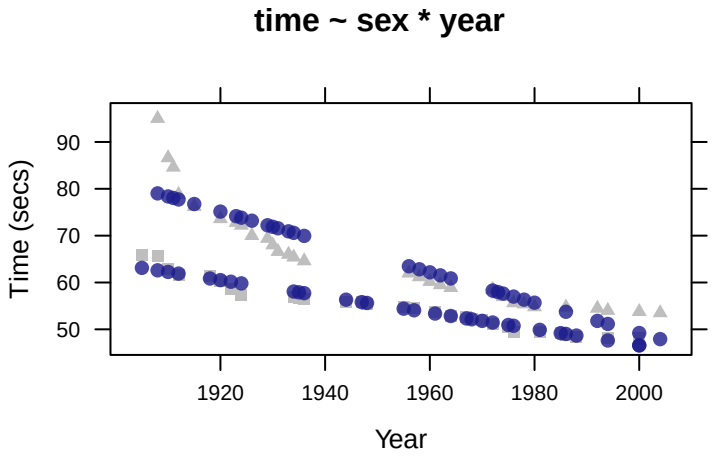


This is a straight-line model with separate lines for the different sexes. The intercept is different for the different sexes, but the slope is the same. This model is reflects the typical difference between men's and women's times.

Students sometimes observe that the function generated by fitting this model doesn't respect the "vertical line test" taught in high-school algebra. This is a good time to remind students that this is a function of *two* variables. It is indeed a function, and for any specific value of the inputs sex and year gives a single value.

4.2.4 `Time` ∼ `Sex`+`Year` + `Sex`:`Year`

There are two ways that the previous model, `time` ∼ `sex`+`year`, misses obvious features in the data: there is no curvature over the years and the slopes are exactly the same for men and women. To construct a model with different slopes for men and women requires that we add a term that combines both `sex` and `year`. Such a term is constructed with the syntax `sex`:`year` (or, what would amount to the same thing, `year`:`sex`).



In statistics, such a term is called an *interaction term*. (In mathematics, it's often called a *bi-linear term*.) It's the term that lets you have different slopes for the different sexes.

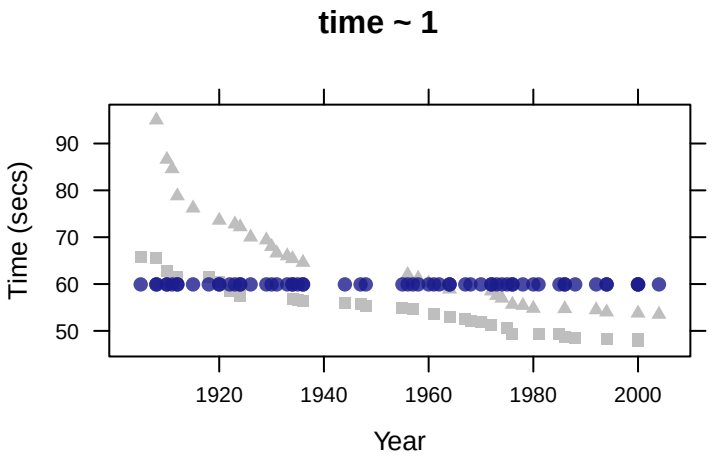
The new phrase "interaction term" creates a need for a retronym, a way to refer to those simple, non-interaction terms that we started with, like `sex` and `year`. (Common retronyms in everyday life are acoustic guitar, snail-mail, World War I, cloth diaper, and whole milk, compound terms that weren't needed until electric guitars, e-mail, disposable diapers, and skim milk were introduced, and World War II showed that the "War to End All Wars" was mis-named.)

The standard terminology for terms like `sex` and `year` is unfortunate: "main effect." It suggests that interaction terms play a lesser role in modeling. This is a bad attitude, since sometimes the interaction is exactly what you're interested in, but the terminology seems enshrined by statistical tradition.

Very often when you are including an interaction term, you want to include the main effects as well. There is a convenient shorthand for this: `time` ∼ `sex`*`year`

4.2.5 The Intercept Only: `Time` ∼ `1`

It's also possible to have models that have no explanatory variables whatsoever. Just the intercept term appears to the right of the model formula.



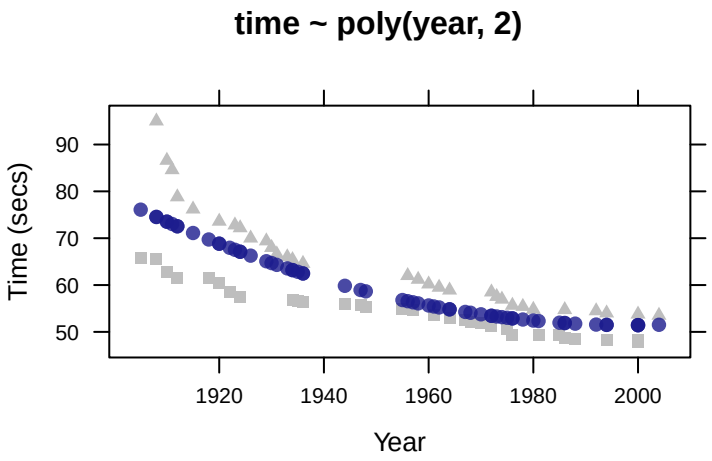
As you might expect, by leaving out both `sex` and `year` from the model, it doesn't reflect the role of either variable in any way. But the model `time` $\sim$ 1 does get one thing very well: the typical record time.

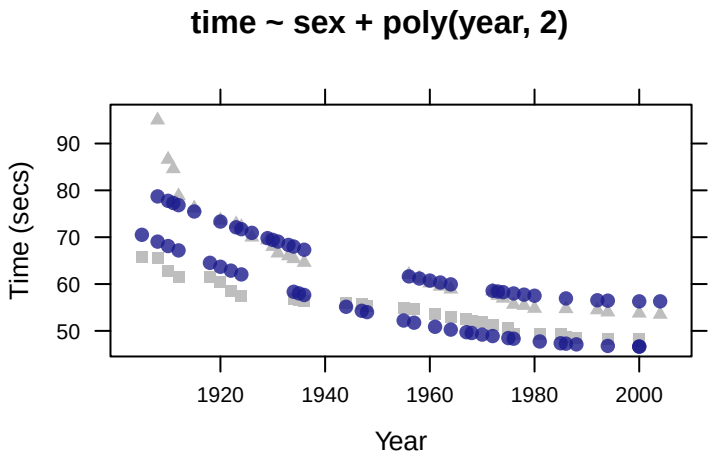
Think of `time` $\sim$ 1 as saying "all the cases are the same." In some ways, it's analogous to the model `time` $\sim$ `sex`. That model says that "all men are the same, and all women are the same." So the difference between `time` $\sim$ 1 and `time` $\sim$ `sex` is just like the difference between a "grand mean" and a "group mean."

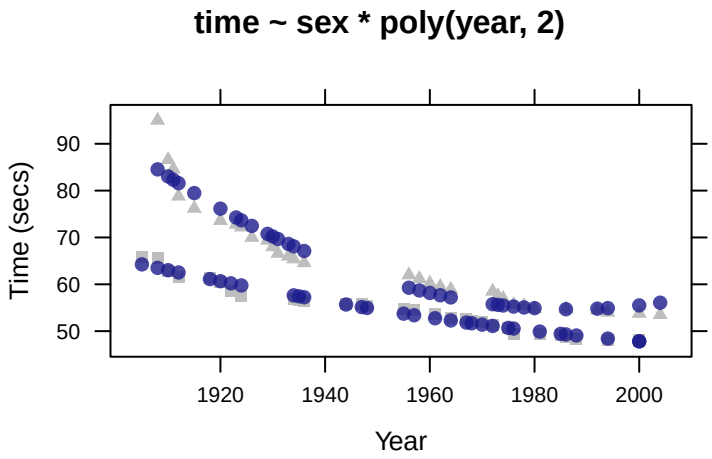
4.2.6 Transformation Terms

You can construct more complicated models by adding in more explanatory variables. (Improved swimming gear? Better training? Refinements in technique?). You can also add in additional terms with more structure. There is a rich variety of ways to do this.

Since many students are familiar (or at least remember vaguely) the idea of quadratics and polynomials, they might be interested to see that the modeling language can handle this. Here are three different models involving a polynomial dependence on `year`:







Although these models reflect more "detail" in the data, namely the curvature, they do it in a way that ultimately does not make sense in terms of the "physics" of world records. Notice how the upward facing parabolas eventually produce a pattern where the record times increase over the years.

There are other sorts of nonlinear terms that might be more appropriate for modeling this sort of data. Exponentials, square roots, etc., even piecewise linear or bent-line forms are all possible within the modeling framework.

4.3 Finding Formulas from Data

Behind the graphical depictions of the models shown in the previous section is a process for finding specific numerical formulas for the models. The software to do this is packaged into the `lm( )` operator and is very easy for students and professionals alike. The human work, and what students need to learn, is not the mechanics of fitting models, but the interpretation of them. A good place to start is with the interpretation of model coefficients.

To illustrate, consider the actual swimming records data employed in the previous examples. This data set is available via the internet, and can be loaded into R with a command like the following:

```
> swim = read.csv("http://www.macalester.edu/~kaplan/ISM/datasets/swim100m.csv")
```


As before, we'll model the world-record time as a function of sex and year.

Here's the very simplest model: all cases are modeled as being the same.

```
> lm( time ~ 1, data=swim )

Coefficients:
(Intercept)
       59.9
```

The fundamental purpose of `lm( )` is to find the coefficients that flesh out the skeleton provided by the model description. For this simple model, the coefficient works out to be the mean of the record times:

```
> with( data=swim, mean(time) )

[1] 59.9
```

Models are more interesting that contain actual explanatory variables. Here's one using the quantitative variable year:

```
> lm( time ~ year, data=swim )

Coefficients:
(Intercept)         year
     567.24        -0.26
```

The coefficients now are the slope and intercept of the straight-line relationship: time $= -0.2599\text{year} + 567.242$.

There's a similar story with categorical variables, like sex:

```
> lm( time ~ sex, data=swim )

Coefficients:
(Intercept)         sexM
       65.2        -10.5
```

Based on the result of the simple all-cases-the-same model time ~ 1, you might suspect that the coefficients are the group means. That's close to being right. Here are the group means:

```
> mean( time ~ sex, data=swim )

  sex    S  N Missing
1   F 65.2 31       0
2   M 54.7 31       0
3 ALL 59.9 62       0
```

The coefficients of the linear model are organized differently than simple group means. Notice that there is a sexM coefficient, but no similarly named coefficient for women. Instead, there is the intercept coefficient. This corresponds to the mean time of one of the groups: the reference group. In this case, the reference group is women.

The other coefficient, sexM tells the *difference* between the mean of the reference group and the mean for the men. In other words, the coefficients are arranged in intercept-slope form, but for categorical variables the coefficient isn't a slope but a finite-difference.

It's important to keep in mind the intercept-slope/difference format when interpreting the coefficients from models with multiple explanatory variables:

```
> lm( time ~ sex + year, data=swim )

Coefficients:
(Intercept)         sexM         year
    555.717       -9.798       -0.251
```

In the MOSAIC package, the syntax of familiar operators like `mean( )`, `sd( )`, etc. has been extended so that the modeling notation can be used to calculate group-by-group values. Our goal is to make is straightforward to transition from conventional basic stats to modeling, by getting students used to the $\sim$ and `data=` notation early. You can even do `mean( time 1, data=swim )`. So you can talk about models in a systematic way even if all you want to cover is means, proportions, medians, etc.

The intercept is so important that the `lm( )` operator includes it by default, even if you don't specify it explicitly in the model design. In those rare cases when you don't want an intercept term, use the notation `-1`

As usual, there is an intercept coefficient. It's meaning is a little bit subtle: it is the intercept for the reference group (in these data, women). The coefficient on `year` is a slope. The `sexM` coefficient is a difference: it's how the intercept differs for group `M` from the reference group.

In traditional mathematical notation, the model formula corresponding to these coefficients is

$$\text{time} = -0.2515\ \text{year} + \begin{cases} 555.7168 & & \text{for women} \\ 555.7168 & -9.7980 & \text{for men} \end{cases}$$

One of the great strengths of R comes from the ability to carry out new computations on the results from other computations. To illustrate, give a name to a fitted model.

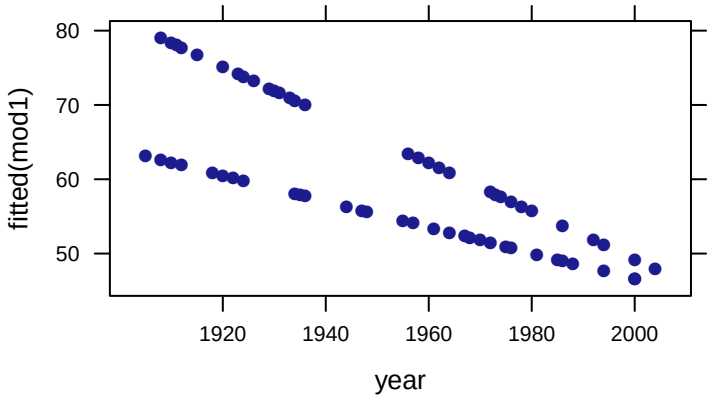
```
> mod1 = lm( time ~ year + sex + year:sex, data=swim )
```

From the model, you can now compute additional information. Important operators for demonstrating basic properties of models are `resid( )`, `fitted( )`, and `predict`.

```
> mean( resid(mod1) )
[1] -1.83e-17
> sd( resid(mod1) )
[1] 3.24
> var( fitted(mod1) )/var(swim$time) # R-squared
[1] 0.893
> sum( fitted(mod1)^2 ) # sum of squares of the fitted
[1] 227996
> sum( resid(mod1)^2 ) # sum of squares of the residuals
[1] 639
```

For plotting the fitted model values

```
> xyplot( fitted(mod1) ~ year, data=swim )
```



The model suggests that women's times will soon break those of men. To evaluate the model for new values of inputs, use `predict( )`.

```
> predict( mod1, newdata=data.frame(year=2020, sex="F") )
   1
42.7
> predict( mod1, newdata=data.frame(year=2020, sex="M") )
```

```
   1
43.1
```

When you get to the stage where you want to talk about statistical inference, you can of course show bootstrapping and permutation tests. For example for bootstrapping standard errors:

```
> s = do(100)* lm( time ~ year + sex + year:sex, data=resample(swim) )
> sd(s)
Intercept      year      sexM year:sexM     sigma r-squared
  71.1102    0.0361   69.7635    0.0354    0.7721    0.0299
```

And, of course, you can do the conventional theoretical calculations for inference.

- Confidence intervals, for instance at a 95% level

```
> confint(mod1, level=0.95)
                2.5 %   97.5 %
(Intercept)  618.7910  775.811
year          -0.3643   -0.284
sexM        -415.3840 -189.544
year:sexM      0.0921    0.208
```

- Analysis of variance

```
> anova(mod1)
Analysis of Variance Table

Response: time
          Df Sum Sq Mean Sq F value  Pr(>F)
year       1   3579    3579   324.7 < 2e-16 ***
sex        1   1484    1484   134.7 < 2e-16 ***
year:sex   1    297     297    26.9 2.8e-06 ***
Residuals 58    639      11
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

- The regression report and other statistics on the model

```
> summary(mod1)

...
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  697.3012    39.2214   17.78  < 2e-16 ***
year          -0.3240     0.0201  -16.12  < 2e-16 ***
sexM        -302.4638    56.4116   -5.36  1.5e-06 ***
year:sexM      0.1499     0.0289    5.19  2.8e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.32 on 58 degrees of freedom
Multiple R-squared: 0.893,        Adjusted R-squared: 0.888
F-statistic:  162 on 3 and 58 DF,  p-value: <2e-16
```

4.4 Example: Genetics before Genes

To emphasize the flexibility that multivariate models gives in asking questions of statistical interest, let's consider a problem of historical significance: Francis Galton's attempt to quantify the heritability of height.

Galton famously developed the correlation coefficient in the late 1880s (see [Gal88].) One of his motivations was to put on a quantitative footing the theory of evolution established by his half-cousin, Charles Darwin. It's important to realize that Darwin himself did not know the mechanism by which traits were inherited. He imagined a particle of inheritance which he called a "gemmule." The words "gene" and "genetics" date from the first decade of the 20th century, the same decade when William Gossett started publishing under the pseudonym "Student." It wasn't until 1944 when DNA was shown to be associated with genetic heritability.

Without a mechanism of genotype, Galton needed to rely on what we now call "phenotype," the observable characteristics themselves. The `mosaic` package includes the `Galton` dataset, transcribed to a modern format, of measurements that Galton himself collected on the heights of adult children and their parents.

```
> data(Galton)
> head(Galton)
```

```
  family father mother sex height nkids
1      1   78.5   67.0   M   73.2     4
2      1   78.5   67.0   F   69.2     4
3      1   78.5   67.0   F   69.0     4
4      1   78.5   67.0   F   69.0     4
5      2   75.5   66.5   M   73.5     4
6      2   75.5   66.5   M   72.5     4
```

By today's standards, Galton's techniques were quite limited, but did suffice to quantify in some ways the heritability of height. Galton examined, for the boys, the correlation between the height of the "mid-parent" and the boy's height. If Galton had R available (rather than having to invent r!), he might have done the following calculation:

```
> Galton$midparent = (Galton$father + 1.08*Galton$mother)/2
> boys = subset(Galton, sex=="M")
> with( boys, cor(midparent,height))
```

```
[1] 0.486
```

This is purely speculation, but it seems unlikely that Galton, with his interests ranging from exploring Namibia to fingerprints to meteorology (and, it must be mentioned, eugenics), would have restricted himself to a correlation between the mid-parent and boy's heights. Might height be inherited differently from the mother and the father? Is the same mechanism at work for girls as for boys? Does one parent's height have a potentiating effect on the influence of the other parent's height?

Presumably, Galton would have constructed more detailed descriptions of the relationship between a child's adult height and the genetic and environmental influences, rather than focusing on only the parent's height. Even with the few variables in Galton's own dataset, one can use models to explore hypothesized relationships.

By all means, use Galton's data to illustrate simple modeling techniques, e.g.,

```
> lm( height ~ father + sex, data=Galton )
```

```
Coefficients:
(Intercept)       father         sexM
     34.461        0.428        5.176
```

But consider going further and using inferential methods to see what evidence is contained in Galton's data more detailed descriptions. For instance, in the relatively simple model

```
> summary( lm( height ~ sex + father + mother + nkids, data=Galton) )
```

```
...
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  16.1877     2.7939    5.79  9.5e-09 ***
sexM          5.2099     0.1442   36.12  < 2e-16 ***
```



```
father        0.3983     0.0296   13.47  < 2e-16 ***
mother        0.3210     0.0313   10.27  < 2e-16 ***
nkids        -0.0438     0.0272   -1.61     0.11
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.15 on 893 degrees of freedom
Multiple R-squared: 0.641,        Adjusted R-squared: 0.639
F-statistic:  398 on 4 and 893 DF,  p-value: <2e-16
```

Ask your students to investigate, through modeling, whether the influence of the parents is different for the different sexes, or whether what looks like an even split of influence between the father and mother could all, in fact, be attributed to the mother.

4.1. DRAFT Outline for a swim-data modeling problem: construct a post-war variable and add it in. Several ways to do this: explore which one's give models that are most satisfactory to you:

- postwar = year $>$ 1945
- interaction with postwar and year.
- pmax(year - 1945, 0)

4.2. Consider the `CPS` data that gives per-hour wages and other measurements for workers in the mid-1980s. Is there evidence in the data for discrimination on the basis of race and/or sex? What covariates might one reasonably adjust for in measuring the difference between the races or the sexes?

5

Computing by Hand

R is a valuable tool, but for an instructor it is just one tool among many for developing students' skills and understanding. In this section, we illustrate some teaching strategies we have found effective that involve students doing the calculations themselves. In some cases, these are actual paper-and-pencil calculations. In others, we take a calculation or method apart, using the computer as if it were a set of hand tools and having the student build the operation. Still other examples are kinesthetic in nature: giving students a physical intuition about the meaning of an operation. And in others, we use the students in class collectively to illustrate variation and its meaning.

Since this is a book about using the computer in teaching, it may seem odd to include "by hand" methods. One reason we do it is to emphasize that the broad goal is to teach statistics and statistical skills and concepts. But another reason has directly to do with building computing skills and the attitudes that support the skilled us of computing. Among these are:

1. The computer is not magical. It's doing the same things you can do yourself, but with greater speed and precision and with much, much greater toleration of tedious repetition.
2. That complicated operations are constructed by putting together simpler operations.
3. That you should be able to open the "black box" and confirm that the results the computer is giving you make sense.

6 Simulation Based Inference

6.1 Simulation and Randomization with the `mosaic` Package

Software environments almost always provide an important feature:

1. They allow potentially complicated operations to be packaged with a simple interface, making them easy to use.

An environment that is integrated with a programming language provides an additional capability:

2. New operations can be constructed from existing ones.

Any proper statistics environment should offer the ease of use of (1). R, like many packages, offers a large set of pre-packaged operations. But, as a modern programming language, R offers the advantage of making (2) available as well as (1). In addition to enabling R to provide ready access to new forms of state-of-the-art computing, the programming-language features of R also enable the operations in (1) to be de-constructed and presented to students in a transparent and intelligible way that reveals the underlying logic.

One of the important goals of the `mosaic` package is to provide elementary commands that can be easily strung together by novices without having to master the esoteric aspects of programming. This chapter will describe a few such key operations and how they connect to one another: random sampling and resampling, replication of random trials, summarizing the results of multiple trials. As you will see, the `mosaic` operations allow students to implement each of the operations in what George Cobb calls the "3 Rs" of statistical inference: Randomization, Replication, and Rejection [Cob07]. By putting the 3 Rs together in various ways, students learn to generalize and internalize the logic of inference, rather than just following formulaic methods. More examples of the use of R as this sort of *sandbox* for experimentation can be found in [HBQ04].

SUGGESTION BOX
We are interested in your feedback while we develop these tools.

There's an interesting discussion of the role of simulation in [Spe11], where he notes the changing role of simulation. It used to be:

> something that people did when they can't do the math. ... It now seems that we are heading into an era when all statistical analysis can be done by simulation.

6.1.1 Sampling From Day 1 with `rflip()`

Since tossing a coin is one of the most familiar examples of randomness, we've added to `mosaic` a function that facilitates simulating coin tosses.

```
> rflip(20)
Flipping 20 coins [ Prob(Heads) = 0.5 ] ...

T H T H H T H H H T H T H H T H T T T H

Result: 11 heads.
> as.numeric(rflip(20))       # just count how many heads
[1] 13
```

In addition to the Lady Tasting Tea activity (Section 1.2.1), other activities can be devised to help students develop a sense for randomness and variability.

1. If you flip a coin 100 times, how often will you get exactly 50 heads? Between 45 and 55 heads? Between 40 and 60 heads?

2. If you flip a coin 20 times, how long is the longest run of either heads or tails typically?

   This can be part of demonstration where you have students write down strings of H and T that "look random" and another set based on actual coin tosses. Then see if you can tell them which are which.

3. `rflip()` allows you to flip biased coins as well. Let's simulate a 90% free throw shooter shooting 10 free throws.

   ```
   > rflip(10, prob=.9)
   Flipping 10 coins [ Prob(Heads) = 0.9 ] ...

   H H H H H H H H H H

   Result: 10 heads.
   ```

   It's the end of the game. Your team is down by one point. Free Throw Freddie, a 90% free throw shooter, has been fouled while shooting. What are the chances that your team wins (because Freddie makes 2 shots in a row), loses (because he misses both shots), or has to play an overtime period (because he makes one of the two)?

   ```
   > simulated.shots <- do(2000) * rflip(2, prob=.90)
   > with( simulated.shots, table(heads) )
   heads
      0    1    2
     14  371 1615
   ```

4. It can be fun to have students compare the results of flipping, spinning, and tipping pennies. Many are surprised to learn that these do not all have same probability of producing heads.

6.1.2 Sampling and Resampling

Arguably, the most important operation in statistics is sampling: ideally, selecting a random subset from a population. Regrettably, sampling takes work and time, so instructors tend to de-emphasize the actual practice of sampling in favor of theoretical descriptions. What's more, the algebraic notation in which much of conventional textbook statistics is written does not offer an obvious notation for sampling.

With the computer, however, these efficiency and notation obstacles can be overcome. Sampling can be placed in its rightfully central place among the statistical concepts in our courses.

In doing so, why not start right off with a sample from a population? One example is the population of US states:

Example 6.1. (Sampling from 50 States)

Students in the US will know that there are 50 states:

```
> nrow(SAT)                # SAT is a data frame in the mosaic package
[1] 50
```

One of the variables to be measured from the population is the student/teacher ratio. There are two equivalent ways to calculate this quantity:

```
> mean(SAT$ratio)
[1] 16.9
> with(SAT, mean(ratio))
[1] 16.9
```

Every student who does this calculation will get the same result. But if each student takes a random sample (without replacement) of, say, 4 states:

```
> sample(SAT, 4)
       state expend ratio salary frac verbal math  sat orig.row
20  Maryland   7.25  17.0   40.7   64    430  479  909       20
15      Iowa   5.48  15.8   31.5    5    516  583 1099       15
23 Minnesota   6.00  17.5   35.9    9    506  579 1085       23
35      Ohio   6.16  16.6   36.8   23    460  515  975       35
```

Then each student gets a potentially different result.

Doing this with a classroom of students will immediately elicit a realization that the random sample is exactly that. Ask the class: Who got California? (There should be one or two in a class of 20.) Who got Alabama? Did anyone get the same state twice? ◇

The value of some measure for the population is, of course, called the **population parameter**. But for a sample, it's a **sample statistic**. It's worthwhile to make the distinction concrete:

```
> with(SAT, mean(ratio))              # population parameter
[1] 16.9
> with(sample(SAT,4), mean(ratio))    # statistic from a sample of 4 states
[1] 16.4
```

Good questions to ask in class at this point: Who got a sample statistic that's bigger than the population parameter? (About half the class will say yes.) Who got a sample statistic that's exactly equal to the population parameter? (It's unlikely that there will be a match.)

At this point, it's helpful to point out that the reason we take samples is because it can be difficult or impossible to measure every member of the population. Of course, it's not so difficult if your population is the 50 US States, but what if your population were the all people living in the US: that's over 300,000,000 people!

6.1.3 Taking Randomness Seriously

It's tempting to think that "random" means that anything goes. This is, after all, the everyday meaning of the word. In statistics, though, the point of randomness is to get a representative sample,

or to assign experimental treatments in some fairly even way. Using randomness properly means taking care and being formal about where the randomness comes from. When you finally submit the research paper you have written after your long years of labor, you don't want it to be rejected because the reviewer isn't convinced that what you called "random" was really so.

First, to demonstrate that the sort of computer-generated random sampling produces reasonable results. Perhaps it's a little too early in the course to talk about sampling distributions, but you certainly can have your class generate random samples of a reasonable size and show that sample statistics are reasonably close to the population parameter. For instance, have each student in your class do this:

```
> with(SAT, mean(ratio))
[1] 16.9
> with(sample(SAT,25), mean(ratio))
[1] 16.7
```

To Do
We'll have more to say about `do()` shortly.

Or, each student can do this several times:

```
> do(5) * with(sample(SAT,25), mean(ratio))
  result
1   16.5
2   16.4
3   17.3
4   16.7
5   16.1
```

This raises the question, naturally enough, of what "reasonably close" means, and what it means to measure "how close." Eventually, that might lead you into consideration of differences, mean differences, mean square differences, and root mean square differences: the standard deviation and variance. But even before encountering this formality the students should be able to see that there is nothing systematically wrong with the answers they get from their random samples. Some are too high compared to the population parameter, some are too low, but as a group they are pretty well centered.

For some applications it is useful to have an alternative vocabulary. If you are fond of examples involving cards, you can use `deal()` and `shuffle()` instead of `sample()`.

```
> # These are equivalent
> sample(cards, 5)
[1] "JD" "3D" "3C" "2H" "4D"
> deal(cards, 5)
[1] "AD" "4S" "QC" "6C" "AH"

> # These are equivalent
> sample(cards)
 [1] "JC"  "3S"  "3C"  "4S"  "2S"  "5S"  "6D"  "2D"  "KS"  "7S"  "AH"  "KD"  "QD"
[14] "4H"  "4D"  "9S"  "KH"  "KC"  "10S" "10C" "3H"  "JS"  "QH"  "2C"  "2H"  "10H"
[27] "10D" "AC"  "7H"  "5C"  "QS"  "5H"  "9C"  "6H"  "AD"  "3D"  "9H"  "7D"  "QC"
[40] "5D"  "8D"  "4C"  "8C"  "7C"  "6C"  "8S"  "9D"  "AS"  "6S"  "JH"  "JD"  "8H"
> shuffle(cards)
 [1] "2H"  "KD"  "3H"  "JH"  "AS"  "QS"  "QS"  "JH"  "AH"  "QS"  "QC"  "JH"  "7C"
[14] "AD"  "2S"  "2C"  "10C" "7C"  "4S"  "8D"  "8H"  "2S"  "AC"  "8D"  "6D"  "QH"
[27] "5H"  "9C"  "4D"  "9D"  "3H"  "6H"  "5D"  "7D"  "JS"  "4H"  "6H"  "2D"  "9D"
[40] "9H"  "7D"  "KD"  "JC"  "AD"  "JS"  "3C"  "8D"  "2D"  "6H"  "5C"  "9H"  "4C"
```

And for sampling with replacement, we offer `resample()`:


```
> # These are equivalent
> sample(1:10, 5, replace=TRUE)
[1]  3  4  5  5 10
> resample(1:10, 5)
[1] 7 4 9 4 7
```

6.1.4 Illuminating Sampling Mistakes

It's helpful at this point to have some examples where an informal or careless approach to randomness leads to misleading results.

Here are some that can be visually compelling:

1. Select books off a library shelf and count how many pages are in each book. When students pick books haphazardly, they get a systematic over-estimate of the page length.

   A random sample of 25 is sufficient to see this for the large majority of students in the class. Going up to 50, though tedious, makes the result even more compelling.

2. A simulation of a real-world study of Alzheimer's disease. The sampling process was to go to nursing homes on a given day and randomly pick out patients who were diagnosed with Alzheimer's. Their files were flagged and, after a few years, the data were examined to find the mean survival time, from when they entered the nursing home to when they died.

3. Sample people and find out how many siblings are in their family (including themselves). Use this data to estimate family size. Since larger families have more children, we will over-sample larger families and over-estimate family size. Section 6.1.7 demonstrates how to fake a data set for this activity. But for the present, we can use the `Galton` dataset and pretend to take a sample of, say, size 300 kids from the whole group of children that Galton had assembled.

   ```
   > data(Galton)
   > with(sample(Galton,300), mean(nkids))
   [1] 6.24
   > with(sample(Galton,300), mean(nkids))
   [1] 6.16
   > do(5) * with(sample(Galton,300), mean(nkids))
     result
   1   6.07
   2   6.15
   3   6.18
   4   6.11
   5   6.03
   ```

   The samples all give about the same answer: about 6 kids in a family. But this is misleading! The case in the `Galton` data is a child, and families with more children have their family size represented proportionally more.

   There are several ways to convert the `Galton` to a new data set where the case is a family. The key information is contained in the `family` variable, which has a unique value for each family. The `duplicated` function identifies levels that are repeats of earlier ones and so the following will do the job (although is not necessarily something that you want to push onto students):

   ```
   > families = subset(Galton, !duplicated(family))
   ```

   Of course, you will want to show that this has done the right job. Compare

   ```
   > nrow(families)
   ```

```
[1] 197
```

to

```
> length(with(Galton, unique(family)))
[1] 197
```

Now check out the average family size when the average is across families, not across kids in families:

```
> with( families, mean(nkids) )
[1] 4.56
```

(Note: In a higher-level course, you could ask students to determine a method to correct for this bias.)

4. Using `googleMap()` (see Section A.1) we can compare uniform sampling of longitude and latitude with correct sampling that takes into account that the earth is a sphere, not a cylinder.

In each of these cases, follow up the haphazard or systematically biased sample with a formally generated sample (which is faster to do, in any event) and show that the formally generated sample gives results that are more representative of the population than the haphazardly sampled data.

6.1.5 Sampling from Distributions

Later in the course, if you cover modeling with distributions, you can sample from these distributions instead of from data sets (treated like populations). R includes set of functions beginning with the letter `r` followed by the (abbreviated) name of a family of distributions (see Table 8.1 in section 8.9 for a list of probability distributions available within base R). These functions all generate a random sample from the distribution given a sample size and any necessary parameters for the distribution.

```
> rnorm(20, mean=500, sd=100)              # fake SAT scores
 [1] 366 366 416 541 507 452 621 443 541 533 612 606 594 420 514 439 353 532 535 367
> rexp(20, rate=1/5)                       # fake lifetime data
 [1]  0.545  3.360  3.450  6.054  1.818  3.990  3.249  4.267 11.645  5.262  7.416
[12]  1.806  5.014  0.847  1.687  2.918  4.358  5.011  0.264  0.354
> rbinom(20, 10, .5)                       # how many heads in 10 coin tosses?
 [1] 6 6 1 6 4 7 5 3 4 6 5 6 4 5 6 6 4 3 5 5
> rgeom(20, .1)                            # how long until Freddy misses his next free throw?
 [1] 29  0 15  7  2  7  0  3  5  2  5  3  1  0  1  1  1 11  4  0
```

The `mosaic` package offers `rdata()` as an alternative to `resample()` with syntax that mirrors these functions for sampling from a distribution.

```
> rdata(20, 1:4)              # samples with replacement
 [1] 4 3 1 4 4 4 2 2 2 1 3 2 1 1 2 2 3 2 2 1
> set.seed(123)
> # these are equivalent;  note the order of the arguments.
> resample(HELP, 2)
> rdata(2, HELP)
> set.seed(123)
> # these are equivalent; sampling without replacement now.
> sample(HELP, 2)
```



```
    age anysubstatus anysub cesd d1 daysanysub dayslink drugrisk e2b female  sex g1b
131  49            1    yes   22  5          1      126        0   4      0 male yes
357  26            1    yes   23  4        106      410        0  NA      0 male  no
    homeless i1  i2  id indtot linkstatus link  mcs  pcs pss_fr racegrp satreat
131 homeless 64 179 144     42          1  yes 45.5 38.1      5   black      no
357   housed  6   6 428     15          0   no 38.3 36.5      5   black      no
    sexrisk substance treat orig.row
131       6   alcohol   yes      131
357       3    heroin    no      357
> rdata(2, HELP, replace=FALSE)
```

6.1.6 do()

The heart of a simulation is doing something over and over. The do() function simplifies the syntax for this and improves the output format (usually returning a data frame). Here's a silly example.

```
> do(3) * "hello"
  result
1  hello
2  hello
3  hello
```

That's a silly example, because we did the same thing each time. If we add randomness, then each replication is (potentially) different:

```
> do(3) * rflip(10)                # 3 times we flip 10 coins
   n heads tails
1 10     6     4
2 10     8     2
3 10     1     9
```

What makes do() clever is that is knows about several types of things we might like to repeat and it tries to keep track of the most important summary statistics for us. If we fit a linear model, for example, do() keeps track of the regression coefficients, $\hat{\sigma}$, and r^2. That makes it useful even if we just want to summarize our data.

```
> do(1) * lm( age ~ sex, HELP)            # using the data as is
  Intercept sexmale sigma r-squared
1      36.3  -0.784  7.71   0.00187
```

But it really shines when we start looking at sampling distributions.

```
> do(3) * lm( age ~ shuffle(sex), HELP)   # simulation under a null hypothesis
  Intercept sexmale sigma r-squared
1      36.8  -1.509  7.69  0.007103
2      35.4   0.325  7.72  0.000307
3      34.8   1.178  7.70  0.004405
```

6.1.7 Generating Fake Data For Sampling Activities

If you don't have actual data at your disposal, you can sometimes simulate data to illustrate your point.

Example 6.2. (Fake Families) We'll simulate 5000 families using a Poisson distribution with a mean of 3 to generate family size. (You can make up whatever elaborate story you like for the population you are considering.)

This example is intended for instructors, not for students.

```
> families <- data.frame(familyid=1:5000, children=rpois(5000,3))
```

Now we generate the people in these families

```
> people <- data.frame(
                familyid = with(families, rep( familyid, children)),
                sibs = with(families, rep( children, children))
                )
```

Computing the mean "family size" two different ways reveals the bias of measuring family size by sampling from children rather than from families.

```
> with(families, mean(children))
[1] 2.99
> with(people, mean(sibs))
[1] 3.98
```

If the result seems mysterious, the following tables might shed some light.

```
> with(families, table(children))
children
   0    1    2    3    4    5    6    7    8    9   10   12
 253  725 1141 1120  848  510  247   95   45   11    4    1
> with(people, table(sibs))
sibs
   1    2    3    4    5    6    7    8    9   10   12
 725 2282 3360 3392 2550 1482  665  360   99   40   12
```

◇

Once built, you can provide these data frames to your students for a sampling exercise. (See Section 7.1.)

Now that we have discussed various methods for generating random data, it's time to put those skills to good use computing p-values.

6.1.8 Lady Tasting Tea

If you don't use the Lady Tasting Tea as a course starter (Section 1.2.1), you can use it as an introduction to testing a proportion.

6.1.9 Golfballs in the Yard

This example can be used as a first example of hypothesis testing or as an introduction to chi-squared tests. As an introduction to hypothesis testing it is very useful in helping students understand what a test statistic is and its role in hypothesis testing.

The Story

Allan Rossman once lived along a golf course. One summer he collected the golf balls that landed in his yard and tallied how many were labeled with 1's, 2's, 3's, and 4's because he was curious to know whether these numbers were equally likely.[1]

[1] You can have some discussion with your students about what population is of interest here. Given the location of the house, the golf balls were primarily struck by golfers of modest ability and significant slice, all playing on one particular golf course. These results may or may not extend to other larger populations of golfers.

Of the first 500 golf balls, 14 had either no number or a number other than 1, 2, 3, or 4. The remaining 486 golf balls form our sample:

1	2	3	4	other
137	138	107	104	14

We can enter this data into R using the `c()` function.

```
> golfballs <- c(137, 138, 107, 104)
```

Coming up with a test statistic

At this point, ask students what they think the data indicates about the hypothesis that the four numbers are equally likely. Students usually notice right away that there are a lot of 2's. But perhaps that's just the result of random sampling. We can generate random samples and see how the random samples compare with the actual data:

```
> table(rdata(486, 1:4))   # 486 draws from the numbers 1 thru 4 (equally likely)

  1   2   3   4
128 125 111 122
```

`rgolfballs` can be generated in advance if you don't want to distract your students with thinking about how do create it. There is a pre-built `rgolfballs` in the `fastR` package.

It is useful to generate some more of these samples to get a better feel for the sampling distribution under the null:

```
> do(25) * table(rdata(486, 1:4))
```

```
    1   2   3   4
1  122 126 112 126
2  118 128 131 109
3  128 112 143 103
4  119 133 120 114
5  115 128 118 125
6  107 136 118 125
7  128 111 117 130
8  122 117 117 130
9  111 132 116 127
10 127 119 113 127
11 124 127 102 133
12 137 128 112 109
13 106 136 107 137
14 126 129 109 122
15 139 127 103 117
16 111 136 115 124
17 140 108 124 114
18 129 114 119 124
19 121 108 120 137
20 128 114 121 123
21 127 114 121 124
22 129 119 131 107
23 125 127 133 101
24 120 123 126 117
25 145 127 105 109
```

Tip
Have your students calculate test statistics mentally from a small portion of the sampling distribution. Assign each student a row or two, then ask for a show of hands to see how many exceed the test statistic calculated from the data.

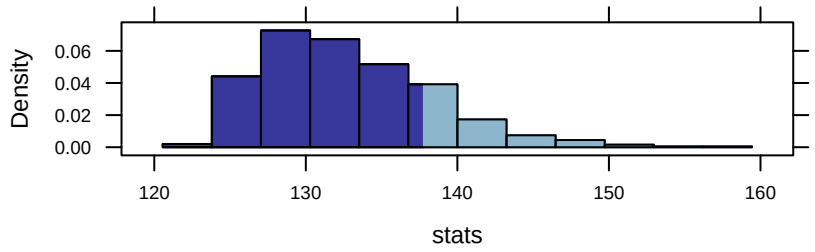
From this we see that it is not incredibly unlikely to see a count of 138 or more. (See samples 3, 15, 17, 25.) Students are often surprised just how often this occurs.

Once students understand the idea of a test statistic and how it is computed from data, it's time to let the computer automate things. First, we generate a better approximation to the sampling distribution assuming each number is equally likely.

```
> rgolfballs <- do(2000) * table(rdata(486, 1:4))
```

The `statTally()` function can tabulate and display the sampling distribution and compared to the test statistic.

```
> print(statTally(golfballs, rgolfballs, max))
Test Stat function: max


Test Stat applied to sample data = 138

Test Stat applied to random data:

50% 90% 95% 99%
132 141 143 149

Of the random samples
         1622 ( 81.1 % ) had test stats < 138
         62 ( 3.1 % ) had test stats = 138
         316 ( 15.8 % ) had test stats > 138
> ladd(print(panel.abline(v=138)))                # add a vertical line
NULL
```



More test statistics

One of the goals of this activity is to have the students understand the role of a test statistic. Students are encouraged to dream up test statistics of their own. The minimum count is often suggested as an alternative, so we try that one next.

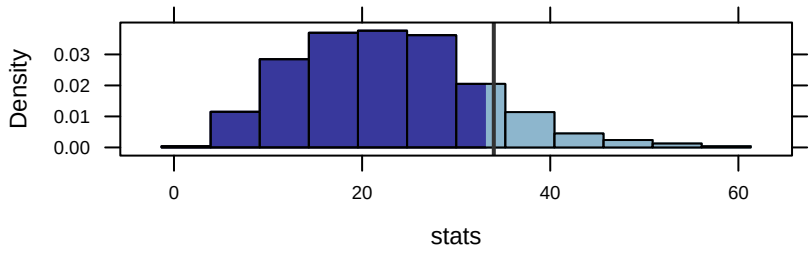
```
> print(statTally(golfballs, rgolfballs, min))   # output surpressed.
```

See Section B.7 for a tutorial on writing your own functions.

These two test statistics (maximum count and minimum count) feel like they aren't making full use of our data. Perhaps we would do better if we looked at the difference between the maximum and minimum counts. This requires writing a simple function.

```
> mystat1 <- function(x) { diff(range(x)) }
> print(statTally(golfballs, rgolfballs, mystat1, v=mystat1(golfballs))) # add a vertical line
Test Stat function: mystat1


Test Stat applied to sample data = 34

Test Stat applied to random data:

50% 90% 95% 99%
 22  36  40  49
```

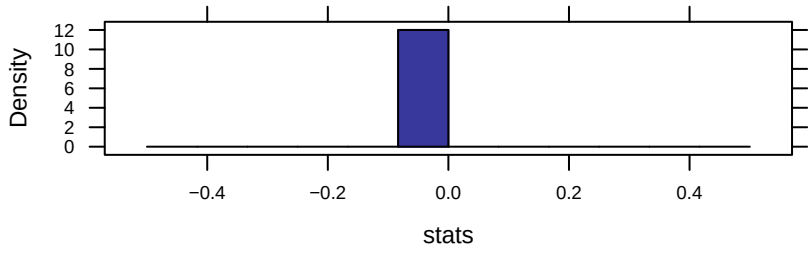
```
Of the random samples
         1712 ( 85.6 % ) had test stats < 34
         42 ( 2.1 % ) had test stats = 34
         246 ( 12.3 % ) had test stats > 34
```



The World's Worst Test Statistic

Usually I get lucky and someone will suggest the world's worst test statistic: The sum of the differences between the counts and $486/4 = 121.5$.

```
> mystat2 <- function(x) { sum( x - 121.5 ) }
> print(statTally(golfballs, rgolfballs, mystat2))
Test Stat function: mystat2


Test Stat applied to sample data = 0

Test Stat applied to random data:

50% 90% 95% 99%
  0   0   0   0

Of the random samples
         0 ( 0 % ) had test stats < 0
         2000 ( 100 % ) had test stats = 0
         0 ( 0 % ) had test stats > 0
```

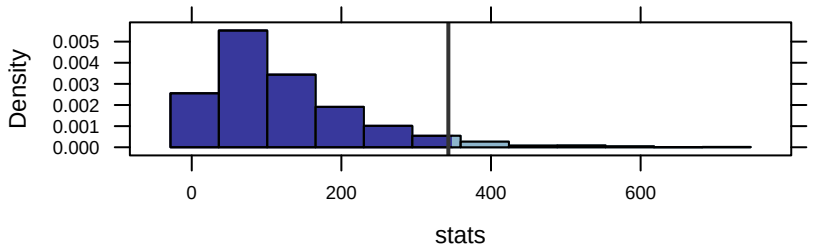


This test statistic is bad because it doesn't depend on the data, so the distribution of the test statistic is the same whether the null hypothesis is true or false.

But it is close to a good idea. Let's add in an absolute value...

Tip
As students come up with test statistics, let them name them, or name them after them (S for the Smith statistic, etc.) It adds to the fun of the activity and mirrors how the statistics they will learn about got their names.

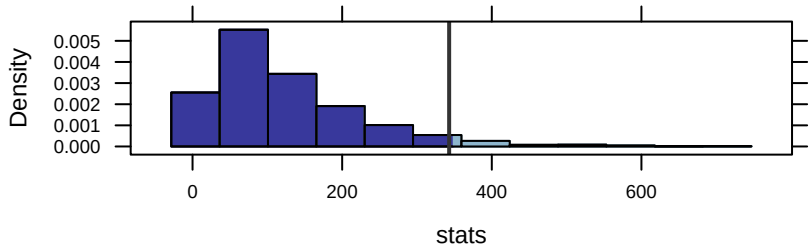
```
> sad <- function(x) { sum(abs(x-121.5)) }
> print(statTally(golfballs, rgolfballs, sad, v=sad(golfballs)))

Test Stat function: sad


Test Stat applied to sample data = 64

Test Stat applied to random data:

50% 90% 95% 99%
 30  48  54  67

Of the random samples
         1970 ( 98.5 % ) had test stats < 64
         5 ( 0.25 % ) had test stats = 64
         25 ( 1.25 % ) had test stats > 64
```



It is a matter of teaching style whether you write this function with the magic number 121.5 hard-coded in or use `mean(x)` instead. The latter is preferable for generalizable method, of course. But the former may have pedagogical advantages, especially in the Intro Stats course.

Squaring those differences (or equivalently using the standard deviation or variance) is also often suggested.

```
> print(statTally(golfballs, rgolfballs, var, v=var(golfballs)))

Test Stat function: var


Test Stat applied to sample data = 343

Test Stat applied to random data:

  50%   90%   95%   99%
 97.7 251.9 313.7 481.1

Of the random samples
         1925 ( 96.25 % ) had test stats < 343
         1 ( 0.05 % ) had test stats = 343
         74 ( 3.7 % ) had test stats > 343
```



This example illustrates some important ideas about hypothesis testing, namely.

1. The test statistic must summarize the evidence we will use to judge the null hypothesis *in a single number* computed from our sample.
2. We judge the test statistic computed from our sample by comparing it to test statistics computed on random samples *assuming the Null Hypothesis is true.*
3. Some test statistics work better than others.

   A good test statistic should look quite different when the null hypothesis is true from how it looks when the null hypothesis is false. (This is a first hint at the idea of power.)

6.2 The Multi-World Metaphor for Statistical Inference

In this section we move toward a more systematic approach to empirical methods with goal of providing a flexible tool that works in a wide range of situations.

Statistical inference is hard to teach.[2] Often, instead of teaching the logic of inference, we teach methods and techniques for calculating the quantities used in inference: standard errors, t-statistics, p-values, etc.

Perhaps because students don't understand the logic, they have strong misconceptions about confidence intervals and, especially, about p-values. For example, even among professional scientists, the mainstream (mis)-understanding of p-values is that they reflect the probability that the null hypothesis is correct.

Part of the reason why statistical inference is hard to grasp is that the logic is genuinely hard. It involves contrapositives, it involves conditional probabilities, it involves "hypotheses." And what is a hypothesis? To a scientist, it is a kind of theory, an idea of how things work, an idea to be proven through lab or field research or the collection of data. But statistics hews not to the scientist's but to the philosopher's or logician's rather abstract notion of a hypothesis: a "proposition made as a basis for reasoning, without any assumption of its truth." (Oxford American Dictionaries) Or more simply:

A hypothesis is a statement that may be true or false.

The simpler "definition" is a useful way to describe this to students, but it is not equivalent to the dictionary definition which includes an important notion of the reason for hypotheses. A hypothesis is a statement that is posed for the purposes of determining what would follow if the statement were true. This is the sense of 'hypothesis of a theorem'. It is also the sense of the null hypothesis. We assume the null hypothesis is true and 'see what follows' from that assumption. Unfortunately, it is not the way this word is often used in high school science courses.

[2] Here we are considering only the frequentist version of inference. The Bayesian approach is different and has different features that make it hard to teach and understand. In these notes, we will be agnostic about frequentist vs Bayesian, except to acknowledge that, for good or bad, the frequentist approach is vastly dominant in introductory statistics courses.

Thought of another way, the scientists have conflated the words 'hypothesis' and 'conjecture'.

What kind of scientist would frame a hypothesis without some disposition to think it might be true? Only a philosopher or a logician.

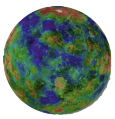
To help connect the philosophy and logic of statistical hypotheses with the sensibilities of scientists, it might be helpful to draw from the theory of kinesthetic learning — an active theory, not a philosophical proposition — that learning is enhanced by carrying out a physical activity. Many practitioners of the reform style of teaching statistics engage in the kinesthetic style; they have students draw M&Ms from a bag and count the brown ones, they have students walk randomly to see how far they get after n steps [Kap09], or they toss a globe around the classroom to see how often a students' index finger lands in the ocean [GN02].

To teach hypothesis testing in a kinesthetic way, you need a physical representation of the various hypotheses found in statistical inference. One way to do this involves not the actual physical activity of truly kinesthetic learning, but concrete stories of making different sorts of trips to different sorts of worlds. A hypothesis may be true on some worlds and false on others. On some planets we will know whether a hypothesis is true or false, on others, the truth of a hypothesis is an open question.

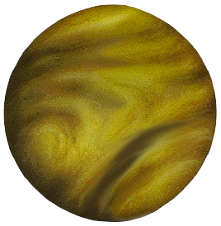
Of course, the planet that we care about, the place about which we want to be able to draw conclusions. It's Planet Earth:

We want to know which hypotheses are true on Earth and which are false. But Earth is a big place and complicated, and it's hard to know exactly what's going on. So, to deal with the limits of our abilities to collect data and to understand complexity, statistical inference involves a different set of planets. These planets resemble Planet Earth in some ways and not in others. But they are simple enough that we know exactly what's happening on them, so we know which hypotheses are true and which are false on these planets.

These planets are:



Planet Sample



Planet Null

Planet Alt

Planet Sample is populated entirely with the cases we have collected on Planet Earth. As such, it somewhat resembles Earth, but many of the details are missing, perhaps even whole countries or continents or seas. And of course, it is much smaller than Planet Earth.

Planet Null is a boring planet. Nothing is happening there. Express it how you will: All groups all have the same mean values. The model coefficients are zero. Different variables are unrelated to one another. Dullsville. But even if it's dull, it's not a polished billiard ball that looks the same from every perspective. It's the clay from which Adam was created, the ashes and dust to which man returns. But it varies from place to place, and that variation is not informative. It's just random.

Finally, there is Planet Alt. This is a cartoon planet, a caricature, a simplified representation of our idea of what is (or might be) going on, our theory of how the world might be working. It's not going to be exactly the same as Planet Earth. For one thing, our theory might be wrong. But also, no theory is going to capture all the detail and complexity of the real world.

In teaching statistical inference in a pseudo-kinesthetic way, we use the computer to let students construct each of the planets. Then, working on that planet, the student can carry out simulations of familiar statistical operations. Those simulations tell the student what they are likely to see *if they were on that planet*. Our goal is to figure out whether Earth looks more like Planet Null or more like Planet Alt.

For the professional statisticians, the planet metaphor is unnecessary. The professional has learned to keep straight the various roles of the null and alternative hypothesis, and why one uses a sample standard error to estimate the standard deviation of the sampling distribution from the population. But most students find the array of concepts confusing and make basic categorical errors. The concrete nature of the planets simplifies the task of keeping the concepts in order. For instance, Type I errors only occur on Planet Null. You have to be on Planet Alt to make a Type II error. And, of course, no matter how many (re)samples we make on Planet Sample, it's never going to look more like Earth than the sample itself.

6.2.1 The Sampling Distribution

Section 6.1.3 shows some examples of drawing random samples repeatedly from a set of data. To discuss the sampling distribution, it's helpful to make clear that the sampling distribution refers to results from the **population**. In the planet metaphor, this means that the samples are drawn from Planet Earth:

It can be a challenge to find a compelling example where you have the population, rather than a sample, in hand. Sports statistics provide one plausible setting, where you have, for example, the names and attributes of every professional player. The ability to sample from Earth mapping software provides another compelling setting. (See Section A.1.)

For our example, we will take the complete list of results from a running race held in Washington, D.C. in April 2005 — the Cherry Blossom Ten-Mile Run. The data set gives the `sex`, `age`, and `net` running time (start line to finish line) for each of the 8636 runners who participated: the population of runners in that race.

```
> population <- TenMileRace
> nrow(population)
```

```
[1] 8636
```

If you have the population on the computer, there's little point in taking a random sample. But the point here is to illustrate the consequences of taking a sample. So, imagine that in fact it was difficult or expensive or dangerous or destructive to collect the data on a runner, so you want to examine just a small subset of the population.

Let's collect a sample of 100 runners:

```
> planet.sample <- sample(population, 100)
```

With that sample, we can calculate whatever statistics are of interest to us, for instance:

```
> with(planet.sample, mean(age))
[1] 36.8
> with(planet.sample, sd(age))
[1] 10.3
> with(planet.sample, mean(sex=="F"))
[1] 0.47
> lm(net ~ age + sex, data=planet.sample)
...
(Intercept)          age         sexM
       5554           14         -957
```

These various numbers are informative in their own way about the sample, but it's important to know how they might relate to the population. For example, how precise are these numbers? That is, if someone had collected a different random sample, how different would their results likely be?

Given that we have the population in hand, that we're on Planet Earth, we can go back and collect such samples many more times and study how much they vary one to the other. We'll do this by replacing `planet.sample` with a command to sample anew from the population.

```
> with(sample(population,100), mean(age))
[1] 36.9
> with(sample(population,100), sd(age))
[1] 10.9
> with(sample(population,100), mean(sex=="F"))
[1] 0.56
> lm(net ~ age + sex, data=sample(population,100))
...
(Intercept)          age         sexM
     5403.0         15.4       -854.0
```

Slightly different results! To quantify this, repeat the sampling process many times and look at the distribution: the **sampling distribution**.

```
> sample.means <- do(500) * with(sample(population,100), mean(age))
> sample.sds   <- do(500) * with(sample(population,100), sd(age))
> sample.props <- do(500) * with(sample(population,100), mean(sex=="F"))
> sample.regressions  <- do(500) * lm(net ~ age + sex, data=sample(population,100))
```

You can display the sampling distribution in several ways: histograms, box-and-whisker plots, density plots. This is worth doing with the class. Make sure to point out that the distribution depends on the statistic being calculated: in the above examples, the mean age, the standard deviation of ages, the fraction of runners who are female, the relationship between running time and `sex` and `age`. (We'll get to the dependence on sample size in a little bit.)

Insofar as we want to be able to characterize the repeatability of the results of the sampling process, it's worth describing it in terms of the spread of the distribution: for instance, the standard deviation. Of course, when talking about a sampling distribution, we give another name to the standard deviation: the *standard error*. Actually calculating the standard error may perhaps solidify in the students mind that it is a standard deviation.

```
> sd(sample.means)  # standard error for mean age
result
  1.02
```



```
> sd(sample.sds)    # standard error for sd of age
result
 0.781
> sd(sample.props)  # standard error for fraction female
result
0.0489
> sd(sample.regressions)  # standard errors for regression statistics
Intercept       age      sexM     sigma r-squared
 344.9749    9.5826  191.3007   78.4503    0.0657
```

Example 6.3. This is an activity to carry out in class, with each student or pair of students at a computer.

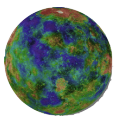
1. Make sure you can replicate the calculations for the standard error of one of the statistics in the `TenMileRace` example with a sample of size $n = 100$. Your reproduction won't be exact, but it should be reasonably close.
2. Now repeat the calculations, but use sample sizes that are larger. From $n = 100$, increase to $n = 400$, then $n = 1600$, then $n = 6400$. How does the standard error depend on n? Does larger n lead to a bigger or smaller standard error? Which of these formulas most closely matches the pattern you see:
   (a) The standard error increases with n.
   (b) The standard error increases with $\sqrt{n}$.
   (c) The standard error gets smaller with increasing n with the pattern $1/\sqrt{n}$.
   (d) The standard error gets smaller with increasing n with the pattern $1/n$.
3. Use the pattern you observed to predict what will be the standard error for a sample of size $n = 1000$. Then carry out the actual simulation of repeated sampling using that sample size and compare your prediction to the result you actually got.
4. In the above, you used `do(500)` replications of random sampling. Suppose you use `do(100)` or `do(2000)` replications instead? Do your results depend systematically on the number of replications?

◇

6.2.2 The Re-Sampling Distribution

The sampling distribution is a lovely theoretical thing. But if it were easy to replicate taking a sample over and over again, wouldn't you just take a larger sample in the first place? The practical problem you face is that the sample you took was collected with difficulty and expense. There's no way you are going to go back and repeat the process that was so difficult in the first place. Remember, real samples from the real population (on planet Earth) can't be obtained by simply asking a computer to sample for us.

This is where Planet Sample comes in.



In our example we have access to the entire population, but this is not typically the case.

Although Planet Earth is very large and we have only incomplete information about it, all the data for Planet Sample is in our computer. So

Although sampling from Planet Earth is difficult, sampling from Planet Sample is easy.

To illustrate, here's a sample from the running population:

```
> planet.sample = sample(population, 100) # one sample from the population
```

In reality, of course, you wouldn't run a command to create the sample. You would go out and do the hard work of randomly selecting cases from your population, measuring their relevant attributes, and recording that data. So let's pretend that's what we did. Pretend that `planet.sample` is the result of laborious data collection.

Now you can go through the exact calculations you did to construct the sampling distribution (on Planet Earth), but instead sample from Planet Sample:

```
> with(sample(planet.sample,100), mean(age))
[1] 36.6
> with(sample(planet.sample,100), mean(age))
[1] 36.6
> with(sample(planet.sample,100), mean(age))
[1] 36.6
```

Wait, something went wrong. We are getting the same mean every time. The reason is that Planet Sample is small. It only has 100 inhabitants and we are sampling all 100 each time. We could take smaller samples, but we want to learn about samples of the same size as our actual sample, so that's not a good option.

CAUTION! Remember to use `resample()` rather than `sample()` to compute a resampling distribution.

Our solution to this problem is to sample *with replacement*. That is, we will select an inhabitant of Planet Sample, record the appropriate data, and then put them back on the planet – possibly selecting that same inhabitant again later in our sample. We'll call sampling with replacement from Planet Sample **resampling** for short. The `resample()` function in the `mosaic` package can do this as easily as sampling without replacement.

```
> with(resample(planet.sample,100), mean(age))
[1] 37.9
> with(resample(planet.sample,100), mean(age))
[1] 37.9
> with(resample(planet.sample,100), mean(age))
[1] 36.6
```

Ah, that looks better. Now let's resample a bunch of times.[3]

```
> resample.means <- do(500) * with(resample(planet.sample,100), mean(age))
> resample.sds   <- do(500) * with(resample(planet.sample,100), sd(age))
> resample.props <- do(500) * with(resample(planet.sample,100), mean(sex=="F"))
> resample.regressions <- do(500) * lm(net ~ age + sex, data=sample(planet.sample,100))
```

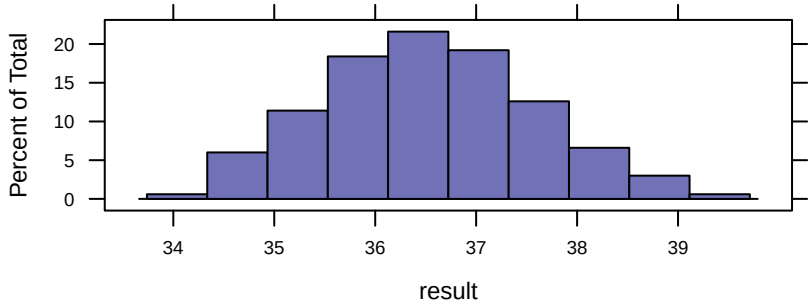
And, then, summarize the resulting distributions:

```
> sd(resample.means)        # standard deviation of mean ages
result
  1.05
> sd(resample.sds)          # standard deviation of sd of age
result
 0.758
```

[3] By default, `resample()` will draw a sample as large as the population of Planet Sample, so we can drop the sample size if we like.


```
> sd(resample.props)        # standard deviation of fraction female
result
0.0492
> sd(resample.regressions)  # standard deviation of resampled regression statistics
Intercept       age      sexM     sigma r-squared
 1.50e-12  1.78e-14  3.47e-13  1.43e-13  2.54e-17
```

```
> histogram( ~ result, resample.means )
```



Sampling like this on Planet Sample, isn't quite the same as constructing the sampling distribution. This is for the simple reason that it is being done on Planet Sample rather than Planet Earth. To emphasize the distinction, it's helpful to refer to the resulting distribution as the *resampling distribution* in contrast to the sampling distribution.

Example 6.4. Have your students compare the results they get from resampling of a fixed sample of size n, to repeated draws from the population with the same sample size n.

Emphasize that the resampling process is good for estimating the width of the sampling distribution, but not so good for estimating the center. That is, the results on Planet Sample will generally compare quite well to Planet Earth for the standard error, but the means of the sampling distributions and the resampling distributions can be quite different.

In a more advanced class, you might ask how big a sample is needed to get a reasonable estimate of the standard error. ◇

The question naturally arises, is the standard error estimated from the resampling distribution good enough to use in place of the standard deviation of the actual sampling distribution. Answering this question requires some solid sense of *what you are using the standard error for*. In general, we use standard errors to get an idea of whether the point estimate is precise enough for the purpose at hand. Such questions can be productively addressed on Planet Alt, where we will journey after a short detour to that most boring of all places, Planet Null.

6.2.3 The Sampling Distribution Under the Null Hypothesis

The Null Hypothesis is often introduced in terms of the values of population parameters, e.g., "The population mean is 98.6," or "The difference between the two group means is zero," or "The population proportion is 50%."

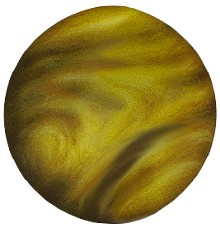
Perhaps this is fine if all you want to talk about is means or proportions or differences between means, as is so often the focus of an introductory statistics course. But instructors would like to think that

their students are going to go farther, and that the introductory course is meant to set them up for doing so.

In the multi-planet metaphor, the Null Hypothesis is about a place where variables are unrelated to one another. Any measured relationship, as indicated by a difference in sample means, a sample correlation coefficient different from zero, non-zero model coefficients, etc., is, on Planet Null, just the result of random sampling fluctuations.

Planet Null is, however, easy to describe in the case of hypotheses about a single proportion. In fact, that is what our Lady Tasting Tea example did (Section 1.2.1).

This formulation is very general and is not hard for students to understand. Ironically, it doesn't work so well for the very simplest null hypotheses that are about single-group means or proportions, so it turns out to be easier to introduce the null hypothesis with the supposedly more complicated cases, e.g., differences in means or proportions, regression coefficients, etc.



Like Planet Sample, Planet Null is a place that you construct. You construct it in a way that makes the Null Hypothesis true, destroying relationships between variables. You've already seen the process in Section 6.3.1: randomization with `shuffle`. The basic idea is to treat relationships as a mapping from the values of explanatory variables in each case to a response variable. By randomizing the explanatory variables relative to the response, you generate the world in which the Null Hypothesis holds true: Planet Null.

Examples

Let's look as some examples.

Example 6.5. The `Whickham` data set contains data from a 1970's sample of 1314 residents of Whickham, a mixed urban and rural district near Newcastle upon Tyne, in the UK. Did more than half of the adult residents of Whickham smoke at that time?

We could use the same approach that worked for the Lady Tasting Tea, but we'll do things a little differently this time. First, let's take a quick look at some of the data.

```
> n <- nrow(Whickham); n
```

```
[1] 1314
```

```
> head(Whickham,3)
```

```
  outcome smoker age
1   Alive    Yes  23
2   Alive    Yes  18
3    Dead    Yes  71
```

We'll answer our question by comparing the smoking rate on Planet Sample

```
> with(Whickham, proptable(smoker))                    # less than 50% of sample smoke
```

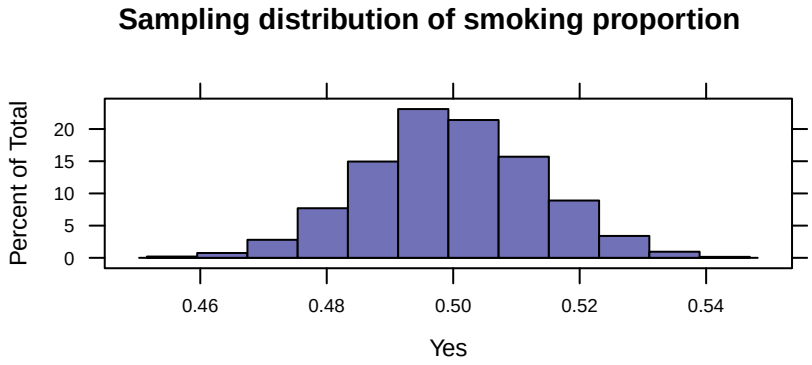
```
smoker
   No   Yes
0.557 0.443
```

to the sampling distribution on Planet Null.


```
> planet.null <- data.frame( smoker=c('Yes','No') )  # equal mix of Yes and No on Planet Null
> null.dist <- do(2000) * with( resample(planet.null, n), proptable(smoker) )
> head(null.dist,3)
     No   Yes
1 0.485 0.515
2 0.488 0.512
3 0.508 0.492
> table(with(null.dist, Yes < .4429))
FALSE
 2000
```

2000 of our 2000 samples from Planet Null had a proportion of smokers smaller than the proportion on Planet Sample. It does not appear that our sample came from Planet Null.

```
> histogram( ~ Yes, null.dist, main='Sampling distribution of smoking proportion')
```



◇

Example 6.6. Do the foot widths differ between boys and girls, judging from the `KidsFeet` data?

```
> data(KidsFeet)
> lm( width ~ sex, data=KidsFeet ) # our sample
...
(Intercept)         sexG
      9.190       -0.406
```

Conceptually, this example can be done without the use of `lm()`, but the syntax is trickier. `lm()` takes care of computing the means of the two groups and the difference in those means. See Section 6.3.1.

Looks like girls' feet are a little bit narrower. But is this the sort of thing we might equally well see on Planet Null, where there is no systematic difference between boys' and girls' feet?

```
> do(1) * lm( width ~ shuffle(sex), data=KidsFeet ) # planet null
  Intercept  sexG sigma r-squared
1      8.93 0.149 0.511    0.0216
```

For this particular sample from Planet Null, the girls' feet are a little wider than the boys'. By generating a sampling distribution on Planet Null, we can see the size of the relationship to be expected just due to sampling fluctuations in a world where there is no relationship.

```
> # (approximate) distribution on planet null
> planet.null <- do(500) * lm( width ~ shuffle(sex), data=KidsFeet )
> head(planet.null,2)
  Intercept   sexG sigma r-squared
1      8.96 0.0594 0.516   0.00342
2      8.96 0.0658 0.515   0.00420
```

```
> with( planet.null, mean(abs(sexG) > abs(-0.4058)) )  # a p-value
[1] 0.006
```

The value of -0.4058 observed in our sample is not very likely on Planet Null. This suggests that our sample was not collected from Planet Null: we can reject the Null Hypothesis.

◇

Example 6.7. Is the survival rate for smokers different from that for non-smokers?

```
> do(1) * lm( outcome=="Alive" ~ smoker, data=Whickham ) # from our sample
  Intercept smokerYes sigma r-squared
1     0.686    0.0754 0.448   0.00694
> do(1) * lm( outcome=="Alive" ~ shuffle(smoker), data=Whickham ) # planet null
  Intercept smokerYes sigma r-squared
1     0.738   -0.0446 0.449   0.00239
> # distribution on planet null
> null.distribution  = do(500) * lm( outcome=="Alive" ~ shuffle(smoker), data=Whickham )
> with(null.distribution, mean( abs(smokerYes)  > abs(0.07538) ) ) # a p-value
[1] 0.014
```

If you're shocked to see that smoking is associated with greater likelihood of being alive (7.5 percentage points greater!) and that the data indicate that this is statistically significant, you should be. But the problem isn't with the calculation, which is the same one you will get from the textbook formulas. The problem is with the failure to take into account covariates. What's shocking is that we teach students about p-values without teaching them about covariates and how to adjust for them.

The big covariate here is `age`. It happens that in the Whickham data, younger people are more likely to smoke. To see this you :

```
> do(1) * lm( smoker=="Yes" ~ age, data=Whickham )          # our sample
  Intercept      age sigma r-squared
1     0.596 -0.00326 0.494    0.0131
> do(1) * lm( smoker=="Yes" ~ shuffle(age), data=Whickham ) # under the null
  Intercept      age sigma r-squared
1     0.402 0.000877 0.497  0.000953
> # distribution on planet null
> null.distribution <- do(500) * lm(smoker=="Yes" ~ shuffle(age), data=Whickham )
> with(null.distribution, mean(abs(age) > abs(-0.00326)) ) # approx. p-value
[1] 0
```

So, greater `age` is associated with lower `smoker` status. And, of course, older people are more likely to die. Taking both factors together, it turns out that smokers are less likely to die, but that's because they are young. ◇

Example 6.8. Let's make up for the deficiency in the above smoking example. One way to do this is to adjust for `age` when considering the effect of `smoker` status. We'll consider this more in Chapter **??**, but for now, we'll just build the model.

```
> # our sample on planet Earth
> do(1) * glm( outcome=="Alive" ~ smoker + age, data=Whickham, family="binomial" )
  Intercept smokerYes    age
1       7.6    -0.205 -0.124
> # o sample on planet null
> do(1) * glm( outcome=="Alive" ~ shuffle(smoker) + age, data=Whickham, family="binomial" )
```



```
  Intercept smokerYes    age
1      7.39    0.0332 -0.122
```

```
> # distribution on planet null
> null.distribution <- do(500) * glm( outcome=="Alive" ~ shuffle(smoker) + age,
                                      data=Whickham, family="binomial" )
> with(null.distribution, mean(abs(smokerYes) > abs(-0.205)) ) # approx. p-value
```

```
[1] 0.196
```

You can see that the coefficient on `smokerYes` is negative, and that the p-value indicates significance. So, smoking in these data are associated with a lower probability of survival, but only when adjusting for `age`.

You might have noticed that the model built here was a logistic model. There's good reason to do that, since we are modeling probabilities the value of which must always be between 0 and 1. But notice also that the logic of hypothesis testing remains the same: construct a Planet Null by randomizing an explanatory variable with respect to a response variable. ◇

As this is being written, the US Space Shuttle is carrying out it's last couple of missions before being retired. For a few years, at least, your students will know what you mean by "Space Shuttle." You can help them remember the way to create Planet Null if, at the cost of some self-dignity, you tell them, "Take the Space Shuffle to Planet Null."

6.2.4 The Sampling Distribution Under the Alternative Hypothesis

Planet Alt is the place where we implement our theories of how the world works. This will seem like an odd statement to those who are familiar with the standard introductory textbook formulation of the alternative hypothesis in its "anything but the Null" form, e.g., $H_a : \mu_1 \neq \mu_2$. Move away from that formulation, whose only point seems to be to inform whether to do a one-tailed or a two-tailed test.

Instead head off to Planet Alt.

How do you get there? You build a simulation of the world as you think it might be.

Example 6.9. To illustrate, imagine that you are interested in researching the potential relationship between vitamin D deficiency and high blood pressure. Your eventual plan is to do a study, perhaps one where you draw blood samples to measure vitamin D levels, and correlate this with high blood pressure. From your previous experience, you know that high blood pressure is particularly a problem in men aged 50 and older, and that black men seem to be more susceptible than whites.

You scour the literature, looking for data on vitaminD and blood pressure. What you find are some papers describing vitamin D levels in blacks and whites, and that, on average, blacks seem to have substantially lower vitamin D levels than whites. There's also data on high blood pressure, but no good study relating vitamin D to blood pressure. That's a problem, but it's also the source of your research opportunity.

You form your alternative hypothesis based on your idea of a world in which the relationship between vitamin D and blood pressure is large enough to be interesting at a clinical level, but small enough to

The instructor would provide this function to the students.

have been missed by past research. You decide a substantial but typical deficiency in vitamin D will, on average lead to a 5mmHg increase in systolic blood pressure.

Now, to construct Planet Alt:

```
> planet.alt = function(n=10,effect.size=1) {
   race = resample( c("B","B","W","W","W"), n)
   D = pmax(0, rnorm(n,mean=(37.7+(71.8-37.7)*(race=="W")),
                       sd=(10+(20-10)*(race=="W"))))
   systolic = pmax(80, rnorm(n,mean=130,sd=8) +
                       rexp(n,rate=.15) +
                       effect.size*(30-D)/2 )
   return( data.frame(race=race,
                      D=D,
                      systolic=round(systolic)) )
 }
```

The internals of such a function is not for the faint of heart. You'll see that it creates people that are randomly of race B or W, and in a proportion that you might choose if you were sampling with the intent of examining seriously the role that race plays. The vitamin D level is set, according to your findings in the literature, to have a mean of 37.7 for blacks and 71.8 for whites. The standard deviations also differ between blacks and whites. Finally, the systolic blood is set to produce a population that is a mixture of a normal distribution around 130 mmHg with an exponential tail toward high blood pressure. A drop of vitaminD levels of 10 units leads to an increase in blood pressure of 5 mmHg. There's a parameter, `effect.size` that will allow you to change this without re-programming.

You might not agree with this specific alternative hypothesis. That's fine. You can make your own Planet Alt. There are plenty to go around!

In contrast to the complexity of writing the simulation, using it is simple. Here's a quick study of size $n = 5$:

```
> planet.alt(5)
  race    D systolic
1    W 54.3      118
2    B 45.1      126
3    B 23.7      136
4    B 48.3      139
5    B 34.9      137
```

◇

Example 6.10. In this example we construct Planet Alt for a comparison of two means.

```
> alt.2.groups <- function(n=10, effect.size=1, sd=1, baseline=0){
         n <- rep(n, length.out=2)
         sd <- rep(sd, length.out=2)
         data.frame(
                 y = c( rnorm(n[1], baseline, sd[1]), rnorm(n[2], baseline+effect.size, sd[2])),
                 group = rep(c('A','B'), n)
         )
 }
```

```
> alt.2.groups(3)
       y group
1 -0.579     A
2 -0.530     A
3  0.628     A
4 -0.316     B
5 -0.504     B
6  0.162     B
```


◇

We'll discuss how to use such simulations to compute power in Section 6.5.

6.3 More Examples

6.3.1 Comparing Two Means

In this section we show several ways to see if the mean ages of men and women in the HELP study are significantly different.

Using `lm()`

We begin with a simple method that uses a big tool – linear models. If you do linear models early, this is the way to go.

In this example, R will automatically convert the `sex` factor into 0's and 1's, so the slope parameter in the model is the difference in the means.

```
> mean(age ~ sex, data=HELP)
```

```
     sex    S   N Missing
1 female 36.3 107       0
2   male 35.5 346       0
3    ALL 35.7 453       0
```

```
> lm(age ~ sex, data=HELP)                        # actual data
```

```
...
(Intercept)      sexmale
     36.252       -0.784
```

The `do()` function conveniently stores the values of the estimated parameters that result from fitting with `lm()`, so it is relatively easy to obtain the sampling distribution for any of these estimated parameters.

```
> do(1) * lm(age ~ sex, data=HELP)                # actual data
```

```
  Intercept sexmale sigma r-squared
1      36.3  -0.784  7.71   0.00187
```

```
> do(2) * lm(age ~ shuffle(sex), data=HELP)       # shuffled data
```

```
  Intercept sexmale sigma r-squared
1      35.6  0.1067  7.72  3.42e-05
2      35.6  0.0312  7.72  2.77e-06
```

```
> set.seed(123)                                          # for later comparison purposes
> null.dist <- do(1000) * lm(age ~ shuffle(sex), HELP)
> test.stat <- (do(1) * lm(age ~ sex, HELP) )$sexmale
> rtest.stats <- null.dist$sexmale
> table(rtest.stats >= test.stat)
```

```
FALSE  TRUE
  190   810
```

```
> mean(rtest.stats >= test.stat)         # compute proportion of extreme statistics
```

```
[1] 0.81
```

```
> prop(rtest.stats >= test.stat)         # compute proportion of extreme statistics (mosaic)
```

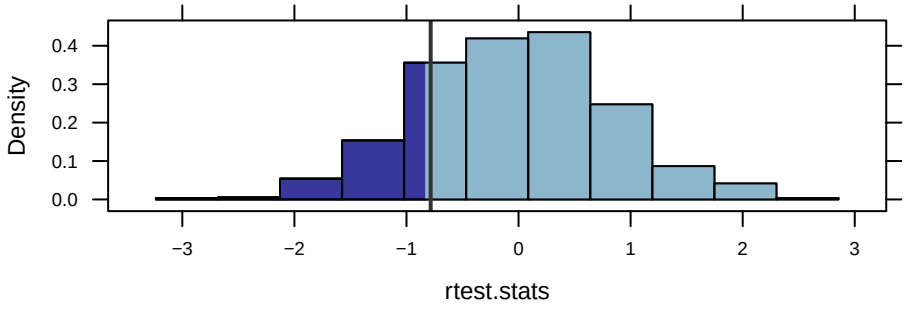
NOTE
The functions `prop()` and `count()` in the `mosaic` package are designed to make the syntax clearer for students. Typically you will see these computed using `mean()` and `sum()`.

```
prop.TRUE
     0.81
```

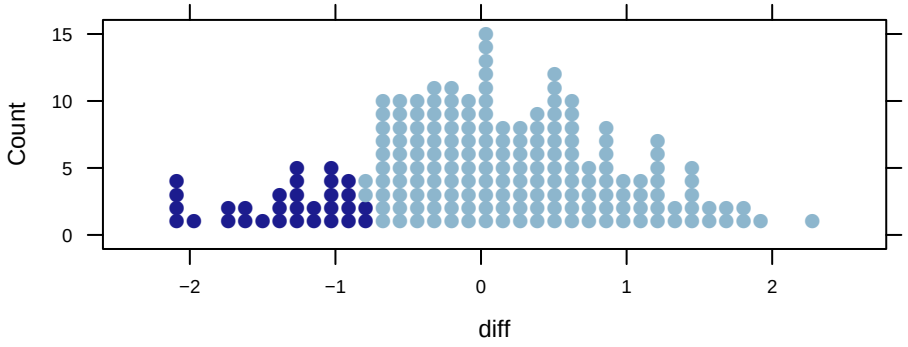
We can't reject the null hypotheses. Many samples on Planet Null have differences in the mean age of men and women at least as large as the different on Planet Sample.

We can use histograms or dotplots to display the sampling distributions graphically.

```
> xhistogram(~ rtest.stats, v=test.stat, groups=rtest.stats >= test.stat)
```




```
> test.stat <-  diff(aggregate(age ~ sex, HELP, mean )$age)
> rtest.stats <- do(200) * c(diff= diff(aggregate(age ~ shuffle(sex), HELP, mean)$age))
> dotPlot(~ diff, rtest.stats, n=40, groups=rtest.stats$diff >= test.stat, pch=16, cex=.8 )
```



Using aggregate() and friends

While `summary()` from `Hmisc` makes a nice visual display of summary statistics by groups,

```
> summary(age ~ sex, data=HELP, fun=mean)
age    N=453

+-------+------+---+----+
|       |      |N  |age |
+-------+------+---+----+
|sex    |female|107|36.3|
```



```
|       |male  |346|35.5|
+-------+------+---+----+
|Overall|      |453|35.7|
+-------+------+---+----+
```

the `aggregate()` function stores its results in a data frame that lends itself better to further computation.

> CAUTION! `summary()` uses `fun` but `aggregate()` uses `FUN`.

```
> aggregate(age ~ sex, data=HELP, FUN=mean)
```

```
     sex  age
1 female 36.3
2   male 35.5
```

To make the syntax even simpler, the `mosaic` package has modified the `mean()` function so that the following is (roughly) equivalent:

```
> mean(age ~ sex, data=HELP)
```

```
     sex    S   N Missing
1 female 36.3 107       0
2   male 35.5 346       0
3    ALL 35.7 453       0
```

The `do()` function flattens these results into a single row. This is especially useful when we add some sampling and/or permuting to the mix.

```
> do(1) * mean(age ~ sex, data=HELP)
```

```
  mean.female mean.male mean.ALL
1        36.3      35.5     35.7
```

A natural test statistic for comparing two means is the difference in sample means, which we can calculate using any of the following

```
> diff(aggregate( age ~ sex, HELP, mean )$age)                          # actual
```

```
[1] -0.784
```

```
> with(aggregate( age ~ sex, HELP, mean ), diff(age))                   # actual
```

```
[1] -0.784
```

```
> with( do(1) * mean(age ~ sex, data=HELP), mean.female - mean.male)    # actual
```

```
[1] 0.784
```

> Although we often show multiple ways to do things, you may prefer to show your students "the one true way". Better that they understand one way well than several ways poorly, or get confused among them.

Using `do()` and `shuffle()` we can now compare -0.78 with the values we obtain if we randomly shuffle the `sex` variable.

```
> with( do(1) * mean(age ~ sex, data=HELP), mean.female - mean.male)     # actual
```

```
[1] 0.784
```

```
> null.dist <- do(3) * mean(age ~ shuffle(sex), data=HELP); null.dist   # shuffled
```

```
  mean.female mean.male mean.ALL
1        35.7      35.6     35.7
2        35.4      35.7     35.7
3        35.0      35.9     35.7
```

```
> diff <- with (null.dist, mean.female - mean.male);  diff
```

```
[1]  0.0577 -0.2866 -0.8533
```

If we increase the number of random shufflings, we get an approximate sampling distribution that we can use to compute an empirical p-value.

```
> set.seed(123)                                          # re-use seed to get same results
> test.stat <- with( do(1) * mean(age ~ sex, data=HELP), mean.female - mean.male)  # actual
> null.dist <- do(1000) * mean(age ~ shuffle(sex), data=HELP); head(null.dist)     # shuffled
```

```
  mean.female mean.male mean.ALL
1        35.4      35.7     35.7
2        35.4      35.7     35.7
3        36.7      35.3     35.7
4        37.1      35.2     35.7
5        35.3      35.8     35.7
6        36.0      35.6     35.7
> rtest.stats <- with(null.dist, mean.female - mean.male)
> table(rtest.stats >= test.stat)
FALSE  TRUE
  810   190
> prop(rtest.stats >= test.stat)         # compute proportion of extreme statistics (mosaic)
prop.TRUE
     0.19
```

With an empirical p-value of approximately 0.19, there is no reason to reject the null hypothesis that the mean age is the same for men and women. It is no coincidence that this is exactly the same p-value we obtained using linear models. By using the same seed, we are obtaining the same approximation to the sampling distribution. And since both methods are computing the same test statistic, the results are identical.

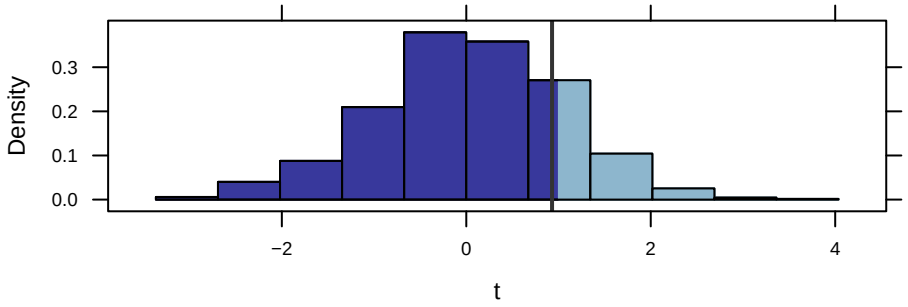
Using `t.test()`

If you prefer to teach randomization methods as an alternative to traditional methods when the assumptions of those procedures are violated, you can use simulation to approximate the sampling distribution of the usual test statistic.

```
> set.seed(123)                                  # so we get the same samples again
> data.t <- stat(t.test( age ~ sex, HELP)); data.t
   t
0.93
> null.dist <- do(1000) * stat(t.test(age ~ shuffle(sex), HELP))
> table(null.dist$t >= data.t)
FALSE  TRUE
  817   183
```

Although we have again used the same samples, notice that the p-value is now a bit different. This is because the t statistic is using the standard deviation as well as the difference in means. So this is a different test. The conclusion, however, is the same.

```
> xhistogram(~ t, null.dist, groups=t >= data.t, v=data.t)
```



6.4 Bootstrap Confidence Intervals

6.5 Power

6.5.1 Linear regression

With a specific alternative hypothesis in hand, the concept of statistical power is not hard to address. Returning to the example of vitamin D and high blood pressure, imagine we decided to do a preliminary study of size $n = 10$. What would be the power of such a study to detect the (hypothesized) relationship between vitamin D and systolic blood pressure, using the standard significance level of 0.05?

The process is pretty simple. Collect some data, compute the appropriate statistic, and see if the result is statistically significant.

```
...
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  140.594     12.269   11.46    3e-06 ***
D             -0.352      0.164   -2.15    0.064 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 14.6 on 8 degrees of freedom
Multiple R-squared: 0.365,        Adjusted R-squared: 0.286
F-statistic:  4.6 on 1 and 8 DF,  p-value: 0.0642
```

It looks like we're on the edge of statistical significance! But that's just one simulation. Let's do several, arranging things to extract the p-value from each simulation. Here's an example:

```
> summary( lm( systolic ~ D, data=planet.alt(n=10) ))$coef[2,4]
[1] 0.00741
```

The `coef[2,4]` at the end is extracting the p-value.

Now you are in a position to repeat this many times:

```
> s = do(100) * summary( lm( systolic ~ D, data=planet.alt(n=10) ))$coef[2,4]
```

The power is the fraction of trials where the p-value is less than the stated signficance level:

```
> mean( s < 0.05 )
[1] 0.76
```

6.5.2 For one sample test

6.6 Exercises, Problems, and Activities

6.1. (This isn't a question for beginning students, just one for developing instructor capabilities.) How would you demonstrate that there are 50 different states, rather than just 50 cases that might include duplicates? Try each of the following and see if you can understand what's going on in each of them:

```
> with(SAT, table(state))
> length(with(SAT, table(state)))
> with( SAT, table(table(state)))
> table(table(SAT$state))
```

6.2.

a) Write an alternative-hypothesis simulation where there are two groups: A and B. In your alternative hypothesis, the population mean of A is 90 and the population mean of B is 110. Let the standard deviation be 30 for group A and 35 for group B.

b) You've got two different suppliers of electronic components, C and D. You hypothesize that one of them, D, is somewhat defective. Their components have a lifetime that's distributed exponentially with a mean lifetime of 1000. Supplier C is better, you think, and meets the specified mean lifetime of 2000.

Write an alternative hypothesis simulation of the two suppliers.

6.3. What's the distribution of p-values under the Null Hypothesis? Many students seem to think that if the Null is true, the p-value will be large. It can help to show what's really going on.

Similar problem on deriving the shape of the F distribution.

7 Taking Advantage of the Internet

The Internet provides a wealth of data spanning the world, access to sophisticated statistical computing, and a practical means for you to communicate with your own students. In this chapter, we'll illustrate some mundane ways for you to distribute and share data and software with your students, web-based interfaces for statistical computing, as well as tools for "scraping" data from the Internet using application program interfaces (API's) or through XML (eXtensible Markup Language).

We draw your attention particularly to provocative papers by Gould [Gou10] and Nolan and Temple Lang [NT10], who highlight the importance of broadening the type of data students encounter in their first courses as well as the role of computing in modern statistics, respectively.

The wealth of data accessible to students on the internet continues to increase at what feels like an exponential rate.

7.1 Sharing With and Among Your Students

Instructors often have their own data sets to illustrate points of statistical interest or to make a particular connection with a class. Sometimes you may want your class as a whole to construct a data set, perhaps by filling in a survey or by contributing their own small bit of data to a class collection. Students may be working on projects in small groups; it's nice to have tools to support such work so that all members of the group have access to the data and can contribute to a written report.

There are now many technologies for supporting such sharing. For the sake of simplicity, we will emphasize three that we have found particularly useful both in teaching statistics and in our professional collaborative work. These are:

- A web site with minimal overhead, such as provided by Dropbox.
- The services of Google Docs.
- A web-based RStudio server for R.

The first two are already widely used in university environments and are readily accessible simply by setting up accounts. Setting up an RStudio web server requires some IT support, but is well within the range of skills found in IT offices and even among some individual faculty.

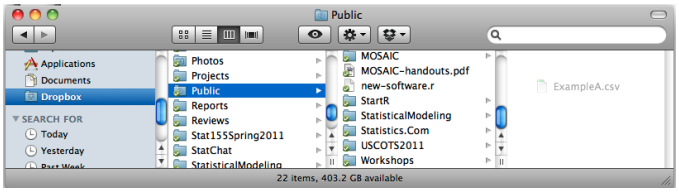


Figure 7.1: Dragging a CSV file to a Dropbox Public directory

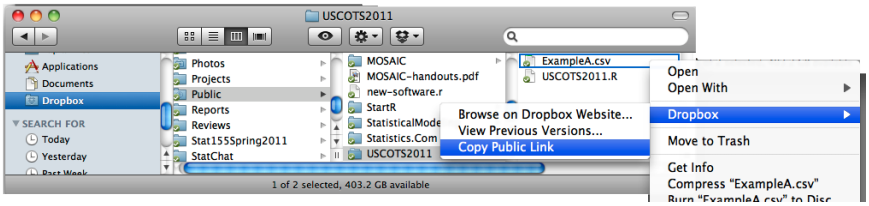


Figure 7.2: Getting the URL of a file in a Dropbox Public directory

7.1.1 Your Own Web Site

Our discussion of Dropbox is primarily for those who do not already know how to do this other ways.

You may already have a web site. We have in mind a place where you can place files and have them accessed directly from the Internet. For sharing data, it's best if this site is public, that is, it does not require a login. That rules out most "course support" systems such as Moodle or Blackboard.

The Dropbox service for storing files in the "cloud" provides a very convenient way to distribute files over the web. (Go to `dropbox.com` for information and to sign up for a free account.) Dropbox is routinely used to provide automated backup and coordinated file access on multiple computers. But the Dropbox service also provides a PUBLIC directory. Any files that you place in that directory can be accessed directly by a URL.

To illustrate, suppose you wish to share some data set with your students. You've constructed this data set in a spreadsheet and stored it as a CSV file, let's call it "example-A.csv". Move this file into the PUBLIC directory under Dropbox — on most computers Dropbox arranges things so that its directories appear exactly like ordinary directories and you'll use the ordinary familiar file management techniques as in Figure 7.1.

Dropbox also makes it straightforward to construct the web-location identifying URL for any file by using mouse-based menu commands to place the URL into the clipboard, whence it can be copied to your course-support software system or any other place for distribution to students. For a CSV file, reading the contents of the file into R can be done with the `read.csv()` function, by giving it the quoted URL:

The history feature in RStudio can be used to re-run this command in future sessions

```
> a <- read.csv("http://dl.dropbox.com/u/5098197/USCOTS2011/ExampleA.csv")
```

This technique makes it easy to distribute data with little advance preparation. It's fast enough to do in the middle of a class: the CSV file is available to your students (after a brief lag while Dropbox synchronizes). It can even be edited by you (but not by your students).

The same technique can be applied to all sorts of files: for example, R workspaces or even R scripts. Of course, your students need to use the appropriate R command: `load()` for a workspace or `source()` for a script.

It's a good idea to create a file with your course-specific R scripts, adding on to it and modifying it as the course progresses. This allows you to distribute all sorts of special-purpose functions, letting you

distribute new R material to your students. For instance, that brilliant new "manipulate" idea you had at 2am can be programmed up and put in place for your students to use the next morning in class. Then as you identify bugs and refine the program, you can make the updated software immediately available to your students.

For example, in the next section of this book we will discuss reading directly from Google Spreadsheets. It happens that we wanted to try a new technique but were not sure that it was worth including in the `mosaic` package. So, we need another way to distribute it to you. Use this statement:

```
> source("http://dl.dropbox.com/u/5098197/USCOTS2011/USCOTS2011.R")
```

Among other things, the operator `readGoogleCSV()` is defined in the script that gets sourced in by that command. Again, you can edit the file directly on your computer and have the results instantly available (subject only to the several second latency of Dropbox) to your students. Of course, they will have to re-issue the `source()` command to re-read the script.

CAUTION!
Security through Obscurity of this sort will not generally satisfy institutional data protection regulations

If privacy is a concern, for instance if you want the data available only to your students, you can effectively accomplish this by giving files names known only to your students, e.g., "Example-A78r423.csv".

7.1.2 GoogleDocs

The Dropbox technique is excellent for broadcasting: taking files you create and distributing them in a read-only fashion to your students. But when you want two-way or multi-way sharing of files, other techniques are called for, such as provided by the GoogleDocs service.

GoogleDocs allows students and instructors to create various forms of documents, including reports, presentations, and spreadsheets. (In addition to creating documents *de novo*, Google will also convert existing documents in a variety of formats.)

Once on the GoogleDocs system, the documents can be edited *simultaneously* by multiple users in different locations. They can be shared with individuals or groups and published for unrestricted viewing and even editing.

For teaching, this has a variety of uses:

- Students working on group projects can all simultaneously have access to the report as it is being written and to data that is being assembled by the group.
- The entire class can be given access to a data set, both for reading and for writing.
- The Google Forms system can be used to construct surveys, the responses to which automatically populate a spreadsheet that can be read by the survey creators.
- Students can "hand in" reports and data sets by copying a link into a course support system such as Moodle or Blackboard, or emailing the link.
- The instructor can insert comments and/or corrections directly into the document.

An effective technique for organizing student work and ensuring that the instructor (and other graders) have access to it, is to create a separate Google directory for each student in your class (Dropbox can also be used in this manner). Set the permission on this directory to share it with the student. Anything she or he drops into the directory is automatically available to the instructor. The student can also share with specific other students (e.g., members of a project group).

Example 7.1. One exercise for students starting out in a statistics course is to collect data to find out whether the "close door" button on an elevator has any effect. This is an opportunity to introduce simple ideas of experimental design. But it's also a chance to teach about the organization of data.

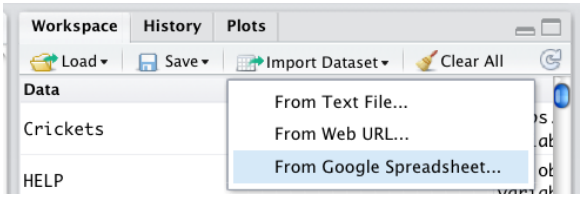
Have your students, as individuals or small groups, study a particular elevator, organize their data into a spreadsheet, and hand in their individual spreadsheet. Then review the spreadsheets in class. You will likely find that many groups did not understand clearly the distinction between cases and variables, or coded their data in ambiguous or inconsistent ways.

Work with the class to establish a consistent scheme for the variables and their coding, e.g., a variable `ButtonPress` with levels "Yes" and "No", a variable `Time` with the time in seconds from a fiducial time (e.g. when the button was pressed or would have been pressed) with time measured in seconds, and variables `ElevatorLocation` and `GroupName`. Create a spreadsheet with these variables and a few cases filled in. Share it with the class.

Have each of your students add his or her own data to the class data set. Although this is a trivial task, having to translate their individual data into a common format strongly reinforces the importance of a consistent measurement and coding system for recording data. ◇

Once you have a spreadsheet file in GoogleDocs, you will want to open it in R. Of course, it's possible to export it as a CSV file, then open it using the CSV tools in R, such as `read.csv()`. But there are easier ways that let you work with the data "live."

In the web-server version of RStudio, described below, you can use a menu item to locate and load your spreadsheet.



If you are using other R interfaces, you must first use the Google facilities for publishing documents.

1. From within the document, use the "Share" dropdown menu and choose "Publish as a Web Page."
2. Press the "Start Publishing" button in the "Publish to the web" dialog box. (See figure 7.3.)
3. In that dialog box, go to "Get a link to the published data." Choose the CSV format and copy out the link that's provided. You can then publish that link on your web site, or via course-support software. Only people with the link can see the document, so it remains effectively private to outsiders.

It turns out that communicating with GoogleDocs requires facilities that are not present in the base version of R, but are available through the `RCurl` package. In order to make these readily available to students, we have created a function that takes a quoted string with the Google-published URL and reads the corresponding file into a data frame:

```
> elev <- readGoogleCSV(
 "https://spreadsheets.google.com/spreadsheet/pub?hl=en&hl=en&key=0Am13enSalO74dEVzMGJSMU5TbTc2eWlWakppQlpj
> head(elev)
  StudentGroup      Elevator CloseButton Time Enroute LagToPress
1           HA Campus Center           N 8.23       N          0
2           HA Campus Center           N 7.57       N          0
3           HA Campus Center           N 7.80       N          0
```

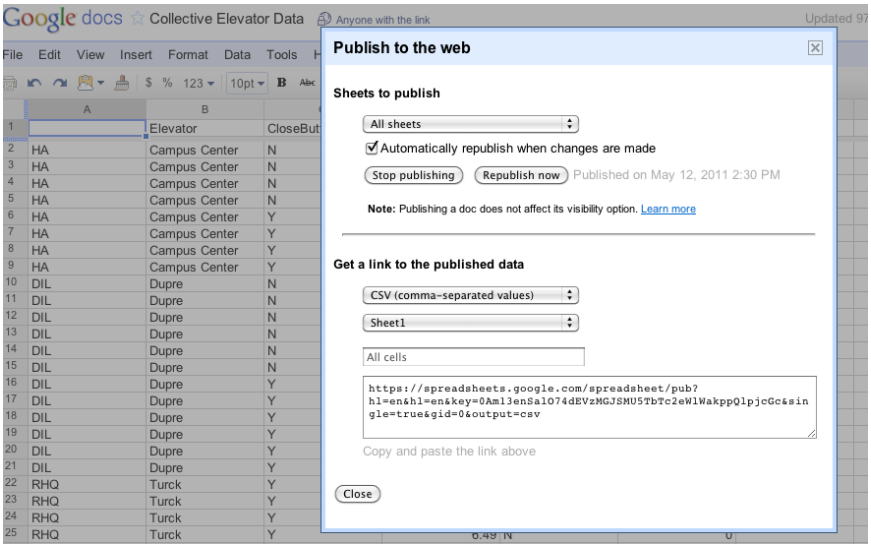


Figure 7.3: Publishing a Google Spreadsheet so that it can be read directly into R.

```
4          HA Campus Center          N 8.30       N          0
5          HA Campus Center          Y 5.81       N          0
6          HA Campus Center          Y 6.60       N          0
```

Of course, you'd never want your students to type that URL by hand; you should provide it in a copy-able form on a web site or within a course support system.

Note that the `readGoogleCSV()` function is not part of the `mosaic` package. As described previously, we make it available via an R source file that can be read into the current session of R using the `source()` command:

```
> source("http://dl.dropbox.com/u/5098197/USCOTS2011/USCOTS2011.R")
```

7.1.3 The RStudio Web Server

RStudio is available as a desktop application that provides a considerately designed interface to the standard R software that you can install on individual computers.

But there is another version of RStudio available, one that takes the form of a web server. There are some substantial advantages to using the web-server version.

- For the user, no installation is required beyond a standard web browser.
- Sessions are continued indefinitely; you can come back to your work exactly where you left it.
- A session started on one computer can be continued on another computer. So a student can move seamlessly from the classroom to the dorm to a friend's computer.
- The web-server system provides facilities for direct access to GoogleDocs.

As RStudio continues to be developed, we anticipate facilities being added that will enhance even more the ability to teach with R:

CAUTION!
These are anticipated future features.

- The ability to create URLs that launch RStudio and read in a data set all in a single click.

- The ability to share sessions simultaneously, so that more than one person can be giving commands. This will be much like Google Docs, but with the R console as the document. Imagine being able to start a session, then turn it over to a student in your classroom to give the commands, with you being able to make corrections as needed.
- The ability to clone sessions and send them off to others. For instance, you could set up a problem then pass it along to your students for them to work on.

But even with the present system, the web-based RStudio version allows you to work with students effectively during office hours. You can keep your own version of RStudio running in your usual browser, but give your visiting a student a window in a new browser: Firefox, Chrome, Safari, Internet Explorer, etc. Each new browser is effectively a new machine, so your student can log in securely to his or her own account.

7.2 Data Mining Activities

We end this chapter with several examples that do data mining via the Internet. Some of these are mere glimpses into what might be possible as tools for accessing this kind of data become more prevalent and easier to use.

7.2.1 What percentage of Earth is Covered with Water?

We can estimate the proportion of the world covered with water by randomly sampling points on the globe and inspecting them using GoogleMaps.

First, let's do a sample size computation. Suppose we want to estimate (at the 95% confidence level) this proportion within ±5%. There are several ways to estimate the necessary sample size, including algebraically solving

$$(1.96)\sqrt{\hat{p}(1-\hat{p})/n} = 0.05$$

for n given some estimated value of $\hat{p}$. The `uniroot()` function can solve this sort of thing numerically. Here we take an approach that looks at a table of values of n and $\hat{p}$ and margin of error.

```
> n <- seq(50,500, by=50)
> p.hat <- seq(.5, .9, by=0.10)
> margin_of_error <- function(n, p, conf.level=.95) {
        -qnorm( (1-conf.level)/2) * sqrt( p * (1-p) / n )
 }
> # calculate margin of error for all combos of n and p.hat
> outer(n, p.hat, margin_of_error) -> tbl
> colnames(tbl) <- p.hat
> rownames(tbl) <- n
> tbl
       0.5    0.6    0.7    0.8    0.9
50  0.1386 0.1358 0.1270 0.1109 0.0832
100 0.0980 0.0960 0.0898 0.0784 0.0588
150 0.0800 0.0784 0.0733 0.0640 0.0480
200 0.0693 0.0679 0.0635 0.0554 0.0416
250 0.0620 0.0607 0.0568 0.0496 0.0372
300 0.0566 0.0554 0.0519 0.0453 0.0339
350 0.0524 0.0513 0.0480 0.0419 0.0314
400 0.0490 0.0480 0.0449 0.0392 0.0294
450 0.0462 0.0453 0.0423 0.0370 0.0277
500 0.0438 0.0429 0.0402 0.0351 0.0263
```

From this it appears that a sample size of approximately 300–400 will get us the accuracy we desire. A class of students can easily generate this much data in a matter of minutes if each student inspects 10–20 maps. The example below assumes a sample size of 10 locations per student. This can be adjusted depending on the number of students and the desired margin of error.

1. Generate 10 random locations.

```
> positions <- rgeo(10); positions
```

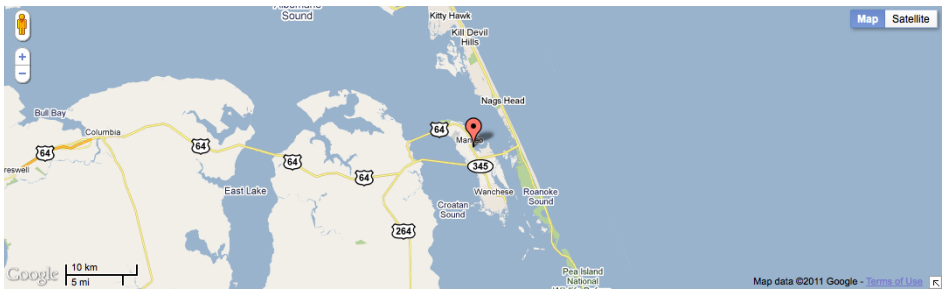
```
      lat     lon
1  -51.85 -123.68
2   13.31  -35.18
3    4.36   -3.64
4  -16.92  123.95
5  -32.44  114.40
6   64.66  -16.17
7  -23.30   41.12
8   -7.99 -107.23
9  -51.99   92.64
10 -62.69 -101.72
```

2. Open a GoogleMap centered at each position.

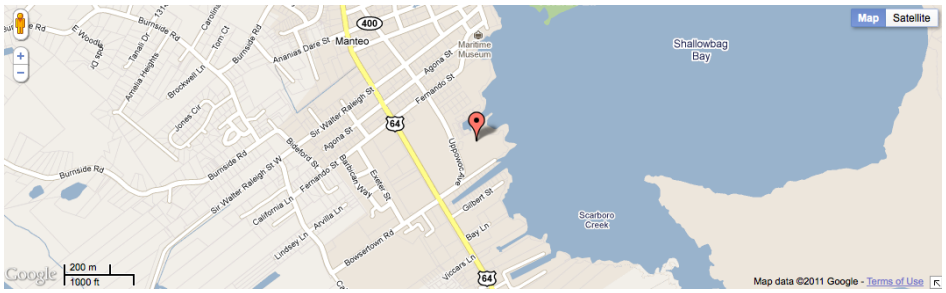
```
> googleMap(pos=positions, mark=TRUE)
```

   You may need to turn off pop-up blocking for this to work smoothly.

3. For each map, record whether the center is located in water or on land. The options `mark=TRUE` is used to place a marker at the center of the map (this is helpful for locations that are close to the coast).

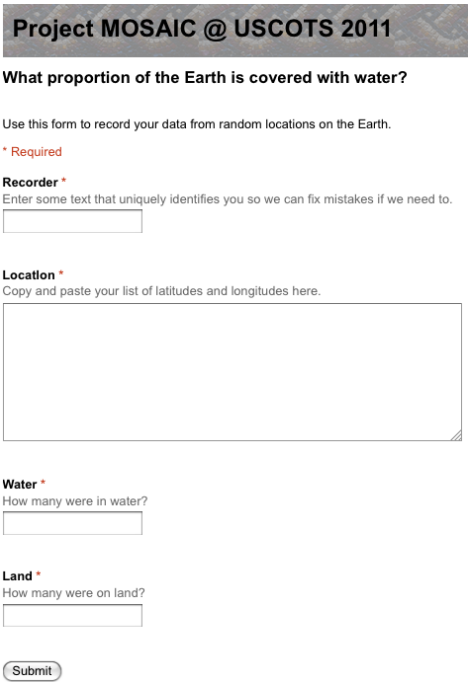


   You can zoom in or out to get a better look.



4. Record your data in a GoogleForm at

   `http://mosaic-web.org/uscots2011/google-water.html`



CAUTION!
This sort of copy-and-paste operation works better in some browsers (Firefox) than in others (Safari).

For the latitude and longitude information, simply copy and paste the output of

```
> positions
```

5. After importing the data from Google, it is simple to sum the counts across the class.

```
> sum(googleData$Water)
[1] 215
> sum(googleData$Land)
[1] 85
```

Then use your favorite method of analysis, perhaps `binom.test()`.

```
> interval(binom.test(215, 300))   # numbers of successes and trials
Method: Exact binomial test

probability of success
                0.717

95% confidence interval:
0.662 0.767
```

7.2.2 Roadless America

The `rgeo()` function can also sample within a latitude longitude "rectangle". This allows us to sample subsets of the globe. In this activity we will estimate the proportion of the continental United States that is within 1 mile of a road.

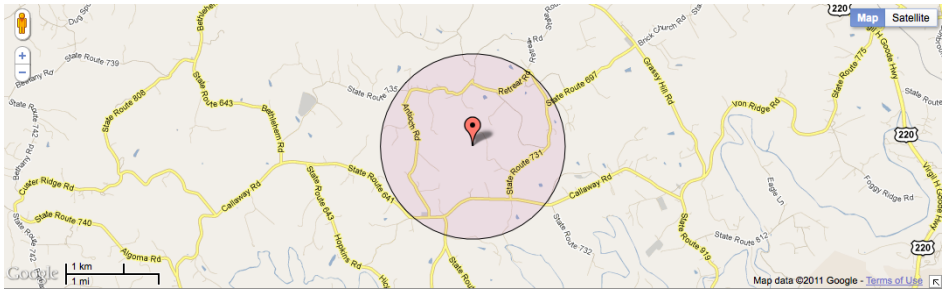
1. Generate a random sample of locations in a box containing the continental United States. Some of these points may be in Canada, Mexico, an ocean or a major lake. These will be discarded from our sample before making our estimate.

```
> positions <- rgeo(10, lonlim=c(-125,-65), latlim=c(25,50)); positions
```

```
    lat    lon
1  43.1 -123.2
2  49.9  -73.9
3  30.8  -89.6
4  28.6 -112.5
5  34.0  -76.0
6  26.2  -81.6
7  32.3 -112.4
8  44.2 -109.1
9  33.8  -67.3
10 42.9  -97.9
```

2. Open a GoogleMap centered at each position. This time we'll zoom in a bit and add a circle of radius 1 to our map.

```
> googleMap(pos=positions, mark=TRUE, zoom=12, radius=1)
```



You may need to turn off pop-up blocking for this to work smoothly.

3. For each map, record whether the center is close (to a road), far (from a road), water, or foreign. You may need to zoom in or out a bit to figure this out.

7.2.3 Variations on the Google Maps theme

There are many other quantities one could estimate using these tools. For example:

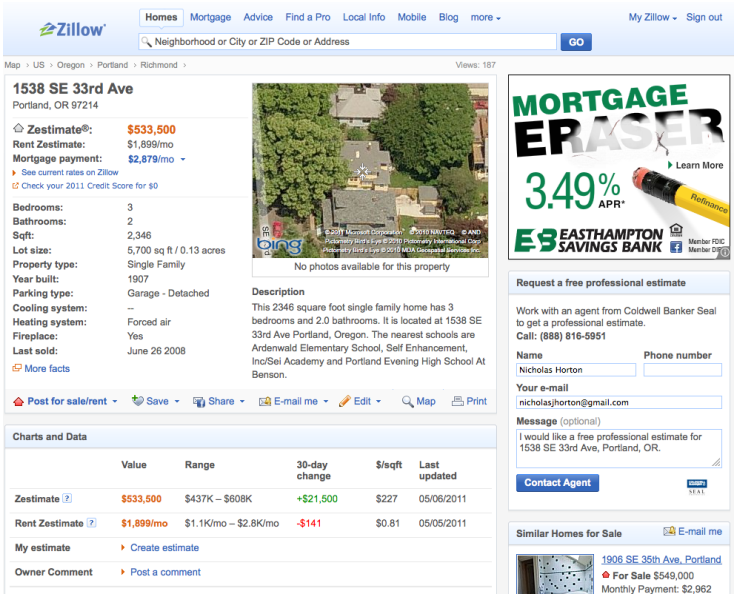
1. What proportion of your home state is within m miles of a lake? (The choice of m may depend upon your state of interest.)

2. Use two proportion procedures or chi-squared tests to compare states or continents. Do all continents have roughly the same proportion of land withing m miles of water (for some m)? Are Utah and Arizona equally roadless?

3. In more advanced classes: What is the average distance to the nearest lake (in some region)? By using concentric circles, one could estimate this from discretized data indicating, for example, whether the nearest lake is within 1/2 mile, between 1/2 mile and 1 mile, between 1 mile and 2 miles, between 2 miles, and 4 miles, between 4 miles and 10 miles, or more than 10 miles away. It may be interesting to discuss what sort of model should be used for distances from random locations to lakes. (It probably isn't normally distributed.)

7.2.4 Zillow

Zillow.com is an online real estate database that can be used to estimate property values using tax records, sales data, and comparable homes.



The folks who run the site have made an application programming interface (API) that specifies how software programs can interface with their system. Duncan Temple Lang has crafted a package in R that talks to Zillow. This can be used to dynamically generate datasets for use in courses, after you (and/or your students) generate a `zillowId` for use with the system. (Danny Kaplan has used `cars.com` to similar ends).

While this is a cool interface, students tend to be less interested in house prices than their instructors!

In this section, we describe how to use Zillow to generate and analyse a dataset comprised of comparable sites to an arbitrary house of interest.

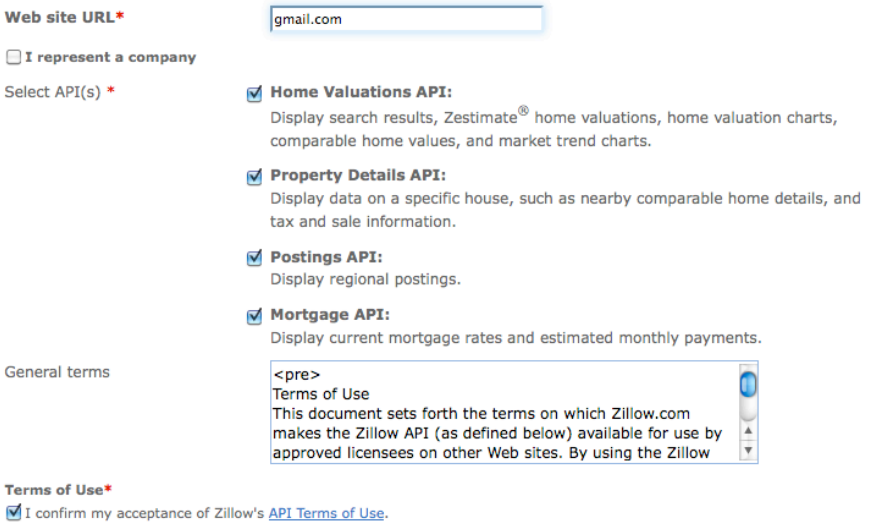
SUGGESTION BOX
`Zillow` is new to the authors and we are still looking for the "killer activity" using `Zillow`. We wonder if it is possible, for example, to sample uniformly among all houses in a city or zip code.

The first step is to create a Zillow account (click on `Register` on the top right of the page at `zillow.com`). You can set up an account or register using Facebook.

Once you have the account, log in, then click on `My Zillow` at the top right. This should display your profile (in this case, for a user named `SC_z`).

Next, open the page: `http://www.zillow.com/webservice/Registration.htm`. This is the application programming interface (API) request, which requires more information if you are a real estate professional. Note that there are limits on the use of these data, which at first glance appear to not

contravene use for statistics activities and data collection. An overview of the API and terms of use can be found at `http://www.zillow.com/howto/api/APIOverview.htm`.



You should receive information about your Zillow ID (a character string of letters and numbers).

Once you've set up your Zillow account, and gotten your Zillow Id, the next step is to install the `Zillow` package. This package is not on CRAN, but can be obtained from Omegahat (a different repository from CRAN) using

```
> install.packages("RCurl")
> install.packages("Zillow", repos="http://www.omegahat.org/R", type="source",
   dependencies="Depends")
```

Next, you should initialize your `zillowID` to the value that you received when you registered with `Zillow.com`.

```
> zillowId <- "set_to_your_zillowId"
```

This allows you to make calls to functions such as `zestimate()` (which allows you to search for information about a particular property) and `getComps()` (which facilitates finding a set of comparable properties. Here we find information about an arbitrary house in California, as well as comparable properties.

```
> require(Zillow)
> est <- zestimate("1280 Monterey Avenue", "94707", zillowId)
> est
         amount    low   high valueChange30Day
24842790  7e+05 623000 819000             7500
> comps <- getComps(rownames(est), zillowId)
> rownames(est)
[1] "24842790"
> names(comps)
 [1] "amount"            "low"               "high"              "valueChange30Day"
 [5] "street"            "zip"               "latitude"          "longitude"
 [9] "taxAssessmentYear" "taxAssessment"     "yearBuilt"         "lotSizeSqFt"
[13] "finishedSqFt"      "bathrooms"         "bedrooms"          "lastSoldPrice"
[17] "lastSold"          "score"
> table(comps$bathrooms)
   1 1.25  1.5    2
  21    1    3    6
```

```
> perctable(comps$bathrooms)
    1  1.25   1.5     2
67.74  3.23  9.68 19.35
> table(comps$bedrooms)
 2  3  4
10 20  1
> perctable(comps$bedrooms)
    2     3     4
32.26 64.52  3.23
> favstats(comps$finishedSqFt)
 min   Q1 median   Q3  max mean  sd  n missing
 886 1198   1291 1465 1870 1321 247 31       0
```

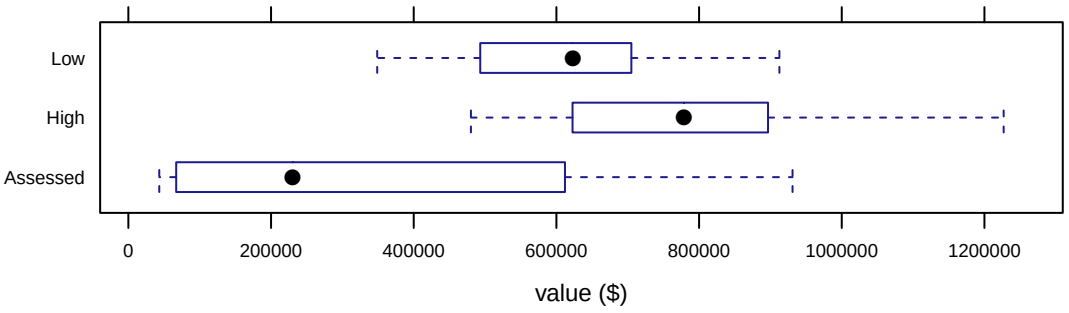
We can compare numerical summaries of the size of the house for houses with different numbers of bedrooms:

```
> require(Hmisc)
> summary(finishedSqFt ~ bedrooms, data=comps, fun=favstats)
finishedSqFt    N=31

+--------+-+--+----+----+-------+----+----+-----+---+--+--------+
|        | |N |min |Q1  |median |Q3  |max |mean |sd |n |missing |
+--------+-+--+----+----+-------+----+----+-----+---+--+--------+
|bedrooms|2|10| 886| 933|1131   |1269|1397|1126 |196|10|0       |
|        |3|20|1054|1273|1329   |1509|1870|1399 |210|20|0       |
|        |4| 1|1709|1709|1709   |1709|1709|1709 |   | 1|0       |
+--------+-+--+----+----+-------+----+----+-----+---+--+--------+
|Overall | |31| 886|1198|1291   |1465|1870|1321 |247|31|0       |
+--------+-+--+----+----+-------+----+----+-----+---+--+--------+
```

This syntax is somewhat dense, since the `bwplot()` function is expecting a data frame, not 3 vectors

We can look at the distribution of Zillow price lower bound, upper bound, as well as assessed (tax) value.

```
> bwplot(rep(c("Low", "Assessed", "High"), each=nrow(comps)) ~ c(low, taxAssessment, high),
   data=comps, horizontal=TRUE, xlab='value ($)')
```



the `reshape` package also provides support for restructuring datasets

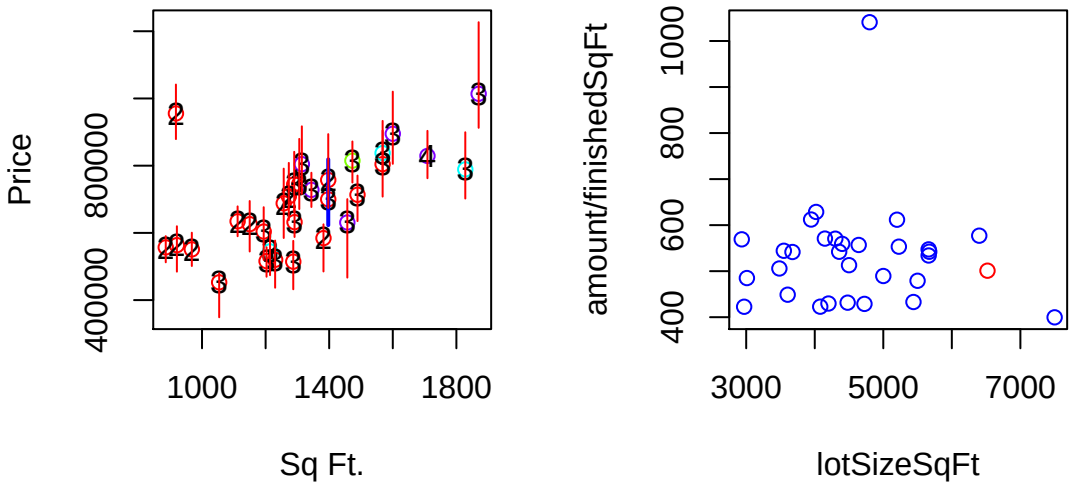
As an alternative to the code above, we could first build a data frame and then use that for our plot.

```
> zillowData <- data.frame(
        value = with(comps, c( low, taxAssessment, high )),
        type  = rep(c("Low", "Assessed", "High"), each=nrow(comps))
        )
> bwplot( value ~ type, zillowData)
```


It's interesting that for these properties, assessed values tend to be lower that both the lower and upper Zillow estimates. We could explore whether this is true in California more generally.

It's possible to plot the results of our comparable properties, which yields a scatterplot of price by square feet (with the number of bedrooms as well as the low and high range) as well as a scatterplot of amount/finished square feet vs log size in square feet.

```
> plot(comps)
```



Several aspects of this activity are worth noting:

1. There is some startup cost for instructors and students (since each user will need their own ZillowID[1]).
2. Once set up, the calls to the Zillow package are very straightforward, and provide immediate access to a wealth of interesting data.
3. This could be used as an activity to provide descriptive analysis of comparable properties, or in a more elaborate manner to compare properties in different cities or areas.
4. Since the latitude and longitude of the comparable properties is returned, users can generate maps using the mechanisms described in A.1.

7.2.5 Twitter

Twitter (`twitter.com`) is a social networking and microblogging service, where users (estimated at 200 million as of May, 2011) can send and read posts of up to 140 characters. Approximately 65 million "tweets" are posted per day.

The `twitteR` package (due to Jeff Gentry) implements a Twitter client for R. This can be used to generate data for student projects, class assignments or data mining.

CAUTION!
Pulling live data from Twitter may generate some unsavory content

[1] By default, the number of calls per day to the API is limited to 1000, which could easily be exceeded in a lab setting if, contrary to the terms of use, the Zillow ID were to be shared.

Table 7.1: Functions available within the twitteR package

Function	Description
getUser()	returns an object of class user
userFriends()	returns lists of class user
userFollowers()	returns a list of class user
searchTwitter()	returns a list of class status
Rtweets()	search for #rstats
showstatus()	take a numeric ID of a tweet and return it
publicTimeline()	current snapshot of the public timeline
userTimeline()	current snapshot of the users public timeline

The package provides a number of interface functions to connect with the service (see Table 7.1 for details).

Some examples may help to better understand the interface. Let's start by seeing what the U.S. Census Bureau has been up to recently.

```
> require(twitteR)
> census <- getUser("uscensusbureau")
> census$getName()
[1] "U.S. Census Bureau"
> census$getDescription()
[1] "Your trusted source for quality data about our people and our economy."
> census$getUrl()
[1] "http://2010census.gov"
> census$getStatusesCount()
[1] 1662
> census$getCreated()
[1] "2009-03-06 17:39:21 UTC"
> census$getFollowersCount()
[1] 12933
```

That's a lot of followers (but then again, the census is a big job). Detailed reports of the last set of twitter status updates can be returned.

```
> census.tweets <- userTimeline(census, n=5)
```

By default, the tweets don't print very nicely. So let's define our own function to extract the text from a tweet and format it more nicely.

```
> printTweet <- function(x) paste( strwrap(x$getText()))
> printTweet(census.tweets[[1]])
[1] "@GovTwit Want to learn more about #Veteran owned businesses in the US? Join"
[2] "us live @Ustream Wed 2PM EDT http://tiny.cc/skt28"
```

Detailed information can be found for individual tweets.

```
> census.tweets[[1]]$getCreated()
[1] "2011-05-16 14:42:38 UTC"
> census.tweets[[1]]$getId()
[1] "70137081795584000"
```

Digging Deeper
Section B.4 provides details on accessing lists.

Using sapply(), we can inspect all of the tweets at once.

```
> sapply(census.tweets, printTweet)
     [,1]
[1,] "@GovTwit Want to learn more about #Veteran owned businesses in the US? Join"
[2,] "us live @Ustream Wed 2PM EDT http://tiny.cc/skt28"
     [,2]
[1,] "@DeptofDefense We hope you will join us live Wed 2PM EDT for a @Ustream"
[2,] "event on #Veteran owned businesses. http://tiny.cc/skt28"
     [,3]
[1,] "@VHVNow We hope you will join us live Wed 2PM EDT for a @Ustream event on"
[2,] "#Veteran owned businesses. http://tiny.cc/skt28"
     [,4]
[1,] "@TodaysGIBill We hope you will join us live Wed 2PM EDT for a @Ustream"
[2,] "event on #Veteran owned businesses. http://tiny.cc/skt28"
```

Alternatively, we can download information about how active Census Bureau followers are in terms of posting their own status updates.

```
> census.followers <- userFollowers(census)
> followers.ntweets <- sapply(census.followers, function(x) x$getStatusesCount())
> favstats(followers.ntweets)
 min Q1 median  Q3   max mean   sd   n missing
   0 11   63.5 390 14501  670 2040 100       0
> census.followers[[which.max(followers.ntweets)]]
[1] "augustine25"
```

That's probably a bit much for students to swallow. Let's write a function to hide some of the gory details from the students.

```
> CountTweets <- function ( users ) {
        return( sapply( users,  function(x) x$getStatusesCount()) )
 }
```

Now we can count tweets like this.

```
> sort(CountTweets(census.followers))
  [1]     0     0     0     0     0     0     0     0     0     0     0     0     0
 [14]     1     1     2     3     3     4     4     4     7     7    10    11    11
 [27]    13    13    15    16    17    17    19    23    24    25    25    28    28
 [40]    31    34    35    37    40    42    46    49    51    57    61    66    66
 [53]    68    69    70    71    91    94    99   103   115   121   122   131   131
 [66]   205   211   240   251   273   274   282   309   366   385   403   408   434
 [79]   473   495   549   563   598   738   739   753  1047  1109  1126  1338  1609
 [92]  1748  1913  1984  2212  3748  4936  7308 11250 14501
```

We can also see what's happening on the R front:

```
> R.tweets <- Rtweets(n=5)    # short for searchTwitter("#rstats", n=5)
> sapply(R.tweets, printTweet)
[[1]]
[1] "Kevin Quinn talks about #rstats, the #IBI R integration, and covers some"
[2] "very cool examples, e.g. predicting gambling 'pain point' #arraybi"

[[2]]
[1] "@hadleywickham good luck! And, as always, thanks for your time &amp;amp;"
[2] "contributions to #rstats that we benefit from"

[[3]]
[1] "squashing bugs in preparation for the next release of ggplot2 #rstats"
```

Digging Deeper the `sapply()` applies a function to each element of a list or vector. It is one of several "apply" functions in R, including `apply()`, and `sapply()`, and `lapply()`, and `tapply()`. The `mosaic` package even include `dfapply()` for application to data frames.

```
[[4]]
[1] "#rstats Use R! 2011 provisional Abstract book."
[2] "http://www.warwick.ac.uk/statsdept/user-2011/draft_booklet.pdf"

[[5]]
[1] "getting terribly confused with my indexing variables in a three-d array :-("
[2] "#rstats"
```

Finally, we can see who has been tweeting the most at USCOTS:

Support functions could be written to simplify this interface for student use. Just another thing for us to do in our spare time...

```
> sort(table(sapply(searchTwitter("#uscots11"), function(x) x$getScreenName())),
   decreasing=TRUE)

   MGEverson    ekendriss georgettenic       gok9go
           3            1            1            1
```

Further support for posting tweets requires use of the Twitter API (with appropriate authentication). More details are provided in the twitteR vignette file.

7.2.6 Other ideas

We've outlined several approaches which efficiently scrape data from the web. But there are lots of other examples which may be worth exploring (and this is clearly a growth area). These include:

RNYTimes interface to several of the *New York Times* web services for searching articles, meta-data, user-generated content and best seller lists (`http://www.omegahat.org/RNYTimes`)

Rflickr interface to the Flickr photo sharing service (`http://www.omegahat.org/Rflickr`)

RGoogleDocs interface to allow listing documents on Google Docs along with details, downloading contents, and uploading files (`http://www.omegahat.org/RGoogleDocs`)

RLastFM interface to the `last.fm` music recommendation site (`http://cran.r-project.org/web/packages/RLastFM`)

8 The Core of a Traditional Course

In this chapter, we will briefly review the commands and functions needed to analyze data from a more traditional introductory statistics course. We will use data from the HELP study: a randomized trial of a novel way to link at-risk subjects with primary care. More information on the dataset can be found in section 9.1.

Some data management is needed by students (and more by instructors). This material is reviewed in section B.6.

Since the selection and order of topics can vary greatly from textbook to textbook and instructor to instructor, we have chosen to organize this material by the kind of data being analyzed. This should make it straightforward to find what you are looking for even if you present things in a different order. This is also a good organizational template to give your students to help them keep straight "what to do when".

Just what is the *traditional course*, anyways?

It's worth noting how we teach the "Traditional Course", which, contrary to rumor, some of us do on occasion. Nick has used the 5th edition of Moore and McCabe [MM06], where he introduces least squares regression at the end of the first week of a semester, and multiple regression (purely descriptively) at the end of the second week. Design is the next main topic, then a brutally pruned intro to probability and sampling distributions. Interval estimation and testing are introduced in one or two settings, then the class returns to inference for multiple regression. While students are working on projects involving a multiple linear regression model with 2 predictors, some categorical data analysis is covered.

8.1 One Quantitative Variable

8.1.1 Numerical Summaries

R includes a number of commands to numerically summarize variables. These include the capability of calculating the mean, standard deviation, variance, median, five number summary, intra-quartile range (IQR) as well as arbitrary quantiles. We will illustrate these using the CESD (Center for Epidemiologic Studies - Depression) measure of depressive symptoms. To improve the legibility of output, we will also set the default number of digits to display to a more reasonable level (see `?options` for more configuration possibilities).

```
> options(digits=3)
> mean(HELP$cesd)
```

```
[1] 32.8

> sd(HELP$cesd)

[1] 12.5

> sd(HELP$cesd)^2; var(HELP$cesd)

[1] 157

[1] 157
```

It is also straightforward to calculate quantiles of the distribution.

```
> median(HELP$cesd)

[1] 34

> five <- fivenum(HELP$cesd)
> five     # display the object (vector of length 5)

[1]  1 25 34 41 60
```

CAUTION!
Note that the hinges are not necessarily the same as the quartiles, due to interpolation in small datasets

The user can interact with the objects created by these functions to calculate the IQR (or the built-in a function can generate this directly).

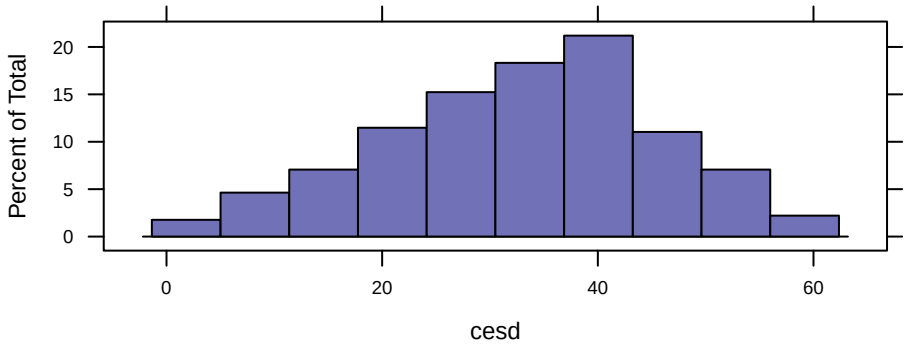
```
> five[4] - five[2]

[1] 16

> IQR(HELP$cesd)

[1] 16

> quantile(HELP$cesd)

  0%  25%  50%  75% 100%
   1   25   34   41   60

> quantile(HELP$cesd, c(.25, .50))

25% 50%
 25  34

> favstats(HELP$cesd)

 min Q1 median Q3 max mean   sd   n missing
   1 25     34 41  60 32.8 12.5 453       0
```

By default, the `quantile()` function displays the quartiles, but can be given a vector of quantiles to display. Finally, the `favstats()` function in the `mosaic` package provides a concise summary of many useful statistics.

8.1.2 Graphical Summaries

The `histogram()` function is used to create a histogram. Here we use the formula interface (as discussed in section 4.2) to specify that we want a histogram of the CESD scores.

```
> histogram(~ cesd, data=HELP)
```



In the HELP dataset, approximately one quarter of the subjects are female.

```
> with( HELP, table(sex) )

sex
female   male
   107    346
```

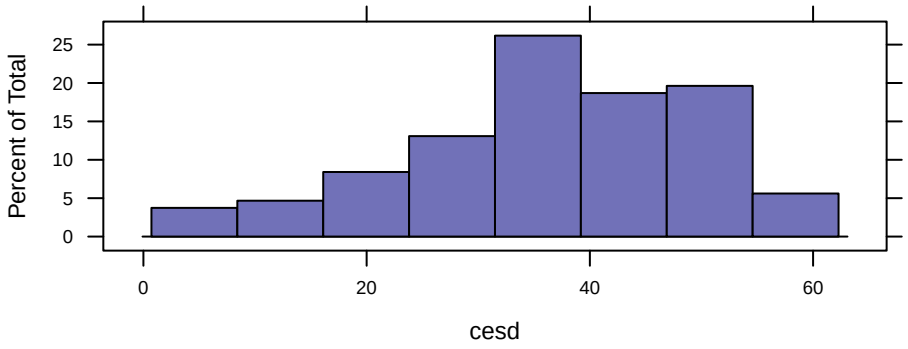
It is straightforward to restrict our attention to just those subjects. If we are going to do many things with a subset of our data, it may be easiest to make a new data frame containing only the cases we are interested in. The `subset()` function can generate a new data frame containing just the women or just the men (see also section B.6.4). Once this is created, we used the `stem()` function to create a stem and leaf plot.

CAUTION!
Note that the equality operator is *two* equal signs

```
> female <- subset(HELP, sex=='female')
> male <- subset(HELP, sex=='male')
> stem(female$cesd)

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

  0 | 3
  0 | 567
  1 | 3
  1 | 555589999
  2 | 123344
  2 | 66889999
  3 | 0000233334444
  3 | 5556666777888899999
  4 | 00011112222334
  4 | 555666777889
  5 | 011122222333444
  5 | 67788
  6 | 0
```

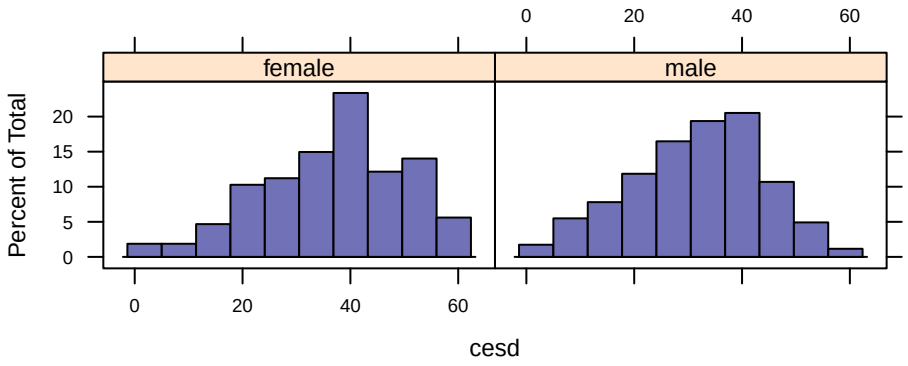
The plotting functions in the `lattice` package have a `subset` argument that allows us to select only a portion of the data. This can be handy for explaration and avoids the need to make (and name) various subsets.

```
> histogram(~ cesd, data=HELP, subset=sex=='female')   # notice the quotes
```



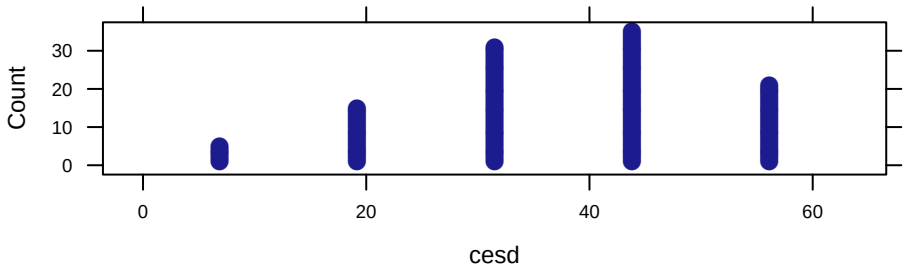
Alternatively, we can make side-by-side plots to compare multiple subsets.

```
> histogram(~ cesd | sex, data=HELP)
```



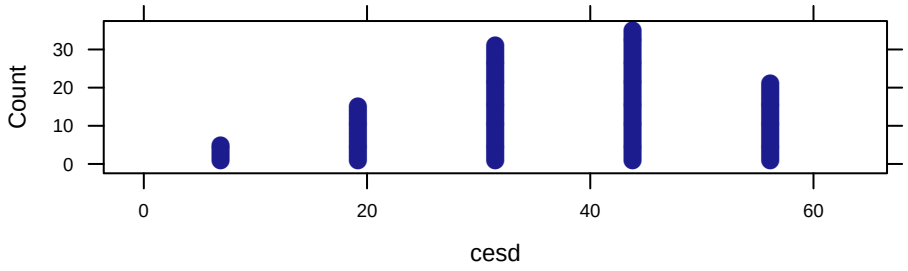
The `dotPlot()` function is used to create a dotplot (a la Fathom).

```
> dotPlot(~ cesd, data=female)
```



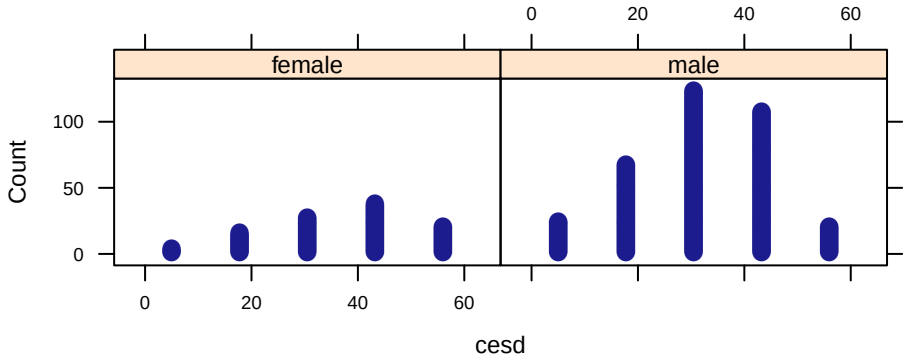
We could also have made our subset "on the fly", just for the purposes of graphing:

```
> dotPlot(~ cesd, data=HELP, subset=(sex=='female'))
```



Or we could make side-by-side dotplots of men and women:

```
> dotPlot(~ cesd | sex, data=HELP)
```

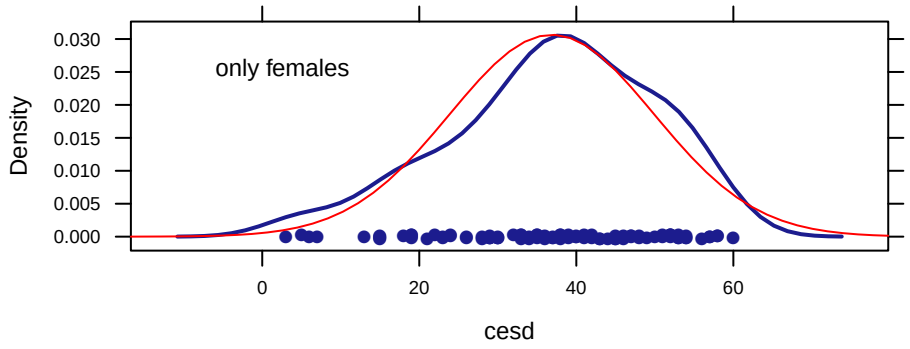


8.1.3 Density Curves

One disadvantage of histograms is that they can be sensitive to the choice of the number of bins. Another display to consider is a density curve.

Density plots are also sensitive to certain choices. If your density plot is too jagged or too smooth, try adjusting the `adjust` argument (larger than 1 for smoother plots, less than 1 for more jagged plots.

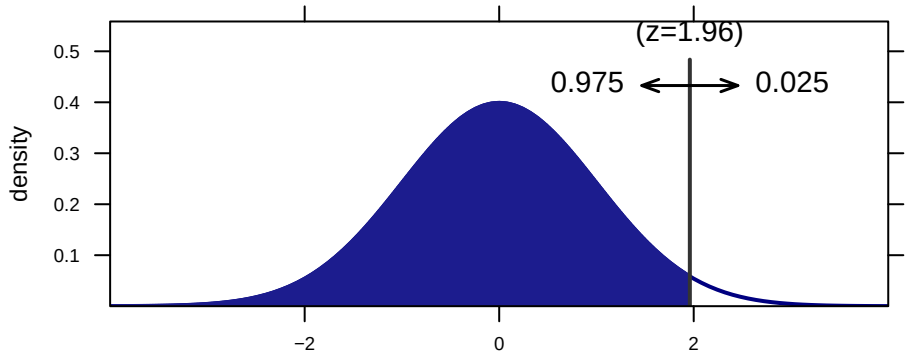
```
> densityplot(~ cesd, data=female)
> ladd(grid.text(x=0.2, y=0.8, 'only females'))
> ladd( panel.mathdensity(args=list(mean=mean(female$cesd), sd=sd(female$cesd)),
   col="red"))
```



The most famous density curve is a normal distribution. The `xpnorm()` function displays the probability that a random variable is less than the first argument, for a normal distribution with mean given by the second argument and standard deviation by the third. More information about probability distributions can be found in section 8.9.

8.1.4 Normal Distributions

```
> xpnorm(1.96, 0, 1)
If X ~ N(0,1), then

        P(X <= 1.96) = P(Z <= 1.96) = 0.975
        P(X >  1.96) = P(Z >  1.96) = 0.025

[1] 0.975
```



8.2 One Categorical Variable

8.2.1 Numerical Summaries

The `xtabs()`, `perctable()` and `proptable()` functions can be used to calculate counts, percentages and proportions, respectively for a categorical variable.

```
> xtabs(~ homeless, HELP)      # display table using counts
homeless
homeless   housed
     209      244
> perctable(HELP$homeless)     # display table using percentages
homeless   housed
    46.1     53.9
> proptable(HELP$homeless)     # display table using proportions
homeless   housed
   0.461    0.539
```

8.2.2 The Binomial Test

An exact confidence interval for a proportion (as well as a test of the null hypothesis is equal to a particular value [by default 0.5]) can be calculated using the `binom.test()` function. The standard `binom.test()` requires us to tabulate.

```
> binom.test(209, 209 + 244)

        Exact binomial test

data:  209 and 209 + 244
number of successes = 209, number of trials = 453, p-value = 0.1101
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.415 0.509
sample estimates:
probability of success
                 0.461
```

The `mosaic` package provides a formula interface that avoids the need to pre-tally the data.

```
> result <- binom.test(~ homeless=="homeless", HELP)
> result

        Exact binomial test

data:  HELP$homeless == "homeless"
number of successes = 209, number of trials = 453, p-value = 0.1101
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.415 0.509
sample estimates:
probability of success
                 0.461

> names(result)
[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
[6] "null.value"  "alternative" "method"      "data.name"
```

As is generally the case with commands of this sort, there are a number of useful quantities available from the object returned by the function. For example, the user can extract the confidence interval either manually

```
> result$conf.int
[1] 0.415 0.509
attr(,"conf.level")
[1] 0.95
```

```
> result$conf.int[1]
[1] 0.415
```

You may want to read the footnote that describes what's going on here in a bit more detail.

or via the `interval()` function in the `mosaic` package.[1]

```
> interval(result)
Method: Exact binomial test

probability of success
                0.461

95% confidence interval:
0.415 0.509
```

We can extract the p-value similarly.

```
> pval(result)
Method: Exact binomial test

Null Hypothesis: probability of success = 0.5
Alt. Hypothesis: probability of success <> 0.5

number of successes = 209  (number of trials = 453)

p-value = 0.1101
```

8.2.3 The Proportion Test

A similar interval and test can be calculated using `prop.test()`. This takes a table (as generated by `xtabs()` as an argument).

```
> prop.test(xtabs(~ homeless=="housed", HELP), correct=FALSE)
        1-sample proportions test without continuity correction

data:  x, null probability 0.5
X-squared = 2.7, df = 1, p-value = 0.1001
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.416 0.507
sample estimates:
    p
0.461
```

`prop.test()` uses a Chi-squared statistic. Most introductory texts use a z-statistic. They are equivalent, but you may need to address this with your students.

It also accepts summarized data, the way `binom.test()` does.

```
> prop.test(209, 209 + 244, correct=FALSE)
        1-sample proportions test with continuity correction

data:  209 and 209 + 244
X-squared = 2.55, df = 1, p-value = 0.1102
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
```

[1] A word of explanation is in order here. In R what you see is not what you get. Most of the objects we encounter in R have a `print()` method. So when we get `result`, what we are seeing displayed is `print(result)`. There may be a good deal of additional information lurking inside the object itself. To make matter even more complicated, some objects are returned *invisibly*, so nothing prints. You can still assign the returned object to a variable and process it later, even if nothing shows up on the screen. This is the case for the `lattice` graphics functions, for example. You can save a plot into a variable, say `myplot` and display the plot again later using `print(myplot)`.

```
 0.415 0.509
sample estimates:
    p
0.461
```

or counts of successes and failures as a single vector.

```
> prop.test(c(209, 244), correct=FALSE)

        1-sample proportions test with continuity correction

data:  c(209, 244)
X-squared = 2.55, df = 1, p-value = 0.1102
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.415 0.509
sample estimates:
    p
0.461
```

To make things simpler still, we've added a formula interface in the `mosaic` package.

```
> prop.test(~ homeless, data=HELP)

        1-sample proportions test with continuity correction

data:  HELP$homeless
X-squared = 2.55, df = 1, p-value = 0.1102
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.415 0.509
sample estimates:
    p
0.461
```

8.2.4 Goodness of Fit Tests

A variety of goodness of fit tests can be calculated against a reference distribution. For the HELP data, we could test the null hypothesis that the proportion of subjects back in the original population in each of the substance abuse groups were equal.

```
> substab <- xtabs(~ substance, data=HELP)
> substab

substance
alcohol cocaine  heroin
    177     152     124

> perctable(HELP$substance)

alcohol cocaine  heroin
   39.1    33.6    27.4

> p <- c(1/3, 1/3, 1/3)   # equivalent to rep(1/3, 3)
> chisq.test(substab, p=p)

        Chi-squared test for given probabilities

data:  substab
X-squared = 9.31, df = 2, p-value = 0.009508

> total <- sum(substab); total

[1] 453
```

```
> total*p

[1] 151 151 151
```

We don't encourage much manual calculations in our courses

We can also calculate this quantity manually, in terms of observed and expected values.

```
> chisq <- sum((substab - total*p)^2/(total*p)); chisq

[1] 9.31

> 1 - pchisq(chisq, 2)

[1] 0.00951
```

x is for eXtra.

Alternatively, the `mosaic` package provides a version of `chisq.test()` with more verbose output.

```
> xchisq.test(substab, p=p)

        Chi-squared test for given probabilities

data:  substab
X-squared = 9.31, df = 2, p-value = 0.009508

 177.00   152.00   124.00
(151.00) (151.00) (151.00)
[4.4768] [0.0066] [4.8278]
< 2.116> < 0.081> <-2.197>

key:
        observed
        (expected)
        [contribution to X-squared]
        <residual>

> rm(substab, p, total, chisq)                    # disposing of variables no longer needed.
```

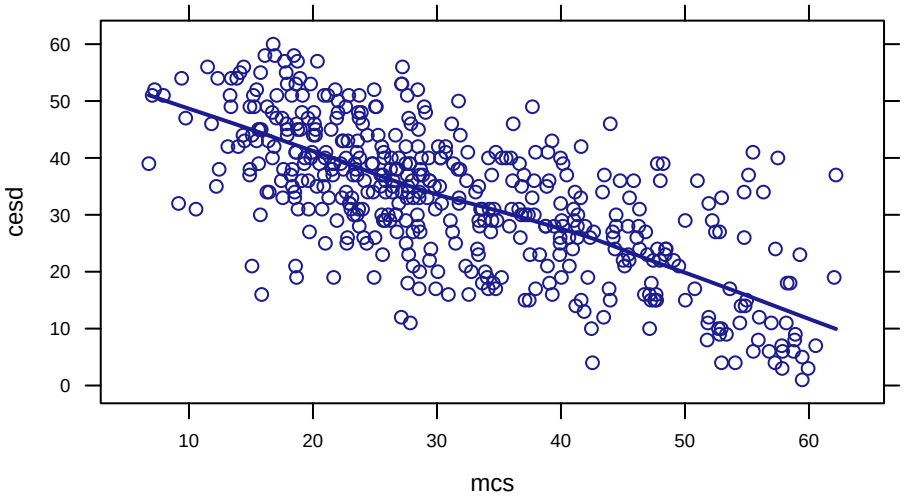
8.3 Two Quantitative Variables

8.3.1 Scatterplots

The lowess line can help to assess linearity of a relationship

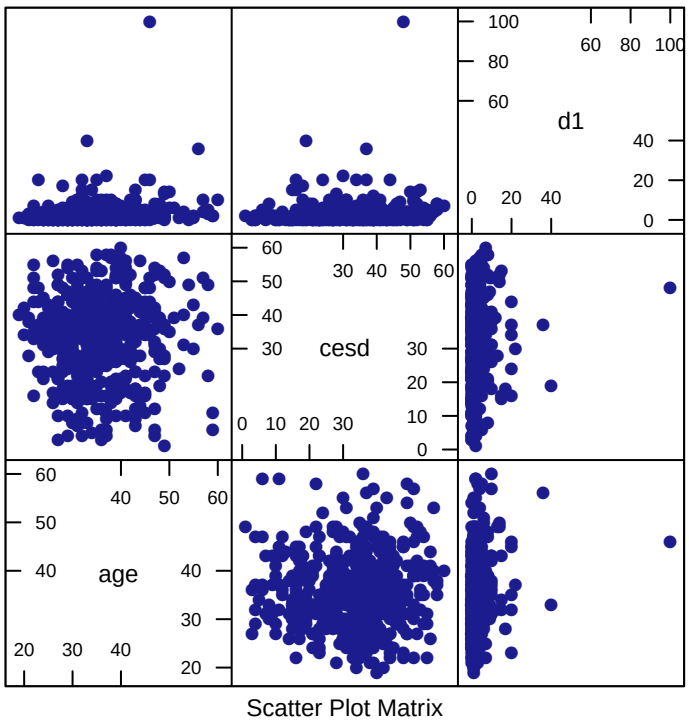
We always encourage students to start any analysis by graphing their data. Here we augment a scatterplot of the CESD (depressive symptoms) and the MCS (mental component score from the SF-36) with a lowess (locally weighted scatterplot smoother) line.

```
> xyplot(cesd ~ mcs, data=HELP, type=c('p','smooth'), pch=1)
```



For multivariate data sets, `splom()` will generate pairwise scatterplots for selected variables.

```
> splom(HELP[,c("age","cesd","d1")])
```



8.3.2 Correlation

Correlations can be calculated for a pair of variables, or for a matrix of variables.

```
> with(HELP, cor(cesd, mcs))
```

```
[1] -0.682
```

```
> with(HELP, cor(cbind(cesd, mcs, pcs)))
```

```
       cesd    mcs    pcs
cesd  1.000 -0.682 -0.293
mcs  -0.682  1.000  0.110
pcs  -0.293  0.110  1.000
```

By default, Pearson correlations are provided, other variants can be specified using the `method` option.

```
> with(HELP, cor(cesd, mcs), method="spearman")
```

```
[1] -0.682
```

As mentioned earlier, we tend to introduce linear regression early in our courses.

8.3.3 Simple Linear Regression

Linear regression models were introduced previously in section 6.3.1. These use the same formula interface to specify the outcome and predictors. Here we consider fitting the model `cesd` ~ `mcs`.

```
> model <- lm(cesd ~ mcs, data=HELP)
> coef(model)
```

```
(Intercept)         mcs
     53.902      -0.665
```

```
> summary(model)
```

```
...
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  53.9022     1.1472    47.0   <2e-16 ***
mcs          -0.6647     0.0336   -19.8   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.16 on 451 degrees of freedom
Multiple R-squared: 0.465,        Adjusted R-squared: 0.464
F-statistic:  392 on 1 and 451 DF,  p-value: <2e-16
```

```
> class(model)
```

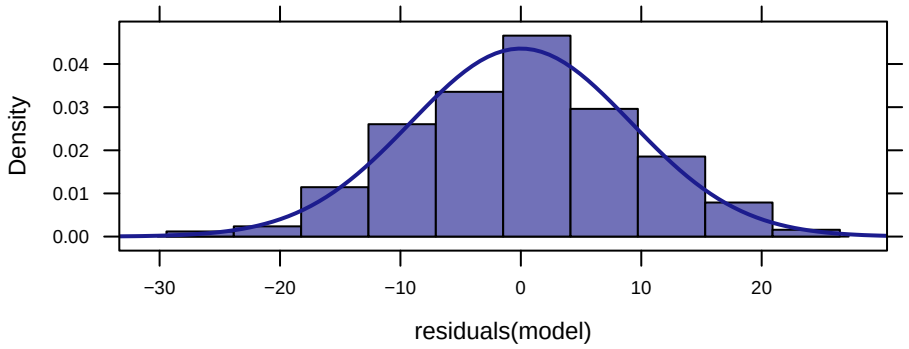
```
[1] "lm"
```

`residuals()` can be abbreviated `resid()`.

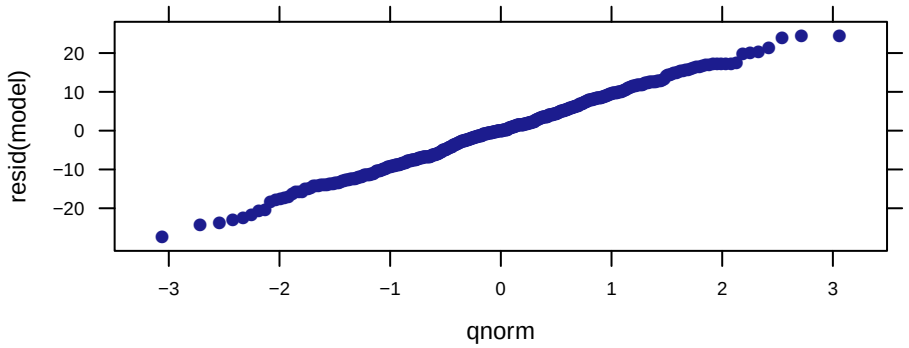
The return value from `lm()` is a linear model object; A number of functions can operate on these objects, as seen previously with `coef()`. As an additional example, there is a function called `residuals()` that will return a vector of the residuals.

Another useful function is `predict()`

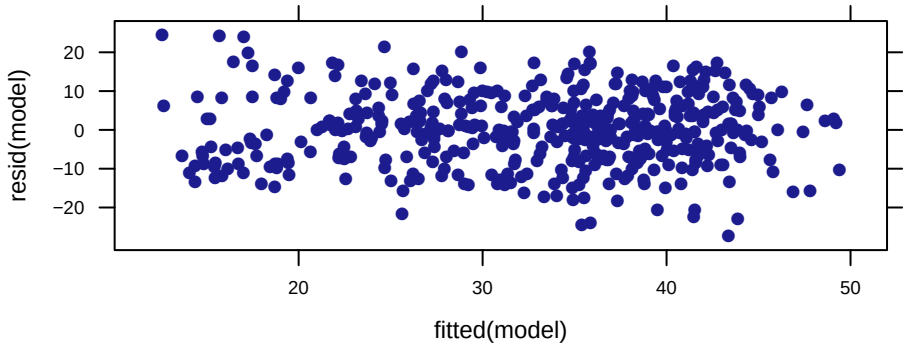
```
> xhistogram(~ residuals(model), density=TRUE)
```




```
> qqmath(~ resid(model))
```

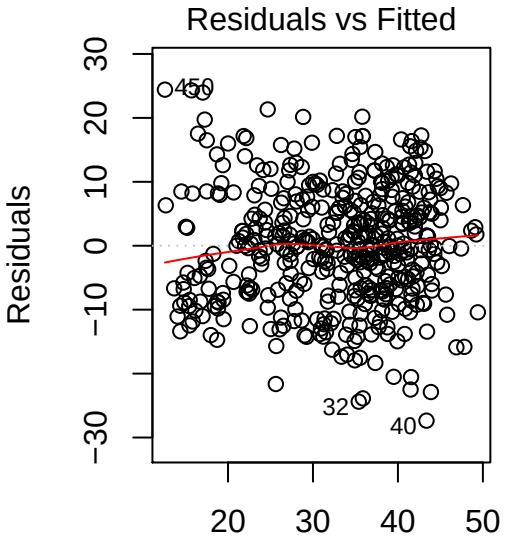



```
> xyplot(resid(model) ~ fitted(model))
```

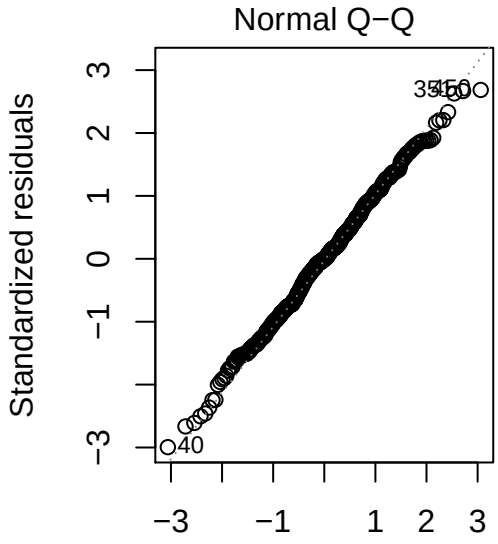


In addition, the `plot()` method can produce a number of diagnostic plots (see `?plot.lm`).

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




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



8.4 Two Categorical Variables

8.4.1 Cross classification Tables

Cross classification (two-way or R by C) tables can be constructed for two (or more) categorical variables. Here we consider the contingency table for homeless status (homeless one or more nights in the past 6 months or housed) and sex.

From Raw Data

```
> xtabs(~ homeless + sex, data=HELP)
```

```
          sex
homeless   female male
  homeless     40  169
  housed       67  177
```

Building Cross Tables from Already Summarized Data

It can be handy to create a table from existing summaries. Here we demonstrate how to generate this for our table of homeless status by sex.

```
> mytab <- enterxtab()
```

```
Row variable name:
1: sex
Read 1 item
Row variable levels:
1: male
2: female
3:                  # entered a blank line here
Read 2 items
Column variable name:
1: status
Read 1 item
Column variable levels:
1: homeless
2: housed
3:                  # and here
Read 2 items
Row 1:
1: 169 40
Read 2 items
Row 2:
1: 177 67
Read 2 items
```

```
> mytab
```

```
        status
sex      homeless housed
  male        169     40
  female      177     67
```

Epidemiologic statistics

To demonstrate how to extend R using your own functions, we can write one which calculates epidemiologic statistics such as the odds ratio (OR). More information about writing your own functions can be found in section B.7.

```
> odds.ratio <- function(x, y, ds) {
   tab = xtabs(~ x + y)
   return(tab[1,1]*tab[2,2]/(tab[1,2]*tab[2,1]))
 }
> with(HELP, odds.ratio(homeless, sex))
```

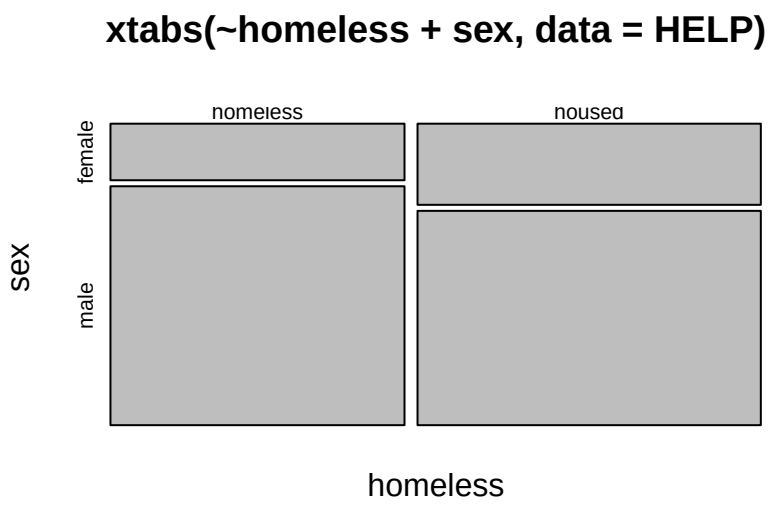
```
[1] 0.625
```

Graphical Summaries

Graphical summaries of cross classification tables may be helpful in visualizing associations. Mosaic plots are one example (though the jury is still out regarding its utility, relative to the low data to ink ratio [Tuf01]. Here we see that males tend to be over-represented amongst the homeless subjects (as represented by the horizontal line which is higher for the homeless rather than the housed).

The `mosaic()` function in the `vcd` package also makes mosaic plots.

```
> mosaicplot(xtabs(~ homeless + sex, data=HELP))
```



8.4.2 Chi-squared tests

```
> chisq.test(xtabs(~ homeless + sex, data=HELP), correct=FALSE)

        Pearson's Chi-squared test

data:  xtabs(~homeless + sex, data = HELP)
X-squared = 4.32, df = 1, p-value = 0.03767
```

There is a statistically significant association found: it is unlikely that we would observe an association this strong if there was homeless status and sex were independent back in the population.

Displaying additional information

When a student finds a significant association, it's important for them to be able to interpret this in the context of the problem. The `xchisq.test()` function provides additional details to help with this process.

```
> xchisq.test(xtabs(~homeless + sex, data=HELP), correct=FALSE)

        Pearson's Chi-squared test

data:  xtabs(~homeless + sex, data = HELP)
X-squared = 4.32, df = 1, p-value = 0.03767

  40.00   169.00
( 49.37) (159.63)
 [1.78]   [0.55]
<-1.33>  < 0.74>

  67.00   177.00
( 57.63) (186.37)
 [1.52]   [0.47]
< 1.23>  <-0.69>
```

```
key:
        observed
        (expected)
        [contribution to X-squared]
        <residual>
```

We observe that there are fewer homeless women, and more homeless men that would be expected.

8.4.3 Fisher's Exact Test

An exact test can also be calculated. This is fairly computationally straightforward for 2 by 2 tables. Options to help constrain the size of the problem for larger tables exist (see `?fisher.test()`).

```
> fisher.test(xtabs(~homeless + sex, data=HELP))

        Fisher's Exact Test for Count Data

data:  xtabs(~homeless + sex, data = HELP)
p-value = 0.0456
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.389 0.997
sample estimates:
odds ratio
     0.626
```

8.5 Quantitative Response to a Categorical Predictor

8.5.1 A Bivariate Predictor: Numerical and Graphical Summaries

Here we will compare the distributions of CESD scores by sex.

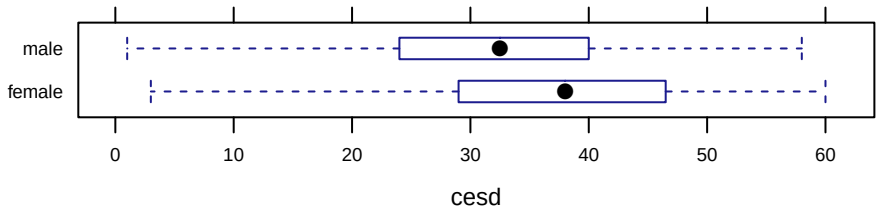
The `aggregate()` function can be used to calculate the mean CESD score for each of the two groups

```
> aggregate(cesd ~ sex, data=HELP, FUN=mean)

     sex cesd
1 female 36.9
2   male 31.6

> aggregate(cesd ~ sex, data=HELP, FUN=favstats)

     sex cesd
1 female    3
2   male    1
```

Boxplots are a particularly helpful graphical display to compare distributions. The `bwplot()` function can be used to display the boxplots for the CESD scores separately by sex. We see from both the numerical and graphical displays that women tend to have slightly higher CESD scores than men.

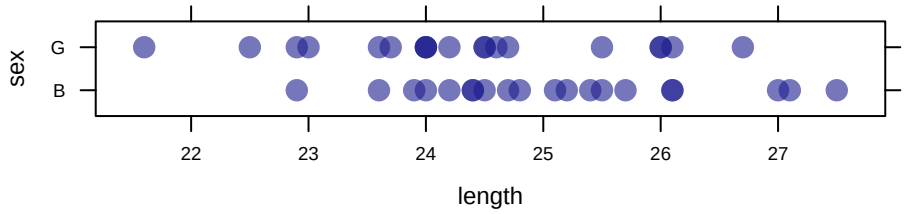
Although we usually put explanatory variables along the horizontal axis, page layout sometimes makes the other orientation preferable for these plots.

```
> bwplot(sex ~ cesd, data=HELP)
```



When sample sizes are small, there is no reason to summarize with a boxplot since `xyplot()` can handle categorical predictors. Even with 10–20 observations in a group, a scatter plot is often quite readable. Setting the alpha level helps detect multiple observations with the same value.

```
> xyplot(sex ~ length, KidsFeet, alpha=.6, cex=1.4)
```



One of us once saw a biologist proudly present side-by-side boxplots. Thinking a major victory had been won, he naively asked how many observations were in each group. "Four," replied the biologist.

8.5.2 A Bivariate Predictor: Two-sample t

The Student's two sample t-test can be run without or with an equal variance assumption.

```
> t.test(cesd ~ sex, data=HELP, var.equal=FALSE)

        Welch Two Sample t-test

data:  cesd by sex
t = 3.73, df = 167, p-value = 0.0002587
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 2.49 8.09
sample estimates:
mean in group female   mean in group male
                36.9                 31.6
```

We see that there is a statistically significant difference between the two groups.

8.5.3 Non-parametric 2 group tests

The same conclusion is seen using a non-parametric (Wilcoxon rank sum) test.

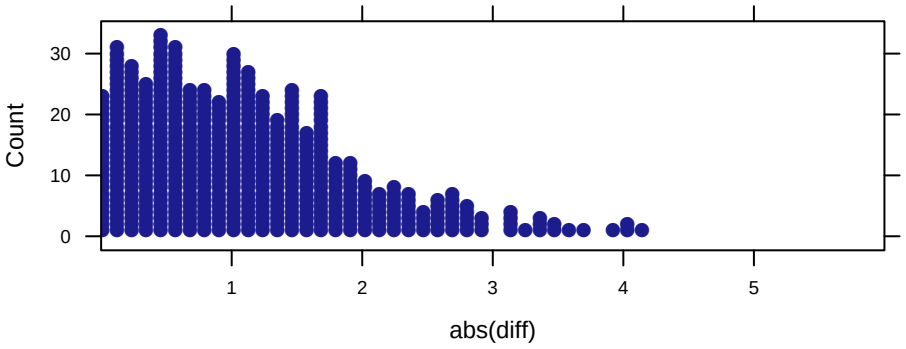
```
> wilcox.test(cesd ~ sex, data=HELP)

        Wilcoxon rank sum test with continuity correction
```

```
data:  cesd by sex
W = 23105, p-value = 0.0001033
alternative hypothesis: true location shift is not equal to 0
```

8.5.4 Permutation test

Here we extend the methods introduced in section 6.3.1 to undertake a two-sided test.

```
> simulations <- 500
> test.stat <-  abs(diff(aggregate(cesd ~ sex, data=HELP, mean)$cesd))
> test.stat
[1] 5.29
> rtest.stats <- do(simulations) * c(diff= diff(aggregate(cesd ~ shuffle(sex), HELP, mean)$cesd))
> dotPlot(~ abs(diff), rtest.stats, n=40, xlim=c(0, 6),
   groups=abs(rtest.stats$diff) >= test.stat, pch=16, cex=.8 )
> ladd(panel.abline(v=test.stat))
```

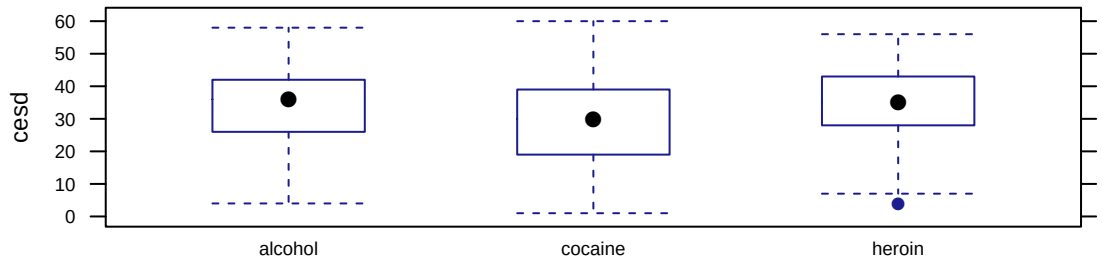


The same conclusion is observed with a permutation test (the observed value is nowhere in the vicinity of the permuted values).

8.5.5 One-way ANOVA

Earlier comparisons were between two groups: we can also consider testing differences in CESD scores by primary substance of abuse (heroin, cocaine, or alcohol).

```
> bwplot(cesd ~ substance, data=HELP)
```



```
> aggregate(cesd ~ substance, data=HELP, FUN=mean)
  substance cesd
1   alcohol 34.4
2   cocaine 29.4
3    heroin 34.9
> mod <- aov(cesd ~ substance, data=HELP)
> summary(mod)
             Df Sum Sq Mean Sq F value  Pr(>F)
substance     2   2704    1352    8.94 0.00016 ***
Residuals   450  68084     151
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

While still high (scores of 16 or more are generally considered to be "severe" symptoms), the cocaine-involved group tend to have lower scores than those whose primary substances are alcohol or heroin.

```
> mod1 <- lm(cesd ~ 1, data=HELP)
> mod2 <- lm(cesd ~ substance, data=HELP)
```

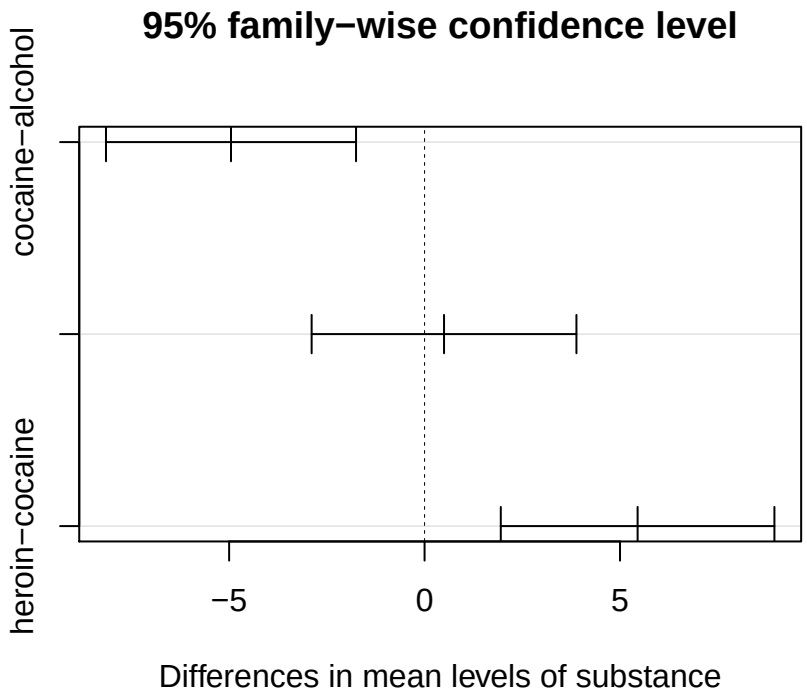
The anova() command can be used to formally compare two (nested) models.

```
> anova(mod1, mod2)
Analysis of Variance Table

Model 1: cesd ~ 1
Model 2: cesd ~ substance
  Res.Df   RSS Df Sum of Sq    F  Pr(>F)
1    452 70788
2    450 68084  2      2704 8.94 0.00016 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

8.5.6 Tukey's Honest Significant Differences

There are a variety of multiple comparison procedures that can be used after fitting an ANOVA model. One of these is Tukey's Honest Significant Difference (HSD). Other options are available with the multcomp package.

```
> with(HELP, tapply(cesd, substance, mean))
alcohol cocaine  heroin
   34.4    29.4    34.9
> compare <- TukeyHSD(mod, "substance")
> compare
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = cesd ~ substance, data = HELP)

$substance
                 diff   lwr   upr p adj
cocaine-alcohol -4.952 -8.15 -1.75 0.001
heroin-alcohol   0.498 -2.89  3.89 0.936
heroin-cocaine   5.450  1.95  8.95 0.001
> plot(compare)
```



Again, we see that the cocaine group has significantly lower CESD scores than the other two groups.

8.6 Categorical Response to a Quantitative Predictor

8.6.1 Logistic Regression

`glm()` has arguments `family` and `link`. The `logit` link is the default link for the binomial family, so we don't need to specify it here.

Logistic regression is available from within the `glm()` function: a variety of link functions and distributional forms for generalized linear models are supported.

```
> logitmod <- glm(homeless ~ age + female, family=binomial, data=HELP)
> summary(logitmod)

Call:
glm(formula = homeless ~ age + female, family = binomial, data = HELP)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.547  -1.202   0.918   1.123   1.360

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.8926     0.4537    1.97    0.049 *
age          -0.0239     0.0124   -1.92    0.055 .
female        0.4920     0.2282    2.16    0.031 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 625.28  on 452  degrees of freedom
```

```
Residual deviance: 617.19  on 450  degrees of freedom
AIC: 623.2

Number of Fisher Scoring iterations: 4
```

```
> exp(coef(logitmod))
```

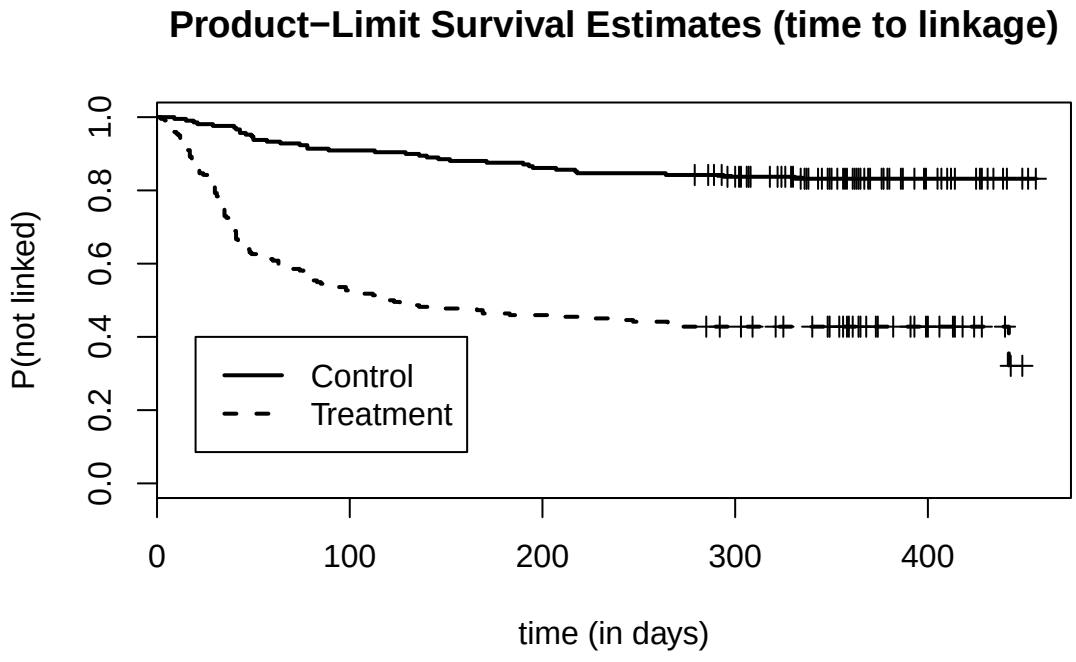
```
(Intercept)         age      female
      2.442       0.976       1.636
```

8.7 Survival Time Outcomes

Extensive support for Survival (Time to Event) analysis is available within the `survival` package.

8.7.1 Kaplan-Meier plot

```
> library(survival)
> fit <- survfit(Surv(dayslink, linkstatus) ~ treat, data=HELP)
> plot(fit, conf.int=FALSE, lty=1:2, lwd=2, xlab="time (in days)",
   ylab="P(not linked)")
> legend(20, 0.4, legend=c("Control", "Treatment"), lty=c(1,2),
   lwd=2)
> title("Product-Limit Survival Estimates (time to linkage)")
```



We see that the subjects in the treatment (HELP clinic) were significantly more likely to link to primary care than the control (usual care) group.

8.7.2 Cox proportional hazards model

```
> library(survival)
> summary(coxph(Surv(dayslink, linkstatus) ~ age + substance, data=HELP))
Call:
coxph(formula = Surv(dayslink, linkstatus) ~ age + substance,
    data = HELP)

  n= 431, number of events= 163
   (22 observations deleted due to missingness)

                     coef exp(coef) se(coef)     z Pr(>|z|)
age               0.00893   1.00897  0.01026  0.87     0.38
substancecocaine  0.18045   1.19775  0.18100  1.00     0.32
substanceheroin  -0.28970   0.74849  0.21725 -1.33     0.18

                 exp(coef) exp(-coef) lower .95 upper .95
age                  1.009      0.991     0.989      1.03
substancecocaine     1.198      0.835     0.840      1.71
substanceheroin      0.748      1.336     0.489      1.15

Rsquare= 0.014   (max possible= 0.988 )
Likelihood ratio test= 6.11  on 3 df,   p=0.106
Wald test            = 5.84  on 3 df,   p=0.12
Score (logrank) test = 5.91  on 3 df,   p=0.116
```

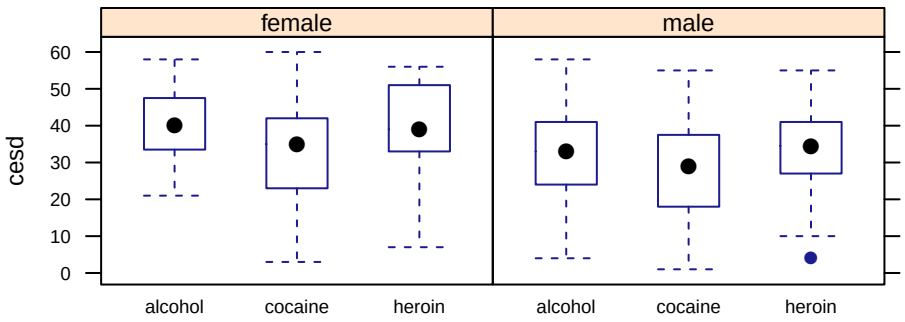
Neither age, nor substance group was significantly associated with linkage to primary care.

8.8 More than Two Variables

8.8.1 Two (or more) way ANOVA

We can fit a two (or more) way ANOVA model, without or with an interaction, using the same modeling syntax.

```
> bwplot(cesd ~ substance | sex, data=HELP)
```




```
> summary(aov(cesd ~ substance + sex, data=HELP))
             Df Sum Sq Mean Sq F value  Pr(>F)
substance     2   2704    1352    9.27 0.00011 ***
sex           1   2569    2569   17.61 3.3e-05 ***
```

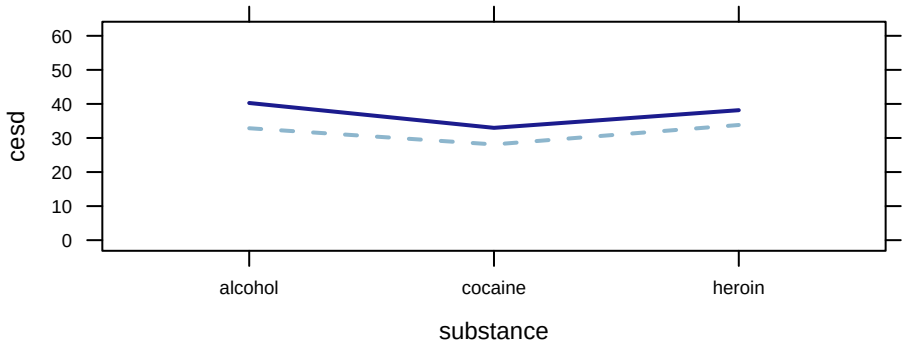
```
Residuals   449  65515     146
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
> summary(aov(cesd ~ substance * sex, data=HELP))
```

```
              Df Sum Sq Mean Sq F value  Pr(>F)
substance      2   2704    1352    9.25 0.00012 ***
sex            1   2569    2569   17.57 3.3e-05 ***
substance:sex  2    146      73    0.50 0.60752
Residuals    447  65369     146
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There's little evidence for the interaction, though there are statistically significant main effects terms for substance group and sex.

```
> xyplot(cesd ~ substance, data=HELP, groups=sex, type='a')
```



8.8.2 Multiple Regression

```
> lm1 <- lm(cesd ~ mcs + age + sex, data=HELP)
> summary(lm1)
```

```
...
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  53.8303     2.3617   22.79   <2e-16 ***
mcs          -0.6548     0.0336  -19.50   <2e-16 ***
age           0.0553     0.0556    1.00   0.3200
sexmale      -2.8993     1.0137   -2.86   0.0044 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.09 on 449 degrees of freedom
Multiple R-squared: 0.476,        Adjusted R-squared: 0.473
F-statistic:  136 on 3 and 449 DF,  p-value: <2e-16
```

```
> lm2 <- lm(cesd ~ mcs + age + sex + mcs*sex, data=HELP)
> summary(lm2)
```

```
...
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  55.3906     2.9903   18.52   <2e-16 ***
mcs          -0.7082     0.0712   -9.95   <2e-16 ***
age           0.0549     0.0556    0.99    0.324
sexmale      -4.9421     2.6055   -1.90    0.058 .
```



```
mcs:sexmale   0.0687     0.0807    0.85    0.395
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.09 on 448 degrees of freedom
Multiple R-squared: 0.477,        Adjusted R-squared: 0.472
F-statistic:  102 on 4 and 448 DF,  p-value: <2e-16
```

```
> anova(lm1, lm2)
Analysis of Variance Table

Model 1: cesd ~ mcs + age + sex
Model 2: cesd ~ mcs + age + sex + mcs * sex
  Res.Df   RSS Df Sum of Sq    F Pr(>F)
1    449 37088
2    448 37028  1      59.9 0.72    0.4
```

There is little evidence for an interaction effect: we may want to drop this from the model.

8.9 Probability and Random Variables

R can calculate quantities related to probability distributions of all types. It is straightforward to generate random variables from these distributions, which can be used for simulation and analysis.

Table 8.1 displays the basenames for probability distributions available within base R. These functions can be prefixed by `d` to find the density function for the distribution, `p` to find the cumulative density function, `q` to find quantiles, and `r` to generate random draws. For example, to find the density function of a binomial random variable, use the command `dbinom()`. The `qDIST()` function is the inverse of the `pDIST()` function, for a given basename `DIST`.

```
> pnorm(1.96, 0, 1)    # P(Z < 1.96)
[1] 0.975
> qnorm(.975, 0, 1)
[1] 1.96
> integrate(dnorm, -Inf, 0)
0.5 with absolute error < 4.7e-05
```

8.10 Power Calculations

While not generally a major topic in introductory courses, power and sample size calculations help to reinforce key ideas in statistics. In this section, we will explore how R can be used to undertake power calculations using analytic approaches (see 6.5 for simulation based approaches). We consider a simple problem with two tests (t-test and sign test) of a one-sided comparison.

Let $X_1, ..., X_{25}$ be i.i.d. $N(0.3, 1)$. Consider testing the null hypothesis $H_0 : \mu = 0$ versus $H_A : \mu > 0$ at significance level $\alpha = .05$. Compare the power of the sign test and the power of the test based on normal theory (one sample one sided t-test) assuming that σ is known.

8.10.1 Sign test

We first start by calculating the Type I error rate for the sign test. Here we want to reject when the number of positive values is large. Under the null hypothesis, this is distributed as a Binomial

Table 8.1: basename for probability distribution and random number generation

Distribution	NAME
Beta	`beta`
binomial	`binom`
Cauchy	`cauchy`
chi-square	`chisq`
exponential	`exp`
F	`f`
gamma	`gamma`
geometric	`geom`
hypergeometric	`hyper`
logistic	`logis`
lognormal	`lnorm`
negative binomial	`nbinom`
normal	`norm`
Poisson	`pois`
Student's t	`t`
Uniform	`unif`
Weibull	`weibull`

random variable with n=25 trials and p=0.5 probability of being a positive value. Let's consider values between 15 and 19.

```
> xvals <- 15:19
> probs <- 1 - pbinom(xvals, 25, 0.5)
> cbind(xvals, probs)
```

```
     xvals   probs
[1,]    15 0.11476
[2,]    16 0.05388
[3,]    17 0.02164
[4,]    18 0.00732
[5,]    19 0.00204
```

So we see that if we decide to reject when $X =$ the number of positive values is 17 or larger, we will have an α level of 0.054, which is near the nominal value in the problem.

We calculate the power of the sign test as follows. The probability that $X > 0$, given that H_A is true is given by:

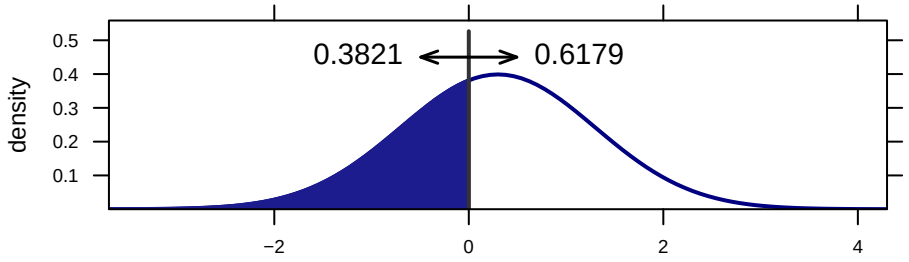
```
> 1-pnorm(0, 0.3, 1)
```

```
[1] 0.618
```

We can view this graphically using the command:

```
> xpnorm(0, 0.3, 1)
```

```
If X ~ N(0.3,1), then

       P(X <= 0) = P(Z <= -0.3) = 0.3821
       P(X >  0) = P(Z >  -0.3) = 0.6179

[1] 0.382
```

So now we need to find the power under the alternative, which is equal to the probability of getting 17 or more positive values, given that $p = 0.6179$:

```
> 1 - pbinom(16, 25, 0.6179)
```

```
[1] 0.338
```

The power is modest at best.

8.10.2 T-test

We next calculate the power of the test based on normal theory. To keep the comparison fair, we will set our α level equal to 0.05388. First we need to find the rejection region.

```
> alpha <- 1-pbinom(16, 25, .5) # == 0.0539, our alpha level.
> n <- 25; sigma <- 1 # given
> stderr <- sqrt(sigma^2/n)
> zstar <- qnorm(1-alpha, 0, 1)
> zstar
```

```
[1] 1.61
```

```
> crit <- zstar*stderr
> crit
```

```
[1] 0.322
```

Therefore, we reject for observed means greater than 0.322.

To calculate the power of this one-sided test we find the area under the alternative hypothesis that is to the right of this cutoff:

```
> power <- 1 - pnorm(crit, 0.3, stderr)
> power
```

```
[1] 0.457
```

Thus, the power of the test based on normal theory is 0.457. To provide a general check (or for future calculations of this sort) we can use the `power.t.test()` in R.

```
> power.t.test(n=25, delta=.3, sd=1, sig.level=alpha, alternative="one.sided",
 type="one.sample")$power
```

```
[1] 0.441
```

This yields a similar estimate to the value that we calculated directly. Overall, we see that the t-test has higher power than the sign test, if the underlying data are truly normal. It's useful to have students calculate power empirically, as demonstrated for this example in 6.5.

8.11 Exercises and Problems

8.1. Using the HELP dataset, create side by side boxplots of the CESD scores by substance abuse group, just for the male subjects.

8.2. Using the HELP dataset, fit a simple linear regression model predicting the number of drinks per day as a function of the mental component score.

This model can be specified using the formula: `i1` $\sim$ `mcs`.

Assess the distribution of the residuals for this model.

8.3. Generate a sample of 1000 exponential random variables with rate parameter equal to 2, and calculate the mean of those variables.

8.4. Find the median of the random variable X, if it is exponentially distributed with rate parameter 10.

8.5. Find the power of a two-sided two-sample t-test where both distributions are approximately normally distributed with the same standard deviation, but the group differ by 50% of the standard deviation. Assume that there are 100 observations per group and an alpha level of 0.01.

8.6. Repeat problem 8.5 by simulating using the methods of section 6.5.

8.7. Find the sample size needed to have 90% power for a two group t-test where the true difference between means is 25% of the standard deviation in the groups (with $\alpha = 0.05$).

9 More Examples: Favorite Data Sets and What to Do With Them

9.1 Health Evaluation and Linkage to Primary Care (HELP) Study

The HELP study was a clinical trial for adult inpatients recruited from a detoxification unit. Patients with no primary care physician were randomized to receive a multidisciplinary assessment and a brief motivational intervention or usual care, with the goal of linking them to primary medical care. Funding for the HELP study was provided by the National Institute on Alcohol Abuse and Alcoholism (R01-AA10870, Samet PI) and National Institute on Drug Abuse (R01-DA10019, Samet PI).

Eligible subjects were adults, who spoke Spanish or English, reported alcohol, heroin or cocaine as their first or second drug of choice, resided in proximity to the primary care clinic to which they would be referred or were homeless. Patients with established primary care relationships they planned to continue, significant dementia, specific plans to leave the Boston area that would prevent research participation, failure to provide contact information for tracking purposes, or pregnancy were excluded.

Subjects were interviewed at baseline during their detoxification stay and follow-up interviews were undertaken every 6 months for 2 years. A variety of continuous, count, discrete, and survival time predictors and outcomes were collected at each of these five occasions. The details of the randomized trial along with the results from a series of additional analyses have been published [SLH+03, RSHS01, HSLS02, LSS+02, KHF+03, SLH+04, SHL+05, SLH+05, LSH+06, WSH+07]. The Institutional Review Board of Boston University Medical Center approved all aspects of the study, including the creation of the de-identified dataset. Additional privacy protection was secured by the issuance of a Certificate of Confidentiality by the Department of Health and Human Services. A copy of the study instruments can be found at: `http://www.math.smith.edu/help`

The `mosaic` package contains several forms of the de-identified HELP dataset. We will focus on `HELP`, which contains 27 variables for the 453 subjects with minimal missing data, primarily at baseline. Variables included in the HELP dataset are described in Table 9.1. More information can be found in [HK11].

Table 9.1: Annotated description of variables in the HELP dataset

VARIABLE	DESCRIPTION (VALUES)	NOTE
`age`	age at baseline (in years) (range 19–60)	
`anysub`	use of any substance post-detox	see also `daysanysub`
`cesd`	Center for Epidemiologic Studies Depression scale (range 0–60)	

d1	how many times hospitalized for medical problems (lifetime) (range 0–100)	
daysanysub	time (in days) to first use of any substance post-detox (range 0–268)	see also anysubstatus
dayslink	time (in days) to linkage to primary care (range 0–456)	see also linkstatus
drugrisk	Risk-Assessment Battery (RAB) drug risk score (range 0–21)	see also sexrisk
e2b	number of times in past 6 months entered a detox program (range 1–21)	
female	gender of respondent (0=male, 1=female)	
g1b	experienced serious thoughts of suicide (last 30 days, values 0=no, 1=yes)	
homeless	1 or more nights on the street or shelter in past 6 months (0=no, 1=yes)	
i1	average number of drinks (standard units) consumed per day (in the past 30 days, range 0–142)	see also i2
i2	maximum number of drinks (standard units) consumed per day (in the past 30 days range 0–184)	see also i1
id	random subject identifier (range 1–470)	
indtot	Inventory of Drug Use Consequences (InDUC) total score (range 4–45)	
linkstatus	post-detox linkage to primary care (0=no, 1=yes)	see also dayslink
mcs	SF-36 Mental Component Score (range 7-62)	see also pcs
pcrec	number of primary care visits in past 6 months (range 0–2)	see also linkstatus, not observed at baseline
pcs	SF-36 Physical Component Score (range 14-75)	see also mcs
pss_fr	perceived social supports (friends, range 0–14)	
satreat	any BSAS substance abuse treatment at baseline (0=no, 1=yes)	
sex	sex of respondent (male or female)	
sexrisk	Risk-Assessment Battery (RAB) sex risk score (range 0–21)	see also drugrisk
substance	primary substance of abuse (alcohol, cocaine or heroin)	
treat	randomization group (randomize to HELP clinic, no or yes)	

Notes: Observed range is provided (at baseline) for continuous variables.

10

Teaching Calculus (and Beyond) in R

Perhaps you are thinking, "Why a chapter about teaching calculus?" Statisticians do one thing, mathematicians do another. Indeed, the large majority of introductory statistics classes, including the AP statistics class, is taught without any mention of calculus-level mathematics. As if to return the favor, a student taking almost any present-day calculus course will find little or no reason to think of a relationship to statistics.

When it comes to software, hardly any calculus-level mathematics course is taught with R. According to the Mathematical Association of America (MAA), in 2005, only about 50% of university-level introductory calculus was taught using any computer technology at all, and most of that is graphing calculators. When computers are used, the software is generally Mathematica or Maple or Sage (which, like R, is free) or sometimes MATLAB.

Indeed, if you want to teach calculus the way it is commonly taught, there seems to be little reason to consider R.

But do you really want to teach calculus the way it is commonly taught? What do you hope to accomplish by teaching calculus and are you successfully accomplishing it using the present techniques? A recent survey by the MAA [**?**] of students in introductory calculus in colleges and universities in the US indicates that at the start of the semester, 58% of students expected to get an A and 94% a B or higher. At the end of the semester, 50% had gotten a C or lower or withdrawn from the course. The majority of students taking introductory calculus have career interests outside of the traditional calculus clients: engineering, the physical sciences, mathematics itself. Fully 30% of students surveyed had their main interest in the medical/biological sciences.

Now think about the people you know outside of mathematics or physics or engineering. How many biology faculty have more than the vaguest reminiscence of the topics of calculus? They may remember it, positively or negatively, but they don't use it.

What scientists across the board do use is statistics. This is true in almost every field: biological, physical, and social sciences. And it turns out that calculus does have centrally important uses in even introductory statistics. Or, rather, if introductory statistics students know some calculus — certain essential parts of calculus, not the large majority of topics featured in almost all of today's courses — they can develop a stronger understanding of statistics and a better ability to use it in practice.

It's likely that the large majority of people reading this book have taken calculus at some point and so have a firm idea of what calculus is about. Put aside for the moment what you happened to study in your course and think instead of what students might actually need to work effectively in a world of complicated systems and massive amounts of data.

What students need:

- The idea of a function, especially functions of two or more variables.
- An ability to visualize functional relationships graphically.
- A modeling strategy: how to approximate potentially complex relationships with simple-to-understand ones.
- Derivatives, especially partial derivatives, to understand relationships and partial relationships.

What they don't need:

- Limits and formal definitions of continuity.
- Most symbolic algorithms, e.g. elaborate applications of the chain rule in differentiation or almost any symbolic integration.
- A course that leads them to drop the study of mathematics before they see the things that they do need.

As evidence that this is not an eccentric view, we refer you to the MAA report on "Curriculum Renewal across the First Two Years," which examined the relationship between mathematics and more than 20 "partner" disciplines: ranging from physics to engineering to business to biology.[**?**]

10.1 Calculus

Most people familiar with R assume that it provides no facilities for the central operations of calculus: differentiation and integration. Nonsense. What those people are thinking of is the way they learned calculus, where the technology for taking integrals and derivatives was based on rules for moving symbols, e.g., $\frac{d}{dx}x^n = nx^{n-1}$ or $\int \cos(ax)dx = \frac{1}{a}\sin(ax)$.

Those rules, and the algebraic notation that underlies them, seem to be so fundamental to people's conception of calculus that they forget that differentiation and integration are general operations that relate functions to one another, not rules for moving symbols.

To illustrate, consider the following function, which we will call `decay`

```
> k = 0.1
> P = 4
> decay = function(t){ exp(-k*t)*sin(2*pi*t/P) }
```

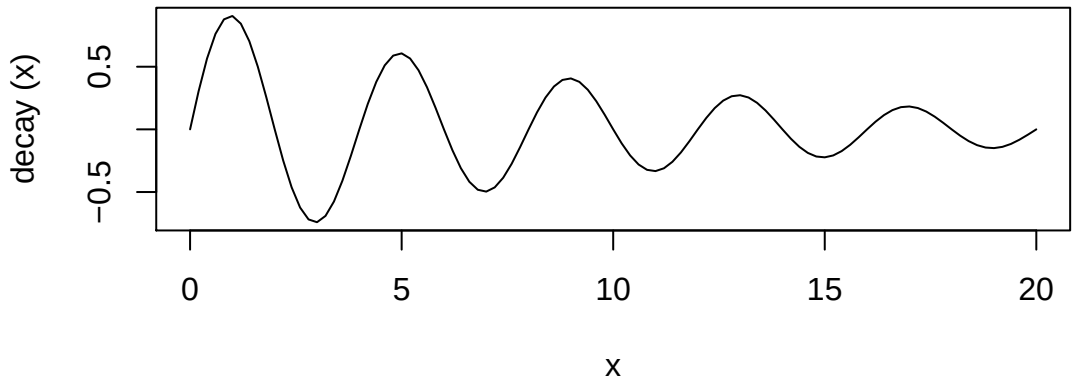
Traditional notation is not so different: $e^{-kt}\sin(\frac{2\pi}{P}t)$.

With R of course, you can evaluate the function at a given input or set of inputs:

```
> decay(5)
[1] 0.607
> decay(1:5)
[1]  9.05e-01  1.00e-16 -7.41e-01 -1.64e-16  6.07e-01
```

and you can plot it out over a in interval for the input:

```
> curve( decay, 0, 20)
```



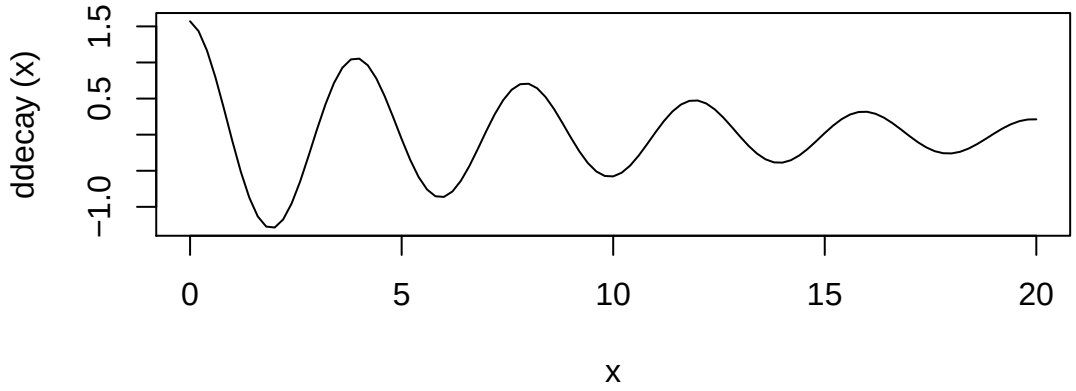
Suggestion Box
We have a dream... of a better way to plot functions. But we're not quite there yet. Suggestions (and offers to help) appreciated.

To find the derivative of this function, use the `D()` operator:

```
> ddecay = D(decay)
```

Again, you can evaluate and plot such functions:

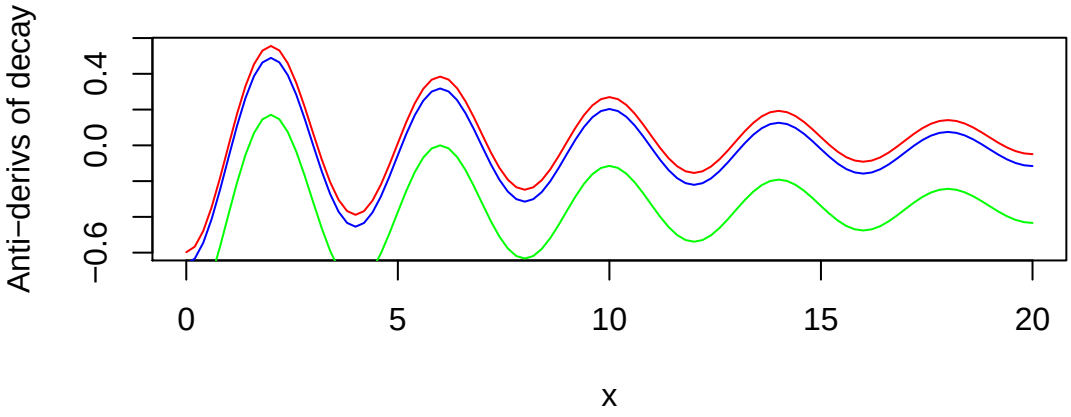
```
> ddecay(4)
[1] 1.05
> curve( ddecay, 0, 20)
```



Integration is a little tricker, because there is a "constant of integration" to be considers. The `mosaic` `integral()` operator let's you compute either a definite or indefinite integral.

Here's an indefinite integral:

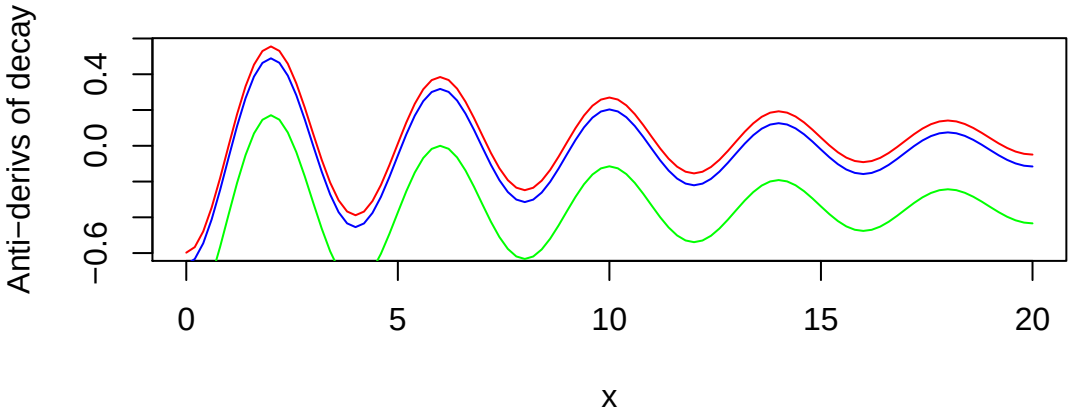
```
> f = antiD(decay)
> curve(f, 0, 20)
```



Of course, there's not just a single indefinite integral. There is the "constant of integration." You can show this directly to students:

```
> f1 = antiD(decay, 1)
> f2 = antiD(decay, 3)
> f3 = antiD(decay, 6)
```

```
> curve(f1, 0, 20, col="red", ylab="Anti-derivs of decay")
> curve(f2, add=TRUE, col="blue")
> curve(f3, add=TRUE, col="green")
```



Each of `f1`, `f2`, etc. are functions, and like any other functions they have a derivative.

```
> df1 = D(f1)
> df2 = D(f2)
> df3 = D(f3)
```

You can plot out these three functions and confirm that they are all the same, as you would expect: the derivative operation undoes the anti-derivative operation.

To calculate a definite integral, you need to specify the boundaries of integration. Currently, the

syntax is unnatural.

```
> antiD(decay, 140)(150)
```

```
[1] 7.21e-07
```

There are different styles for dealing with parameters.

This constructs a particular antiderivative integrating from a lower bound of 140, and that evaluates that antiderivative at an upper bound of 150.

In this example with `decay`, we are using a style of defining the parameters *outside* of the function itself. That makes it easy to explore different parameter values; just change the assigned values of `P` and `k`.

Another style is to create a function that itself constructs a function:

```
> makeDecay = function(P=3,k=.8){
   function(t){ exp(-k*t)*sin(2*pi*t/P) }
 }
```

It's still unclear what are the relative merits of these different styles when it comes to teaching. As the `mosaic` package develops, we'll try to support an effective style for handling parameters.

What might be troubling to those experienced with traditional ways of teaching integration and differentiation is that no formula is being given. But the resulting function is still a function, and you can do with it what you need: evaluate it or plot it over an interval, differentiate it, integrate it, find where it takes on a specified value.

Keep in mind that not all important functions have formulas that are readily interpretable. Indeed, students learn a set of names for such functions and study their properties, rather than learning how to identify them from a formula. The trigonometric functions, sin, cos, etc. are examples.

Many important functions do not have integrals that can be expressed in terms of a simple formula. For instance, the normal distribution

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(x - mu)^2}{2\sigma^2}\right).$$

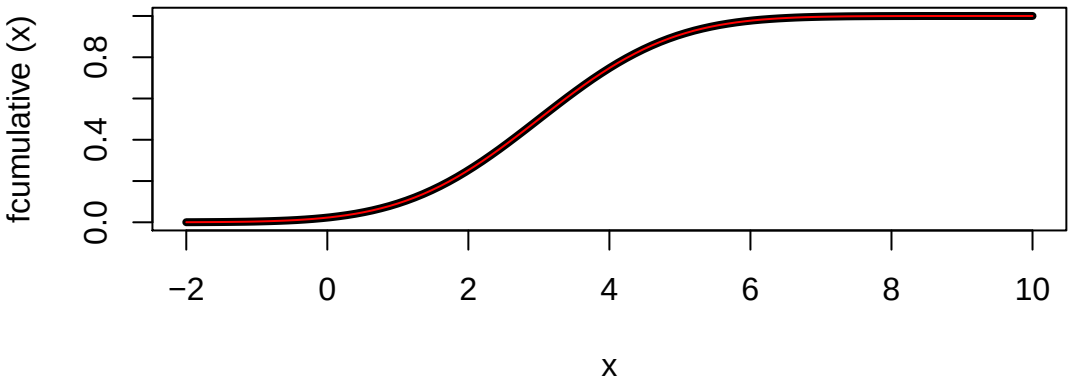
We could, of course, write this out as the equivalent formula in R, but the function is so important it already has a name in R: `dnorm()`.

Let's integrate it, setting $\mu = 3$ and $\sigma = 1.5$:

```
> f = function(x){dnorm(x, mean=3, sd=1.5) }
```

This f is a particular member of the family of normal distributions. Here is its cumulative function:

```
> fcumulative = antiD(f,-Inf)
> curve(fcumulative, -2,10, lwd=4) # by integration
> curve( pnorm(x, mean=3, sd=1.5), add=TRUE, col="red") # the built-in
```



There's little point in computing this integral, however, except to show that it matches the built-in `pnorm()` function.

One of the advantages to teaching integration and differentiation in a way that doesn't depend on constructing formulas, is that you can use functions that don't have simple formulas for modeling. For example, you can use functions generated by splining through data points.

10.2 Linear Algebra

Showing how to introduce Linear Algebra by taking linear combinations of functions to fit a set of data.

10.3 Differential Equations

Showing dynamics in one variable and two, e.g., the phase plane.

Handouts

This appendix contains materials that can be handed out as part of class activities (or repurposed for other situations).

A.1 What percentage of Earth is Covered with Water?

You have probably heard that the Earth is 2/3 water, or 70% water, or some other proportion water. How do people "know" this sort of stuff? The answer is they don't "know" it, but someone has estimated it. Using the power of GoogleMaps, we can make our own estimate and see if it is compatible with your folk knowledge.

We can estimate the proportion of the world covered with water by randomly sampling points on the globe and inspecting them using GoogleMaps.

1. First, let's do a sample size computation. If everyone in the class inspects 10 locations, what will our margin of error be?

2. Suppose we want to estimate (at the usual 95% confidence level) this proportion with ±5%. How large would our sample need to be?

3. Now fill in the following table.

margin of error	sample size
±5%	
±4%	
±2%	
±1%	

4. Generate enough random locations so that when we combine our data, we will get the precision we desire. To do this, replace the number 10 with the appropriate value.

```
> positions <- rgeo(10); positions
```

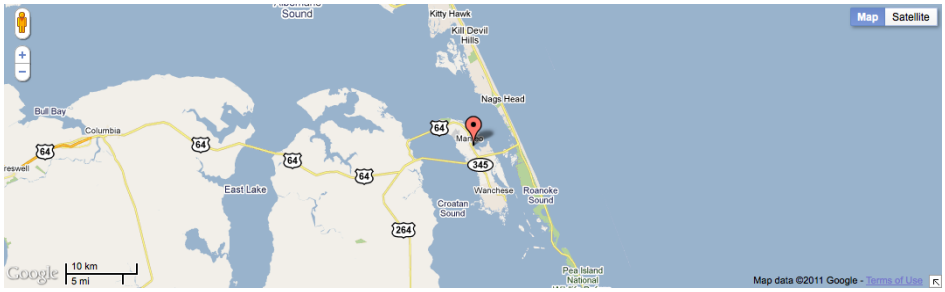
```
       lat    lon
1   16.253   76.2
2  -10.223   76.2
3   23.650  174.8
4  -74.632   69.6
5  -54.131  138.9
6   18.390   32.1
7   -0.543 -150.7
8   60.953  -41.5
9    3.883  -51.7
10   0.207   50.8
```

5. Open a GoogleMap centered at each position.

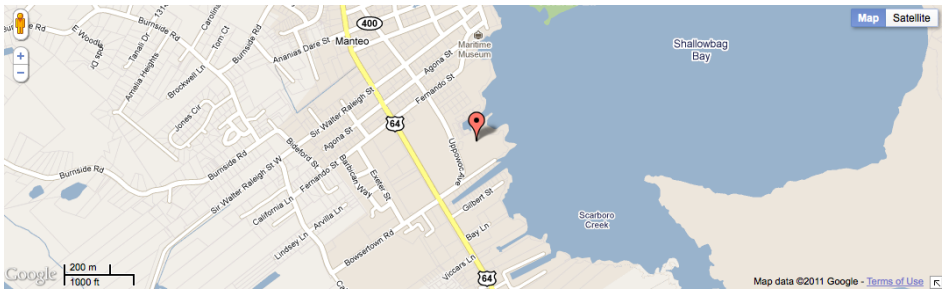
   Turn off pop-up blocking for your default browser, and then enter

```
> googleMap(pos=positions, mark=TRUE)
```

6. For each map, record whether the center is located in water or on land. `mark=TRUE` is used to place a marker at the center of the map which is helpful for locations that are close to the coast.

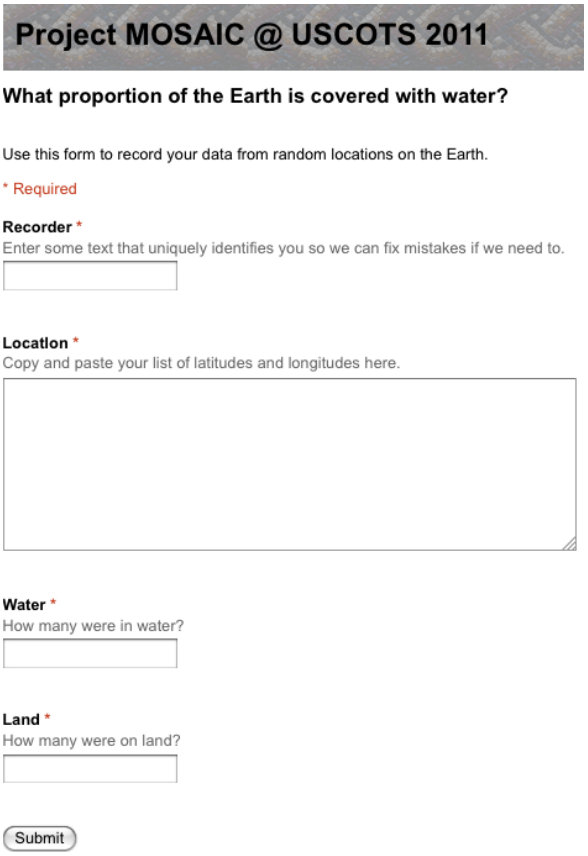


   You can zoom in or out to get a better look.



7. Record your data in a GoogleForm at

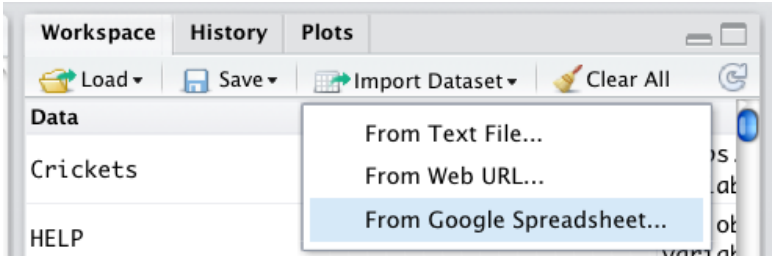
   `http://mosaic-web.org/uscots2011/google-water.html`



For the latitude and longitude information, simply copy and paste the output of

```
> positions
```

8. After everyone has entered their data into the GoogleForm, grab the data from Google:



9. Now it is simple to sum the counts across the class. Here's how it turned out in a different class:

```
> sum(Water$Water)
[1] 215
> sum(Water$Land)
[1] 85
```

10. Using our new data and your favorite analysis method (perhaps `binom.test()`) give a 95% confidence interval for the proportion of the world covered with water.

More About R

This material is more advanced than students in Intro Stats need, but is good for more advanced students and instructors to know.

B.1 Installing and Using Packages

R is open source software. Its development is supported by a team of core developers and a large community of users. One way that users support R is by providing **packages** that contain data and functions for a wide variety of tasks.

B.1.1 Installing packages from CRAN

If you need to install a package, most likely it will be on CRAN. Before a package can be used, it must be **installed** (once per computer) and **loaded** (once per R session). For example, to use `mosaic`:

```
> install.packages("mosaic")   # fetch package from CRAN to local machine.
> require(mosaic)              # load the package so it can be used.
```

If you are running on a machine where you don't have privileges to write to the default library location, you can install a personal copy of a package. If the location of your personal library is first in `R_LIBS`, this will probably happen automatically. If not, you can specify the location manually:

```
> install.packages("mosaic", lib="~/R/library")
```

On a networked machine, be sure to use a different local directory for each platform since packages must match the platform.

Installing packages on a Mac or PC is something you might like to do from the GUI since it will provide you with a list of packages from which you can select the ones of interest. Binary packages have been precompiled for a particular platform and are generally faster and easier to set up, if they are available. Source packages need to be compiled and built on your local machine. Usually this happens automatically – provided you have all the necessary tools installed on your machine – so the only disadvantage is the extra time it takes to do the compiling and building.

B.1.2 Installing other packages

Occasionally you might find a package of interest that is not available via a repository like CRAN. Typically, if you find such a package, you will also find instructions on how to install it. If not, you can usually install directly from the zipped up package file.

```
> install.packages('some-package.tar.gz',
                   repos=NULL)          # use a file, not a repository
```

B.1.3 Finding packages

There are several ways to find packages

- Ask your friends.
- Google: Put 'cran' in the search.
- Rseek: `http://rseek.org` provides a search engine specifically designed to find information about R.
- CRAN task views.

  A number of folks have put together task views that annotate a large number of packages and summarize they are good for. They are organized according to themes. Here are a few examples of available task views:

Bayesian	Bayesian Inference
Econometrics	Computational Econometrics
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays, Dynamic Graphics, Graphic Devices, and Visualization
Multivariate	Multivariate Statistics
SocialSciences	Statistics for the Social Sciences

- Bioconductor (`http://www.bioconductor.org/`) and Omegahat (`http://www.omegahat.org/R`) are another sources of packages (specify the `repos=` option to use these).
- *R Journal* (formerly *R News*) is available via CRAN and often has articles about new packages and their capabilities.
- Write your own.

  You can write your own packages, and it isn't that hard to do (but we won't cover this here).

B.2 Some Workflow Suggestions

In short: *Think like a programmer.*

- Use R interactively only to get documentation and for quick one-offs.
- Store your code in a file.

  You can execute all the code in a file using

```
> source("file.R")
```

RStudio has options for executing some or all lines in a file, too. See the buttons in the panel for any R script. (You can create a new R script file in the main file menu.)

If you work at the interactive prompt in the console and later wish you had been putting your commands into a file, you can save your past commands with

```
> savehistory("someRCommandsIalmostLost.R")
```

You can selectively save portions of your history to a script file using the History panel in RStudio.

Then you can go back and edit the file.

RStudio makes this especially easy by providing a integrated environment for working with R script files, an active R session, the R session history, etc. Many of these tasks can be done with the click of a button in RStudio.

- Use meaningful names.
- Write reusable functions.

  Learning to write your own functions will greatly increase your efficiency. (Stay tuned for details.)
- Comment your code.

  It's amazing what you can forget. The comment character in R is `#`.

B.3 Working with Data

B.3.1 Data in R packages

Data sets in the `datasets` package or any other loaded package are available via the `data()` function. Usually, the use of `data()` is unnecessary, however, since R will search most loaded packages (they must have been created with the lazy-load option) for data sets without the explicit use of `data()`. The `data()` function can be used to restore data after it has been modified or to control which package is used when data sets with the same name appear in multiple packages.

B.3.2 Loading data from flat files

R can read data from a number of file formats. The two most useful formats are `.csv` (comma separated values) and white space delimited. Excel and most statistical packages can read and write data in these formats, so these formats make it easy to transfer data between different software. R provides `read.csv()` and `read.table()` to handle these two situations. They work nearly identically except for their default settings: `read.csv()` assumes that the first line of the file contains the variable names but `read.table()` assumes that the data begins on the first line with no names for the variables, and `read.table()` will ignore lines that begin with '`#`' but `read.csv()` will not.

The default behavior can be overridden for each function, and there are a number of options that make it possible to read other file formats, to omit a specified number of lines at the top of the file, etc. If you are making the file yourself, always include meaningful names in either file format.

It is also possible to read data from a file located on the Internet. Simply replace the file name with a URL. The data read below come from [Tuf01].

```
> # need header=TRUE because there is a header line.
> # could also use read.file() without header=TRUE
> traffic <-
    read.table("http://www.calvin.edu/~rpruim/fastR/trafficTufte.txt",
    header=TRUE)
> traffic
```

```
  year cn.deaths   ny   cn   ma   ri
1 1951       265 13.9 13.0 10.2  8.0
2 1952       230 13.8 10.8 10.0  8.5
3 1953       275 14.4 12.8 11.0  8.5
4 1954       240 13.0 10.8 10.5  7.5
5 1955       325 13.5 14.0 11.8 10.0
6 1956       280 13.4 12.1 11.0  8.2
7 1957       273 13.3 11.9 10.2  9.4
8 1958       248 13.0 10.1 11.8  8.6
9 1959       245 12.9 10.0 11.0  9.0
```

Notice the use of `<-` in the example above. This is the **assignment operator** in R. It can be used in either direction (`<-` or `->`). In the first line of the example above, the results of `read.table()` are stored in a variable called `traffic`. `traffic` is a **data frame**, R's preferred container for data. (More about data types in R as we go along.)

The `na.strings` argument can be used to specify codes for missing values. The following can be useful for SAS output, for example:

```
> read.csv('file.csv', na.strings=c('NA','','.','na')) -> someData
```

because SAS uses a period (`.`) to code missing data, but R by default reads that as string data, which forces the entire variable to be of character type instead of numeric.

For convenience the `mosaic` package provides `read.file()` which uses the file name to determine which of `read.csv()`, `read.table()`, and `load()` to use and sets the defaults to

- `header=TRUE`,
- `comment.char="#"`, and
- `na.strings=c('NA','','.','na')`

for `read.csv()` and `read.table()`.

```
> traffic <- read.file("http://www.calvin.edu/~rpruim/fastR/trafficTufte.txt")
```

B.3.3 Manually typing in data

If you need to enter a small data set by hand, the `scan()` function is quick and easy. Individual values are separated by white space or new lines. A blank line is used to signal the end of the data. By default, `scan()` is expecting decimal data (which it calls **double**, for double precision), but it is possible to tell `scan()` to expect something else, like **character** data (i.e., text). There are other options for data types, but numerical and text data will usually suffice for our purposes. See `?scan` for more information and examples.

```
myData1 <- scan()
15 18
12
21 23 50 15

myData1

myData2 <- scan(what="character")
"red" "red" "orange" "green" "blue" "blue" "red"

myData2
```


Be sure when using `scan()` that you remember to save your data somewhere. Otherwise you will have to type it again.

B.3.4 Creating data frames from vectors

The `scan()` function puts data into a **vector**, not a **data frame**. We can build a **data frame** for our data as follows.

```
> myDataFrame <- data.frame(color=myData2, number=myData1)
> myDataFrame
   color number
1    red     15
2    red     18
3 orange     12
4  green     21
5   blue     23
6   blue     50
7    red     15
```

B.3.5 Getting data from mySQL data bases

The `RMySQL` package allows direct access to data in MySQL data bases. This can be convenient when dealing with subsets of very large data sets. A great example of this is the 12 gigabytes of data from the Airline on-time performance dataset included in the 2009 Data Expo (`http://stat-computing.org/dataexpo/2009`). There is an online document describing this type of manipulation.

B.3.6 Generating data

The following code shows a number of ways to generate data systematically.

```
> x <- 5:20; x                  # all integers in a range
 [1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> # structured sequences
> seq(0, 50, by=5)
 [1]  0  5 10 15 20 25 30 35 40 45 50
> seq(0, 50, length=7)
[1]  0.00  8.33 16.67 25.00 33.33 41.67 50.00
> rep(1:5, each=3)
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
> rep(1:5, times=3)
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
> c(1:5, 10, 3:5)               # c() concatenates vectors
[1]  1  2  3  4  5 10  3  4  5
```

R can also sample from several different distributions.

```
> rnorm(10, mean=10, sd=2)  # random draws from normal distribution
 [1] 12.08  7.59  9.68 10.16 11.39  8.81  9.69  9.29  7.40 10.33
> x <- 5:20                 # all integers in a range
> sample(x, size=5)         # random sample of size 5 from x (no replacement)
```

```
[1] 20  7  9 17 10
```

Functions for sampling from other distributions include `rbinom()`, `rchisq()`, `rt()`, `rf()`, `rhyper()`, etc. See Section 8.9 for more information.

B.3.7 Saving Data

`write.table()` and `write.csv()` can be used to save data from R into delimited flat files.

```
> ddd <- data.frame(number=1:5, letter=letters[1:5])
> args(write.table)
function (x, file = "", append = FALSE, quote = TRUE, sep = " ",
    eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE,
    qmethod = c("escape", "double"), fileEncoding = "")
NULL
> write.table(ddd, "ddd.txt")
> write.csv(ddd, "ddd.csv")
> # this system call should work on a Mac or Linux machine
> system("head -20 ddd.txt ddd.csv")
```

Data can also be saved in native R format. Saving data sets (and other R objects) using `save()` has some advantages over other file formats:

- Complete information about the objects is saved, including attributes.
- Data saved this way takes less space and loads much more quickly.
- Multiple objects can be saved to and loaded from a single file.

The downside is that these files are only readable in R.

```
> abc <- "abc"
> ddd <- data.frame(number=1:5, letter=letters[1:5])
> save(ddd, abc, file="ddd.rda")   # saves both objects in a single file
> load("ddd.rda")                  # loads them both
```

For more on importing and exporting data, especially from other formats, see the *R Data Import/Export* manual available on CRAN.

B.4 Primary R Data Structures

B.4.1 Modes and other attributes

In R, data is stored in objects. Each **object** has a *name*, *contents*, and also various *attributes*. Attributes are used to tell R something about the kind of data stored in an object and to store other auxiliary information. Two important attributes shared by all objects are **mode** and **length**.

```
> w <- 2.5; x <- c(1,2); y <- "foo"; z <- TRUE; abc <- letters[1:3]
> mode(w); length(w)
[1] "numeric"
[1] 1
> mode(x); length(x)
```

```
[1] "numeric"
[1] 2
> mode(y); length(y)
[1] "character"
[1] 1
> y[1]; y[2]              # not an error to ask for y[2]
[1] "foo"
[1] NA
> mode(z); length(z)
[1] "logical"
[1] 1
> abc
[1] "a" "b" "c"
> mode(abc); length(abc)
[1] "character"
[1] 3
> abc[3]
[1] "c"
```

Each of the objects in the example above is a **vector**, an ordered container of values that all have the same mode.[1] The `c()` function concatenates vectors (or lists). Notice that `w`, `y`, and `z` are vectors of length 1. Missing values are coded as `NA` (not available). Asking for an entry "off the end" of a vector returns `NA`. Assigning a value "off the end" of a vector results in the vector being lengthened so that the new value can be stored in the appropriate location.

There are important ways that R has been optimized to work with vectors since they correspond to variables (in the sense of statistics). For categorical data, a **factor** is a special type of vector that includes an additional attribute called *levels*. A factor can be ordered or unordered (which can affect how statistics are done and graphs are made) and its elements can have mode `numeric` or `character`.

A **list** is similar to a vector, but its elements may be of different modes (including `list`, `vector`, etc.). A **data frame** is a list of vectors (or factors), each of the same length, but not necessarily of the same mode. This is R's primary way of storing data sets. An **array** is a multi-dimensional table of values that all have the same mode. A **matrix** is a 2-dimensional array.

The access operators (`[ ]` for vectors, matrices, arrays, and data frames, and `[[ ]]` for lists) are actually *functions* in R. This has some important consequences:

- Accessing elements is slower than in a language like C/C++ where access is done by pointer arithmetic.
- These functions also have named arguments, so you can see code like the following

```
> xm <- matrix(1:16, nrow=4); xm
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

[1] There are other modes in addition to the ones shown here, including `complex` (for complex numbers), `function`, `list`, `call`, and `expression`.

```
> xm[5]
[1] 5
> xm[,2]                    # this is 1 dimensional (a vector)
[1] 5 6 7 8
> xm[,2, drop=FALSE]         # this is 2 dimensional (still a matrix)
     [,1]
[1,]    5
[2,]    6
[3,]    7
[4,]    8
```

Many objects have a **dim attribute** that stores the dimension of the object. You can change it to change the shape (or even the number of dimensions) of a vector, matrix, or array. You can see all of the non-intrinsic attributes (mode and length are intrinsic) using `attributes()`, and you can set attributes (including new ones you make up) using `attr()`. Some attributes, like dimension, have special functions for accessing or setting. The `dim()` function returns the dimensions of an object as a vector. Alternatively the number of rows and columns can be obtained using `nrow()` and `ncol()`.

```
> ddd <- data.frame(number=1:5, letter=letters[1:5])
> attributes(ddd)
$names
[1] "number" "letter"

$row.names
[1] 1 2 3 4 5

$class
[1] "data.frame"
> dim(ddd)
[1] 5 2
> nrow(ddd)
[1] 5
> ncol(ddd)
[1] 2
> names(ddd)
[1] "number" "letter"
> row.names(ddd)
[1] "1" "2" "3" "4" "5"
```

B.4.2 What `is` it?

R provides a number of functions for testing the mode or class of an object.

```
> mode(xm); class(xm)
[1] "numeric"
[1] "matrix"
> c(is.numeric(xm), is.character(xm), is.integer(xm), is.logical(xm))
[1]  TRUE FALSE  TRUE FALSE
> c(is.vector(xm), is.matrix(xm), is.array(xm))
[1] FALSE  TRUE  TRUE
```

B.4.3 Changing modes (coercion)

If R is expecting an object of a certain mode or class but gets something else, it will often try to **coerce** the object to meet its expectations. You can also coerce things manually using one of the many `as.???()` functions.

```
> apropos("^as\\.")[1:10]      # just a small sample
 [1] "as.array"               "as.array.default"       "as.call"
 [4] "as.category"            "as.character"           "as.character.condition"
 [7] "as.character.Date"      "as.character.default"   "as.character.error"
[10] "as.character.factor"
> xm
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> # convert numbers to strings (this drops attributes, including dimension)
> as.character(xm)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15" "16"
> # convert matrix to vector
> as.vector(xm)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
> as.logical(xm)
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
> alpha <- c("a", "1", "b", "0.5")
> mode(alpha)
[1] "character"
> as.numeric(alpha)      # can't do the coercion, so NAs are introduced
[1]  NA 1.0  NA 0.5
> as.integer(alpha)      # notice coercion of 0.5 to 0
[1] NA  1 NA  0
```

B.5 More About Vectors

Vectors are so important in R that they deserve some additional discussion. In Section B.3.6 we learned how to generate some simple vectors. Here we will learn about some of the operations and functions that can be applied to vectors.

B.5.1 Names and vectors

We can give each position in a vector a name. This can be very handy for certain uses of vectors.

```
> myvec <- 1:5; myvec
[1] 1 2 3 4 5
> names(myvec) <- c('one','two','three','four','five'); myvec
  one   two three  four  five
    1     2     3     4     5
```

Names can also be specified as a vector is created using `c()`.

```
> another <- c(mean=10, sd=2, "trimmed mean"=9.7); another
        mean           sd trimmed mean
        10.0          2.0          9.7
```

B.5.2 Vectorized functions

Many R functions and operations are "vectorized" and can be applied not just to an individual value but to an entire vector, in which case they are applied componentwise and return a vector of transformed values. Most traditional mathematics functions are available and work this way.

```
> x <- 1:5; y <- seq(10, 60, by=10); z <- rnorm(10); x; y
[1] 1 2 3 4 5
[1] 10 20 30 40 50 60
> y + 1
[1] 11 21 31 41 51 61
> x * 10
[1] 10 20 30 40 50
> x < 3
[1]  TRUE  TRUE FALSE FALSE FALSE
> x^2
[1]  1  4  9 16 25
> log(x); log(x, base=10)            # natural and base 10 logs
[1] 0.000 0.693 1.099 1.386 1.609
[1] 0.000 0.301 0.477 0.602 0.699
```

Vectors can be combined into a matrix using `rbind()` or `cbind()`. This can facilitate side-by-side comparisons.

```
> # compare round() and signif() by binding rowwise into matrix
> rbind(round(z, digits=2), signif(z, digits=2))
     [,1] [,2] [,3] [,4] [,5] [,6]  [,7]  [,8] [,9] [,10]
[1,]   -3 0.43 0.87 -0.9 1.53   -2 -0.64 -0.36 0.35 -0.51
[2,]   -3 0.43 0.87 -0.9 1.50   -2 -0.64 -0.36 0.35 -0.51
```

B.5.3 Functions that act on vectors as vectors

Other functions, including many statistical functions, are designed to work on the vector as a vector. Often these return a single value (technically a vector of length 1), but other return types are used as appropriate.

```
> x <- 1:10; z <- rnorm(100)
> mean(z); sd(z); var(z); median(z)  # basic statistical functions
[1] 0.0831
[1] 0.99
[1] 0.98
[1] 0.0888
> range(z)                           # range returns a vector of length 2
[1] -2.55  2.45
```



```
> sum(x); prod(x)                          # sums and products
[1] 55
[1] 3628800
> z <- rnorm(5); z
[1] -1.285 -0.519  0.430  1.660 -0.690
> sort(z); rank(z); order(z)               # sort, rank, order
[1] -1.285 -0.690 -0.519  0.430  1.660
[1] 1 3 4 5 2
[1] 1 5 2 3 4
> rev(x)                                   # reverse x
 [1] 10  9  8  7  6  5  4  3  2  1
> diff(x)                                  # pairwise differences
[1] 1 1 1 1 1 1 1 1 1
> cumsum(x)                                # cumulative sum
 [1]  1  3  6 10 15 21 28 36 45 55
> cumprod(x)                               # cumulative product
 [1]       1       2       6      24     120     720    5040   40320  362880 3628800
> sum(x); prod(x)                          # sums and products
[1] 55
[1] 3628800
```

Whether a function is vectorized or treats a vector as a unit depends on its implementation. Usually, things are implemented the way you would expect. Occasionally you may discover a function that you wish were vectorized and is not. When writing your own functions, give some thought to whether they should be vectorized, and test them with vectors of length greater than 1 to make sure you get the intended behavior.

Some additional useful functions are included in Table B.1.

B.5.4 Recycling

When vectors operate on each other, the operation is done componentwise, recycling the shorter vector to match the length of the longer.

```
> x <- 1:5; y <- seq(10, 70, by=10)
> x + y
[1] 11 22 33 44 55 61 72
```

In fact, this is exactly how things like `x + 1` actually work. If `x` is a vector of length n, then `1` (a vector of length 1) is first recycled into a vector of length n; then the two vectors are added componentwise. Some vectorized functions that take multiple vectors as arguments will first use recycling to make them the same length.

B.5.5 Accessing elements of vectors

R allows for some very interesting and useful methods for accessing elements of a vector that combine the ideas above. First, recall that the `[ ]` operator is actually a function. Furthermore, it is vectorized.

Table B.1: Some useful R functions.

`cumsum()`
`cumprod()`
`cummin()`
`cummax()`
	Returns vector of cumulative sums, products, minima, or maxima.

`pmin(x,y,...)` `pmax(x,y,...)`	Returns vector of parallel minima or maxima where ith element is max or min of `x[i]`, `y[i]`, ...
`which(x)`	Returns a vector of indices of elements of `x` that are true. Typical use: `which(y > 5)` returns the indices where elements of `y` are larger than 5.
`any(x)`	Returns a `logical` indicating whether any elements of `x` are true. Typical use: `if ( any(y > 5) ) { ...}`.
`na.omit(x)`	Returns a vector with missing values removed.
`unique(x)`	Returns a vector with repeated values removed.
`table(x)`	Returns a table of counts of the number of occurrences of each value in `x`. The table is similar to a vector with names indicating the values, but it is not a vector.
`paste(x,y,...,` `sep=" ")`	Pastes `x` and `y` together componentwise (as strings) with `sep` between elements. Recycling applies.

```
> x <- seq(2, 20, by=2)
> x[1:5]; x[c(1, 4, 7)]
[1]  2  4  6  8 10
[1]  2  8 14
```

`[ ]` accepts `logicals` as arguments well. The boolean values (recycled, if necessary) are used to select or deselect elements of the vector.

```
> x <- seq(2, 20, by=2)
> x[c(TRUE, TRUE, FALSE)]      # skips every third element (recycling!)
[1]  2  4  8 10 14 16 20
> x[x > 10]                    # more typical use of boolean in selection
[1] 12 14 16 18 20
```

Negative indices are used to omit elements.

```
> x <- seq(2, 20, by=2)
> x[c(TRUE,TRUE,FALSE)]        # skips every third element (recycling!)
[1]  2  4  8 10 14 16 20
> x[x > 10]                    # more typical use of boolean in selection
[1] 12 14 16 18 20
```

Here are some more examples.

```
> notes <- toupper(letters[1:7]); a <- 1:5; b <- seq(10, 100, by=10)
> toupper(letters[5:10])
[1] "E" "F" "G" "H" "I" "J"
> paste(letters[1:5], 1:3, sep='-')
[1] "a-1" "b-2" "c-3" "d-1" "e-2"
```



```
> a+b
 [1]  11  22  33  44  55  61  72  83  94 105
> (a+b)[ a+b > 50]
[1]  55  61  72  83  94 105
> length((a+b)[a+b > 50])
[1] 6
> table(a+b > 50)
FALSE  TRUE
    4     6
```

B.6 Manipulating Data Frames

B.6.1 Adding new variables to a data frame

We can add additional variables to an existing data frame by simple assignment.

```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1   setosa    :50
 1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3   versicolor:50
 Median :5.80   Median :3.00   Median :4.35   Median :1.3   virginica :50
 Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
 3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
 Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5

> iris$SLength <- cut(iris$Sepal.Length, 4:8)    # cut places data into bins

> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1   setosa    :50
 1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3   versicolor:50
 Median :5.80   Median :3.00   Median :4.35   Median :1.3   virginica :50
 Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
 3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
 Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5
  SLength
 (4,5]:32
 (5,6]:57
 (6,7]:49
 (7,8]:12
```

It is an error to add a vector of the wrong length.

The CPS data frame contains data from a Current Population Survey (current in 1985, that is). Two of the variables in this data frame are age and educ. We can estimate the number of years a worker has been in the workforce if we assume they have been in the workforce since completing their education and that their age at graduation is 6 more than the number of years of education obtained. We can this as a new variable in the data frame simply by assigning to it:

```
> CPS$workforce.years <- with(CPS, age - 6 - educ)
> favstats(CPS$workforce.years)
 min Q1 median Q3 max mean   sd   n missing
  -4  8     15 26  55 17.8 12.4 534       0
```

In fact this is what was done for all but one of the cases to create the exper variable that is already in the CPS data.

```
> with(CPS, table(exper - workforce.years))
```

```
  0   4
533   1
```

B.6.2 Dropping variables

Since we already have educ, there is no reason to keep our new variable. Let's drop it. Notice the clever use of the minus sign.

```
> CPS1 <- subset(CPS, select = -workforce.years)
```

Any number of variables can be dropped or kept in this manner by supplying a vectors of variables names.

```
> CPS1 <- subset(CPS, select = -c(workforce.years,exper))
```

If we only want to work with the first few variables, we can discard the rest in a similar way. Columns can be specified by number as well as name (but this can be dangerous if you are wrong about where the columns are):

```
> CPSsmall <- subset(CPS, select=1:4)
> head(CPSsmall,2)
```

```
  wage educ race sex
1  9.0   10    W   M
2  5.5   12    W   M
```

B.6.3 Renaming variables

Both the column (variable) names and the row names of a data frames can be changed by simple assignment using names() or row.names().

```
> ddd                       # small data frame we defined earlier
```

```
  number letter
1      1      a
2      2      b
3      3      c
4      4      d
5      5      e
```

```
> row.names(ddd) <- c("Abe","Betty","Claire","Don","Ethel")
> ddd                       # row.names affects how a data.frame prints
```

```
       number letter
Abe         1      a
Betty       2      b
Claire      3      c
Don         4      d
Ethel       5      e
```

More interestingly, it is possible to reset just individual names with the following syntax.

```
> row.names(ddd)[2] <- "Bette"         # misspelled a name, let's fix it
> row.names(ddd)
```

```
[1] "Abe"    "Bette"  "Claire" "Don"    "Ethel"
```


The faithful data set (in the datasets package, which is always available) has very unfortunate names.

```
> names(faithful)
```

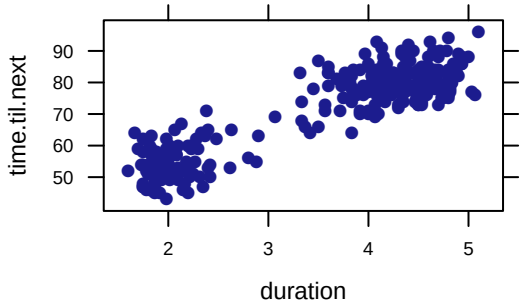
```
[1] "eruptions" "waiting"
```

The measurements are the duration of an euption and the time until the subsequent eruption, so let's give it some better names.

```
> names(faithful) <- c('duration', 'time.til.next')
> head(faithful, 3)
```

```
  duration time.til.next
1     3.60            79
2     1.80            54
3     3.33            74
```

```
> xyplot(time.til.next ~ duration, faithful)
```



If the variable containing a data frame is modified or used to store a different object, the original data from the package can be recovered using data().

```
> data(faithful)
> head(faithful, 3)
```

```
  eruptions waiting
1     3.60      79
2     1.80      54
3     3.33      74
```

If we want to rename a variable, we can do this using names(). For example, perhaps we want to rename educ (the second column) to education.

```
> names(CPS)[2] <- 'education'
> CPS[1,1:4]
```

```
  wage education race sex
1    9        10    W   M
```

If we don't know the column number (or generally to make our code clearer), a few more keystrokes produces

See Section B.5 for information that will make it clearer what is going on here.

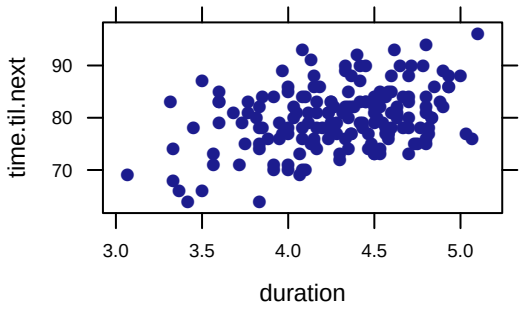
```
> names(CPS)[names(CPS) == 'education'] <- 'educ'
> CPS[1,1:4]
```

```
  wage educ race sex
1    9   10    W   M
```

B.6.4 Creating subsets

We can also use `subset()` to reduce the size of a data set by selecting only certain rows.

```
> data(faithful)
> names(faithful) <- c('duration', 'time.til.next')
> # any logical can be used to create subsets
> faithfulLong <- subset(faithful, duration > 3)
> xyplot( time.til.next ~ duration, faithfulLong )
```



Of course, if all we want to do is produce a graph, there is no reason to create a new data frame. The plot above could also be made with

```
> xyplot( time.til.next ~ duration, faithful, subset=duration > 3 )
```

B.6.5 Merging datasets

The `fusion1` data frame in the `fastR` package contains genotype information for a SNP (single nucleotide polymorphism) in the gene *TCF7L2*. The `pheno` data frame contains phenotypes (including type 2 diabetes case/control status) for an intersecting set of individuals. We can merge these together to explore the association between genotypes and phenotypes using `merge()`.

```
> require(fastR)
> head(fusion1,3)
     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
3  9380 RS12255372        1       3       4       GT     0     0     1     1
> head(pheno,3)
    id     t2d  bmi sex  age smoker chol waist weight height   whr sbp dbp
1 1002    case 32.9   F 70.8 former 4.57 112.0   85.6    161 0.987 135  77
2 1009    case 27.4   F 53.9  never 7.32  93.5   77.4    168 0.940 158  88
3 1012 control 30.5   M 53.9 former 5.02 104.0   94.6    176 0.933 143  89
> # merge fusion1 and pheno keeping only id's that are in both
> fusion1m <- merge(fusion1, pheno, by.x='id', by.y='id', all.x=FALSE, all.y=FALSE)
> head(fusion1m, 3)
    id     marker markerID allele1 allele2 genotype Adose Cdose Gdose Tdose     t2d
1 1002 RS12255372        1       3       3       GG     0     0     2     0    case
2 1009 RS12255372        1       3       3       GG     0     0     2     0    case
3 1012 RS12255372        1       3       3       GG     0     0     2     0 control
```



```
   bmi sex  age smoker chol waist weight height   whr sbp dbp
1 32.9   F 70.8 former 4.57 112.0   85.6    161 0.987 135  77
2 27.4   F 53.9  never 7.32  93.5   77.4    168 0.940 158  88
3 30.5   M 53.9 former 5.02 104.0   94.6    176 0.933 143  89
```

In this case, since the values are the same for each data frame, we could collapse `by.x` and `by.y` to `by` and collapse `all.x` and `all.y` to `all`. The first of these specifies which column(s) to use to identify matching cases. The second indicates whether cases in one data frame that do not appear in the other should be kept (`TRUE`) or dropped (filling in `NA` as needed) or dropped from the merged data frame.

Now we are ready to begin our analysis.

```
> xtabs(~t2d + genotype + marker, fusion1m)

, , marker = RS12255372

         genotype
t2d        GG  GT  TT
  case    737 375  48
  control 835 309  27
```

B.6.6 Slicing and dicing

`reshape()` provides a flexible way to change the arrangement of data. It was designed for converting between long and wide versions of time series data and its arguments are named with that in mind.

A common situation is when we want to convert from a wide form to a long form because of a change in perspective about what a unit of observation is. For example, in the `traffic` data frame, each row is a year, and data for multiple states are provided.

```
> traffic

  year cn.deaths   ny   cn   ma   ri
1 1951       265 13.9 13.0 10.2  8.0
2 1952       230 13.8 10.8 10.0  8.5
3 1953       275 14.4 12.8 11.0  8.5
4 1954       240 13.0 10.8 10.5  7.5
5 1955       325 13.5 14.0 11.8 10.0
6 1956       280 13.4 12.1 11.0  8.2
7 1957       273 13.3 11.9 10.2  9.4
8 1958       248 13.0 10.1 11.8  8.6
9 1959       245 12.9 10.0 11.0  9.0
```

We can reformat this so that each row contains a measurement for a single state in one year.

```
> longTraffic <-
        reshape(traffic[,-2], idvar="year", ids=row.names(traffic),
        times=names(traffic)[3:6], timevar="state",
        varying=list(names(traffic)[3:6]),
        v.names="deathRate",
        direction="long")
> head(longTraffic)

        year state deathRate
1951.ny 1951    ny      13.9
1952.ny 1952    ny      13.8
1953.ny 1953    ny      14.4
1954.ny 1954    ny      13.0
1955.ny 1955    ny      13.5
1956.ny 1956    ny      13.4
```

And now we can reformat the other way, this time having all data for a given state form a row in the data frame.

```
> stateTraffic <- reshape(longTraffic, direction='wide',
                          v.names="deathRate", idvar="state", timevar="year")
> stateTraffic
        state deathRate.1951 deathRate.1952 deathRate.1953 deathRate.1954
1951.ny    ny           13.9           13.8           14.4           13.0
1951.cn    cn           13.0           10.8           12.8           10.8
1951.ma    ma           10.2           10.0           11.0           10.5
1951.ri    ri            8.0            8.5            8.5            7.5
        deathRate.1955 deathRate.1956 deathRate.1957 deathRate.1958 deathRate.1959
1951.ny           13.5           13.4           13.3           13.0           12.9
1951.cn           14.0           12.1           11.9           10.1           10.0
1951.ma           11.8           11.0           10.2           11.8           11.0
1951.ri           10.0            8.2            9.4            8.6            9.0
```

In simpler cases, `stack()` or `unstack()` may suffice. `Hmisc` also provides `reShape()` as an alternative to `reshape()`.

B.7 Functions in R

Functions in R have several components:

- a **name** (like `histogram`)[2]
- an ordered list of named **arguments** that serve as inputs to the function

  These are matched first by name and then by order to the values supplied by the call to the function. This is why we don't always include the argument name in our function calls. On the other hand, the availability of names means that we don't have to remember the order in which arguments are listed.

  Arguments often have **default values** which are used if no value is supplied in the function call.
- a **return value**

  This is the output of the function. It can be assigned to a variable using the assignment operator (`=`, `<-`, or `->`).
- **side effects**

  A function may do other things (like make a graph or set some preferences) that are not necessarily part of the return value.

When you read the help pages for an R function, you will see that they are organized in sections related to these components. The list of arguments appears in the **Usage** section along with any default values. Details about how the arguments are used appear in the **Arguments** section. The return value is listed in the **Value** section. Any side effects are typically mentioned in the **Details** section.

Now let's try writing our own function. Suppose you frequently wanted to compute the mean, median, and standard deviation of a distribution. You could make a function to do all three to save some typing. Let's name our function `favstats()`. `favstats()` will have one argument, which we are assuming will be a vector of numeric values.[3] Here is how we could define it:

[2] Actually, it is possible to define functions without naming them; and for short functions that are only needed once, this can actually be useful.

[3] There are ways to check the **class** of an argument to see if it is a data frame, a vector, numeric, etc. A really robust function should check to make sure that the values supplied to the arguments are of appropriate types.


```
> favstats <- function(x) {
     mean(x)
     median(x)
     sd(x)
 }
> favstats((1:20)^2)
[1] 128
```

The first line says that we are defining a function called `favstats()` with one argument, named `x`. The lines surrounded by curly braces give the code to be executed when the function is called. So our function computes the mean, then the median, then the standard deviation of its argument.

But as you see, this doesn't do exactly what we wanted. So what's going on? The value returned by the last line of a function is (by default) returned by the function to its calling environment, where it is (by default) printed to the screen so you can see it. In our case, we computed the mean, median, and standard deviation, but only the standard deviation is being returned by the function and hence displayed. So this function is just an inefficient version of `sd()`. That isn't really what we wanted.

We can use `print()` to print out things along the way if we like.

```
> favstats <- function(x) {
     print(mean(x))
     print(median(x))
     print(sd(x))
 }
> favstats((1:20)^2)
[1] 144
[1] 110
[1] 128
```

Alternatively, we could use a combination of `cat()` and `paste()`, which would give us more control over how the output is displayed.

```
> altfavstats <- function(x) {
     cat(paste("  mean:", format(mean(x),4),"\n"))
     cat(paste(" edian:", format(median(x),4),"\n"))
     cat(paste("    sd:", format(sd(x),4),"\n"))
 }
> altfavstats((1:20)^2)
  mean: 144
 edian: 110
    sd: 128
```

Either of these methods will allow us to see all three values, but if we try to store them ...

```
> temp <- favstats((1:20)^2)
[1] 144
[1] 110
[1] 128
> temp
[1] 128
```

A function in R can only have one return value, and by default it is the value of the last line in the function. In the preceding example we only get the standard deviation since that is the value we calculated last.

We would really like the function to return all three summary statistics. Our solution will be to store all three in a vector and return the vector.[4]

[4] If the values had not all been of the same mode, we could have used a list instead.

```
> favstats <- function(x) {
         c(mean(x), median(x), sd(x))
 }
> favstats((1:20)^2)
[1] 144 110 128
```

Now the only problem is that we have to remember which number is which. We can fix this by giving names to the slots in our vector. While we're at it, let's add a few more favorites to the list. We'll also add an explicit `return()`.

```
> favstats <- function(x) {
     result <- c(min(x), max(x), mean(x), median(x), sd(x))
     names(result) <- c("min","max","mean","median","sd")
     return(result)
 }
> favstats((1:20)^2)
   min    max   mean median     sd
     1    400    144    110    128
> summary(Sepal.Length~Species, data=iris, fun=favstats)
Sepal.Length    N=150

+-------+----------+---+---+---+----+------+-----+
|       |          |N  |min|max|mean|median|sd   |
+-------+----------+---+---+---+----+------+-----+
|Species|setosa    | 50|4.3|5.8|5.01|5.0   |0.352|
|       |versicolor| 50|4.9|7.0|5.94|5.9   |0.516|
|       |virginica | 50|4.9|7.9|6.59|6.5   |0.636|
+-------+----------+---+---+---+----+------+-----+
|Overall|          |150|4.3|7.9|5.84|5.8   |0.828|
+-------+----------+---+---+---+----+------+-----+
> aggregate(Sepal.Length~Species, data=iris, FUN=favstats)
     Species Sepal.Length.min Sepal.Length.max Sepal.Length.mean Sepal.Length.median
1     setosa            4.300            5.800             5.006                5.000
2 versicolor            4.900            7.000             5.936                5.900
3  virginica            4.900            7.900             6.588                6.500
  Sepal.Length.sd
1           0.352
2           0.516
3           0.636
```

Notice the use of `::` to select the `favstats()` function from the `mosaic` package rather than the one we just defined.

Notice how nicely this works with `aggregate()` and with the `summary()` function from the `Hmisc` package. You can, of course, define your own favorite function to use with `summary()`. The `favstats()` function in the `mosaic` package includes the quartiles, mean, standard, deviation, sample size and number of missing observations.

```
> mosaic::favstats(rnorm(100))
  min     Q1  median    Q3 max   mean    sd   n missing
 -1.8 -0.598 -0.0173 0.662 2.5 0.0201 0.998 100       0
```

Exercises

B.1. Using `faithful` data frame, make a scatter plot of eruption duration times vs. the time since the previous eruption.

B.2. 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.

```
> names(fusion1)
 [1] "id"       "marker"   "markerID" "allele1"  "allele2"  "genotype" "Adose"
 [8] "Cdose"    "Gdose"    "Tdose"
> names(fusion2)
 [1] "id"       "marker"   "markerID" "allele1"  "allele2"  "genotype" "Adose"
 [8] "Cdose"    "Gdose"    "Tdose"
```

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.

Bibliography

[Cob07] G. W. Cobb, *The introductory statistics course: a Ptolemaic curriculum?*, Technology Innovations in Statistics Education **1** (2007), no. 1.

[Fis25] R. A. Fisher, *Statistical methods for research workers*, Oliver & Boyd, 1925.

[Fis70] ———, *Statistical methods for research workers*, 14th ed., Oliver & Boyd, 1970.

[Gal88] F. Galton, *Co-relations and their measurement, chiefly from anthropometric data*, Proceedings of the Royal Society of London (1888), 135–145.

[GN02] A Gelman and D Nolan, *Teaching statistics: a bag of tricks*, Oxford University Press, 2002.

[Gou10] R. Gould, *Statistics and the modern student*, International Statistical Review **78** (2010), no. 2, 297–315.

[HBQ04] N. J. Horton, E. R. Brown, and L. Qian, *Use of R as a toolbox for mathematical statistics exploration*, The American Statistician **58** (2004), no. 4, 343–357.

[HK11] N. J. Horton and K. Kleinman, *Using R for data management, statistical analysis, and graphics*, 1st ed., Chapman & Hall, 2011.

[HS05] N.J. Horton and S.S. Switzer, *Statistical methods in the journal*, New England Journal of Medicine **353** (2005), no. 18, 1977–1979.

[HSLS02] N. J. Horton, R. Saitz, N. M. Laird, and J. H. Samet, *A method for modeling utilization data from multiple sources: application in a study of linkage to primary care*, Health Services and Outcomes Research Methodology **3** (2002), 211–223.

[Kap09] D. Kaplan, *Statistical modeling: A fresh approach*, CreateSpace.com, 2009.

[KHF+03] S. G. Kertesz, N. J. Horton, P. D. Friedmann, R. Saitz, and J. H. Samet, *Slowing the revolving door: stabilization programs reduce homeless persons' substance use after detoxification*, Journal of Substance Abuse Treatment **24** (2003), no. 3, 197–207.

[LSH+06] M. J. Larson, R. Saitz, N. J. Horton, C. Lloyd-Travaglini, and J. H. Samet, *Emergency department and hospital utilization among alcohol and drug-dependent detoxification patients without primary medical care*, American Journal of Drug and Alcohol Abuse **32** (2006), 435–452.

[LSS+02] J. Liebschutz, J. B. Savetsky, R. Saitz, N. J. Horton, C. Lloyd-Travaglini, and J. H. Samet, *The relationship between sexual and physical abuse and substance abuse consequences*, Journal of Substance Abuse Treatment **22** (2002), no. 3, 121–128.

[MM06] D. S. Moore and G. P. McCabe, *Introduction to the practice of statistics*, 3rd ed., W.H.Freeman and Company, 2006.

[NT10] D. Nolan and D. Temple Lang, *Computing in the statistics curriculum*, The American Statistician **64** (2010), no. 2, 97–107.

[Rep10] Committee Report, *Common core state standards for mathematics*, Tech. report, Common Core State Standards Initiative, 2010.

[RSHS01] V. W. Rees, R. Saitz, N. J. Horton, and J. H. Samet, *Association of alcohol consumption with HIV sex and drug risk behaviors among drug users*, Journal of Substance Abuse Treatment **21** (2001), no. 3, 129–134.

[Sal01] D. Salsburg, *The lady tasting tea: How statistics revolutionized science in the twentieth century*, W.H. Freeman, New York, 2001.

[SHL+05] R. Saitz, N. J. Horton, M. J. Larson, M. Winter, and J. H. Samet, *Primary medical care and reductions in addiction severity: a prospective cohort study*, Addiction **100** (2005), no. 1, 70–78.

[SLH+03] J. H. Samet, M. J. Larson, N. J. Horton, K. Doyle, M. Winter, and R. Saitz, *Linking alcohol and drug dependent adults to primary medical care: A randomized controlled trial of a multidisciplinary health intervention in a detoxification unit*, Addiction **98** (2003), no. 4, 509–516.

[SLH+04] R. Saitz, M. J. Larson, N. J. Horton, M. Winter, and J. H. Samet, *Linkage with primary medical care in a prospective cohort of adults with addictions in inpatient detoxification: room for improvement*, Health Services Research **39** (2004), no. 3, 587–606.

[SLH+05] C. W. Shanahan, A. Lincoln, N. J. Horton, R. Saitz, M. J. Larson, and J. H. Samet, *Relationship of depressive symptoms and mental health functioning to repeat detoxification*, Journal of Substance Abuse Treatment **29** (2005), 117–123.

[Spe11] T. Speed, *Simulation*, IMS Bulletin **40** (2011), 18.

[Tuf01] E. R. Tufte, *The visual display of quantitative information*, 2nd ed., Graphics Press, Cheshire, CT, 2001.

[WSH+07] J. D. Wines, R. Saitz, N. J. Horton, C. Lloyd-Travaglini, and J. H. Samet, *Overdose after detoxification: a prospective study*, Drug and Alcohol Dependence **89** (2007), 161–169.