Introduction to Nonparametric Statistics Using R

Michael Hughes

STA/DSC 333: Nonparametric Statistics
Miami University
Oxford, Ohio

Revised January 2012

Table of contents

STA333 Lecture 1
The bare essentials of R (part 1)

1.1 What is R?

R is a system for statistical analysis and graphics, and is a dialect of the S language created by the AT&T Bell Laboratories (now Lucent Technologies). S is available as the commercial software S-PLUS by Insightful Corporation. R, however, is an open-source project developed by dozens of volunteers for more than a decade and is available to anyone from the Internet under the GNU General Public License.

R consists of a "base" package and numerous optional user-contributed add-on packages for more specialized tasks in statistical and other types of analyses. It is fast becoming the primary tool for contemporary statistical computing. Increasingly, implementations of ground-breaking statistical methodologies appear first as R add-on packages. In some scientific disciplines such as bioinformatics and genetics, R is already the workhorse of choice for statistical analyses.

People use R to analyze data in countless fields such as agriculture, astrophysics, botany, climatology, environmental science, econometrics, engineering, business and finance, genetics, geography, psychology, social sciences, zoology, and many more.

Here are some things you might want to know about R if you are encountering it for the first time:

- **R is an environment for statistical computing and graphics.** You can think of it as a combination of a statistics package and a programming language.
- **R is completely free.** Because it is "open-source" software, you don't have to pay for it, and you can make any modifications you want to it.
- **R runs on all popular operating systems**, including Windows, MacOS, Linux, and many Unix variants.

- **R is <u>not</u> supported by any commercial enterprise**, but it has a very active development community. The on-line manuals are complete (but not always helpful). This is one of the reasons why there are numerous sources of support such as user groups, Wikis, etc. Also, there are many books on R.
- **Every standard statistical analysis can be done in R.** R has an enormous number of standard and cutting-edge statistical capabilities built in, a wide variety of free add-on packages that add even more functionality, and you can extend it further by developing your own customized procedures.
- **R is mostly command-line driven**, although some graphical user interfaces (GUIs) have been developed. These, however, limit R's capabilities.

The last item in the above list deserves a bit more discussion. The truth is that R is not the simplest software to use. Frankly, if you are only used to "point-and-click" menu-driven interfaces with computer software, you are in for an awakening.

R will not hold your hand. It may drive you to drink (if you don't already), or even cause premature hair loss! In all seriousness, you will need to be prepared to think logically about the tasks you want to perform, and then provide R with written logical code (called *scripts*) to carry out those tasks. This makes it harder to use and may require you to adopt a novel way of thinking about how you do computer work, but it also allows for great flexibility and customization of what you may ask of the software. This can be very rewarding once you achieve a certain level of mastery of R.

In this class, we will only begin to scratch the surface of R's capabilities. So be very patient, work a lot with it, make lots of mistakes, try to figure out what is going on and what is going wrong, and learn as you go.

1.2 Installing R

R must first be installed on your computer. The most current version may be downloaded from a Comprehensive R Archive Network (CRAN) "mirror" site on the Internet. There are links at the R home page at `http://cran.r-project.org/`.

Windows installation of the R base package:

1. Go to the R site at `http://cran.r-project.org`.
2. Under `Download and Install R`, click on **Download R for Windows**.
3. Click on **install R for the first time**.
4. Click on the link for the set-up program. As this is being written, the most current version of R is version 2.14.1, and the set-up file is named **R-2.14.1-win32.exe**.
5. Click **Run**. If you are asked to choose options while installing, the defaults should be fine. I do suggest putting a shortcut for R out on your desktop.

MacOS X (Leopard, Snow Leopard, Lion) installation of the R base package:

1. Go to the R site at `http://cran.r-project.org`.
2. Under `Download and Install R`, click on **Download R for MacOS X.**
3. Click on the link for the set-up file **R-2.14.0.pkg**. As this is being written, the most current version of R is version 2.14.0.
4. Following the installation instructions. If you are asked to choose options while installing, the defaults should be fine.

Help files and FAQs are available at these sites, should you need them.

1.3 R add-on packages

The base installation of R includes a number of packages in its library. These are collections of both functions and data, and will accommodate most of what we will need for this course. However, there are nearly 700 additional add-on packages that reside at CRAN. These comprise add-ons that perform more very specialized tasks, contain user-contributed datasets, or are even designed to accompany published textbooks.

The `base` package, the `stats` package, and the `datasets` package (among several others) are automatically loaded (or in R lingo, "attached") at the beginning of a session. Use the command `sessionInfo()` to see what packages are currently attached:

```
> sessionInfo()
R version 2.10.1 Patched (2009-12-22 r45703)
i386-pc-mingw32

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base
```

If you type `library()`, you will get a pop-up list of all other R packages currently installed on your computer. If you want to "attach" an add-on package, find the name in the list and type `library(`*`packagename`*`)`. For example,

```
> library(coin)
Loading required package: survival
Loading required package: splines
Loading required package: mvtnorm
Loading required package: modeltools
Loading required package: stats4

> sessionInfo()
R version 2.10.1 Patched (2009-12-22 r45703)

attached base packages:
[1] stats4    splines   stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
[1] coin_0.6-9     modeltools_0.2-15 mvtnorm_0.9-0     survival_2.34-1
```

Installing a package that doesn't come with the base installation

Under Windows or other systems where the R console has menus across the top, choose Install Package(s)... from the Packages menu and follow the instructions. You must have a live Internet connection to do so.

I will let you know when we need additional add-on packages for STA/DSC333.

1.4 Starting R

We will use R primarily in an interactive mode: in other words, we give R a written instruction, and it responds with an answer.

To start R in Windows, double-click the R icon on your desktop, or find the program under the Start menu. When the program opens, you get a window called the R console, which contains something like this:

```
R version 2.10.1 Patched (2009-12-22 r45703)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

When R starts, it will first search for any work that you've saved previously in the current directory. If it finds some, that work will be reloaded and a brief message will be printed that a previous "workspace" has been restored. When you later quit R, you will be asked if you want to save the current session. Thus, you may continue to work from session to session without starting over from scratch.

The command prompt > is where you dialog with R. This is where you type or paste commands to be processed by R. Your commands are executed when you hit the Enter key.

1.5 Using R as a calculator

The simplest application of R is as a basic calculator. Use the symbols +, –, *, and / to respectively denote addition, subtraction, multiplication, and division. Exponentiation is denoted using ^. The usual order of mathematical operations is preserved by R, and parentheses may be used to give priority to certain intermediate calculations.

```
> 3 + 5
[1] 8
> 3/4
[1] 0.75
> 4^2
[1] 16
> 4 + 3 * 2
[1] 10
> (4 + 3)*2
[1] 14
```

Note that each "answer" is preceded by `[1]`. This will make sense once we define what a data vector is shortly.

Built-in functions. Numerous mathematical and statistical functions are also provided in R. Such functions have specific names that R recognizes which you must type, followed by a required pair of parentheses, inside which you provide necessary function arguments.

Below are a few common function examples. *(The # is a "comment character." All text in the line following a # is ignored by R. In the examples below, the comments are provided simply as an aid in comprehending what is being done. You don't need to type them if you are trying the examples yourself.)*

```
> sqrt(40)          # the square root function
[1] 6.324555
> squareroot(40)    # R doesn't recognize a function named "squareroot"
Error: could not find function "squareroot"
> exp(1)            # the exp function: exp(x) = e^x
[1] 2.718282
> log(14)           # the log base-e function (natural log, or ln)
[1] 2.639057
```

R produces an error message (and no answer) when it encounters something it doesn't recognize.

Many functions have extra arguments that allow you to change their default settings. For example, suppose instead of computing $\log_e 14 = \ln 14$, you wanted to calculate $\log_{10} 14$. To change the base of the log, change the `base=` argument from the default of e to 10 as follows:

```
> log(14, base=10)
[1] 1.146128
```

R expects a certain order for function arguments. For this reason, you can just provide the values of the arguments in the right order and R will use them correctly. In the above example, all R

really knows is that the base is the second argument in parentheses separated by a comma. So we could get the same result without explicitly typing the name of the argument:

```
> log(14, 10)
[1] 1.146128
```

R will not read your mind, so if you specify the arguments in the wrong order, you'll get the wrong answer:

```
> log(10, 14)
[1] 0.8725029
```

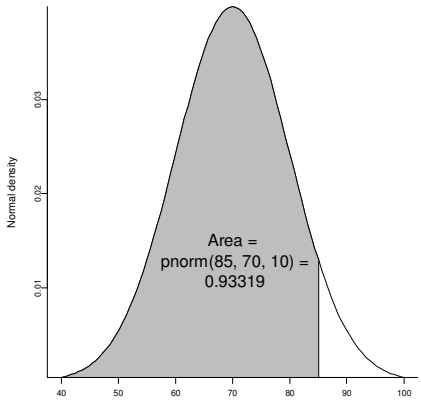
This is not $\log_{10} 14$, but rather $\log_{14} 10$.

A very useful R function is `pnorm()`, which replaces the usual normal probability table you encountered in introductory statistics. `pnorm()` returns the area to the left of a particular point on a specified normal curve:

```
> pnorm(-1.96, mean=0, sd=1)
[1] 0.02499790

> pnorm(85, 70, 10)
[1] 0.9331928

> 1 - pnorm(85, 70, 10)
[1] 0.0668072

> pnorm(85, 70, -10)
[1] NaN
Warning message:
In pnorm(q, mean, sd, lower.tail, log.p) : NaNs produced
```

The first example finds the area to the left of –1.96 on the standard normal distribution, i.e. the z-curve. (Recall from the old fashioned z-table that this value was 0.025.)

The second example finds the area to the left of 85 on the normal curve with mean $\mu = 70$ and standard deviation $\sigma = 10$ (i.e., it finds $P(X \leq 85)$ where $X \sim N(70, 10)$). See the picture at right.



The third example finds the area to the right of 85 on the same curve.

The fourth example didn't produce any result, but rather the warning `NaNs produced`, which means "Not a Number". Can you guess the reason why?

1.6 Assignment

We will usually need to name a value so that we can reference it later. The process of naming a value is called *assignment*. Assignment is done via the <– operator, which takes the value on the left-hand side and assigns it to an R "object" with the name you provide on the right.

Assignment does not produce any printed output. Instead, you need to call the assigned name to see the value:

```
> x <- 10           # x is assigned the value 10
> x                 # call the value of x
[1] 10

> x <- x + 5        # Replace x with the value x + 5
> x                 # call the value of x
[1] 15

> y <- x*2          # assign a new object named y w/ twice the value of x
> y
[1] 30
```

Acceptable names may be made out of letters, numbers, and the period (.) symbol. The only practical restriction is that names must start with a letter:

```
> age <- 44
> my.age.in.dog.years <- age/7
> my.age.in.dog.years               # call the value
[1] 6.285714

> MyAgeInDays <- age*365
> MyAgeInDays                       # call the value
[1] 16060

> myageindays                       # call the value
Error: object "myageindays" not found
```

The last call in the above example exposes a critical fact: R is case-sensitive. `MyAgeInDays` is not recognized as the same object as `myageindays`. This also applies to functions and other R commands, so be very careful:

```
> Sqrt(16)
Error: could not find function "Sqrt"
```

1.7 Data vectors: using the `c()` function

So far we've been looking at R objects that contain only a single value. However, as is common in statistics, we often measure a variable on a set of individuals. For example, suppose the variable X represents the GPA of any arbitrary Miami University student. If we collect a sample of $n = 8$ MU students in section A of STA333 and measure their GPAs, we have 8 values of the variable X. We always use n to be the number of observations, and we reference different observations via an index, as in $x_1, x_2, \ldots, x_8$ (in general, $x_1, x_2, \ldots, x_n$).

Suppose the 8 observed GPAs are 3.13, 3.55, 2.92, 2.73, 3.00, 3.18, 2.66, and 3.76. To store these values in R, we use a **data vector**. A data vector is a concatenation of different values of a variable, and can be made using the `c()` function:

```
> gpa.section.a <- c(3.13, 3.55, 2.92, 2.73, 3.00, 3.18, 2.66, 3.76)
```

The values are entered separated by commas. Once stored, the vector object acts just like a single value object like we saw earlier. You call it by typing its name:

```
> gpa.section.a
[1] 3.13 3.55 2.92 2.73 3.00 3.18 2.66 3.76
```

And now, the answer to the mysterious `[1]`: it simply refers to the index of the first observation in the printed row.

You can also concatenate (i.e., combine) data vectors:

```
> gpa.section.b <- c(3.14, 3.13, 3.25, 2.93, 3.73, 3.50, 2.70)
> all.gpa <- c(gpa.section.a, gpa.section.b)
> all.gpa
 [1] 3.13 3.55 2.92 2.73 3.00 3.18 2.66 3.76 3.14 3.13 3.25 2.93 3.73
[14] 3.50 2.70
```

Data vectors have a type. One restriction on data vectors is that all the values must be of the same type. Type can be *numeric*, as in `all.gpas` above; *character strings*, as in

```
> simpsons <- c("Homer", "Marge", "Bart", "Lisa", "Maggie")
```

or *logical*, which we will see shortly. Character strings are made with matching quotes (you can use either double or single quotes). For example, the vector object `zzz` defined below is character:

```
> zzz <- c('1', '2', '3', '4', '5', '6')
```

If you try to mix data types within a vector, they will all be coerced to character strings, which prevents mathematical operations.

1.8 Using functions on data vectors

R has many useful functions that work on the elements of a data vector. Below are some illustrative examples you are invited to try out:

```
> gpa.section.a
[1] 3.13 3.55 2.92 2.73 3.00 3.18 2.66 3.76
> sum(gpa.section.a)       # returns the sum of vector elements
[1] 24.93
> length(gpa.section.a)    # returns the vector length (i.e., n)
[1] 8
> mean(gpa.section.a)      # returns the vector mean
[1] 3.11625
> min(gpa.section.a)       # returns the minimum value
[1] 2.66
> max(gpa.section.a)       # returns the maximum value
[1] 3.76
> sort(gpa.section.a)      # returns the sorted vector elements
[1] 2.66 2.73 2.92 3.00 3.13 3.18 3.55 3.76
```

Note that the mean could also be computed formulaically:

```
> sum(gpa.section.a)/length(gpa.section.a)
[1] 3.11625
```

Vectorization of functions. One of the great efficiencies in an "object-oriented" language like R is that functions apply to vectors much like they apply to single items. In other words, most functions will apply to each element of a data vector at the same time. Consider this example, which adds a curve of 0.05 to everyone's GPA in section A:

```
> gpa.section.a
[1] 3.13 3.55 2.92 2.73 3.00 3.18 2.66 3.76
> gpa.section.a.curve <- gpa.section.a + 0.05
> gpa.section.a.curve
[1] 3.18 3.60 2.97 2.78 3.05 3.23 2.71 3.81
```

Or, we could calculate how far each GPA is away from the section average (known as the "deviations"):

```
> gpa.section.a.deviations <- gpa.section.a - mean(gpa.section.a)
> gpa.section.a.deviations
[1]  0.01375  0.43375 -0.19625 -0.38625 -0.11625  0.06375 -0.45625
[8]  0.64375
```

► **Example:** Use R to calculate the standard deviation of the values 3, 7, 6, 9, 12 and 8. First, recall the formula for the sample standard deviation:

$$s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$$

Here is R code that does the calculation (with comments added):

```
> x <- c(3, 7, 6, 9, 12, 8)     # create the data vector
> xbar <- mean(x)               # calculate the mean, store as object
> x - xbar                      # calculate each deviation
[1] -4.5 -0.5 -1.5  1.5  4.5  0.5
> (x-xbar)^2                    # square each deviation
[1] 20.25  0.25  2.25  2.25 20.25  0.25
> sum((x-xbar)^2)               # find the sum of the squared deviations
[1] 45.5
> n <- length(x)                # find the sample size, store as object
> n                             # display the sample size
[1] 6
> variance <- sum((x-xbar)^2)/(n-1)   # calculate the sample variance
> std.dev <- sqrt(variance)           # sq root of variance =  std dev
> std.dev                             # display the answer
[1] 3.016621
```

Answer: the sample standard deviation is $s = 3.02$.

You can accomplish all the above steps much more efficiently by combining the necessary steps in one command (after first creating the data vector, of course):

```
> x <- c(3, 7, 6, 9, 12, 8)
> stddev <- sqrt(sum((x-mean(x))^2)/(length(x)-1))
> stddev
[1] 3.016621
```

And as you might expect, this was all unnecessary, since something as common as the standard deviation has its own built-in function in R, named `sd()`:

```
> x <- c(3, 7, 6, 9, 12, 8)
> sd(x)
[1] 3.016621
```

… but at least you can see how the answer was constructed! ◄

One important lesson to take from this example is that *you can build your own functions in R*. There is where the real power of R lies: it is highly customizable to specific tasks, which makes it quite unlike most pre-packaged software that restricts you to "point-and-click" options. During this semester, we will see a lot of this flexibility in action as we tackle a wide variety of tasks.

Writing your own functions will be introduced in the next lecture.

1.9 Referencing vector elements using indices

Using R to access the elements of a data vector is very straightforward. Each observation x_1, x_2, …, x_n can be referenced by its citing its index using square brackets, as in `x[1],x[2],…, x[n]`.

Below are some illustrations of referencing. The examples use a data vector consisting of weekly average high temperatures in Oxford over the summer. *(Note that the data vector* `temps` *required more than one line to type in due to its length. The line break is denoted in R by +):*

```
> temps <- c(76.1, 79.4, 79.0, 82.5, 83.2, 84.6, 87.0, 84.8, 86.0,
+ 83.3, 88.5)
> temps[2]                  # extracts the 2nd element
[1] 79.4
> temps[length(temps)]      # extracts the last element
[1] 88.5
> temps[1:4]                # extracts the 1st through 4th elements
[1] 76.1 79.4 79.0 82.5
> temps[c(3,4,8)]           # extracts 3rd, 4th, 8th elements
[1] 79.0 82.5 84.8
```

Negative indices in R have the effect of returning all ***but*** the referenced elements:

```
> temps[-2]                 # extracts all but the 2nd element
[1] 76.1 79.0 82.5 83.2 84.6 87.0 84.8 86.0 83.3 88.5
> temps[-(1:4)]             # extracts all but the 1st thru 4th elements
[1] 83.2 84.6 87.0 84.8 86.0 83.3 88.5
> temps[-c(3,4,8)]          # extracts all but the 3rd, 4th, 8th elements
[1] 76.1 79.4 83.2 84.6 87.0 86.0 83.3 88.5
```

You can also assign values to elements of a vector by using index references:

```
> temps
[1] 76.1 79.4 79.0 82.5 83.2 84.6 87.0 84.8 86.0 83.3 88.5
> temps[2] <- 88.8
> temps
[1] 76.1 88.8 79.0 82.5 83.2 84.6 87.0 84.8 86.0 83.3 88.5
> temps[3:5] <- c(95, 100, 105)
> temps
[1]  76.1  88.8  95.0 100.0 105.0  84.6  87.0  84.8  86.0  83.3  88.5
```

1.10 Tips and help with R

You will require assistance when learning and using R. I also need help from time to time, so you are not alone in this!

Help with built-in functions. If you know the name of the function that you want to find out about, just type a question mark, `?`, followed immediately (without a space) by the name of the function. To find out about graphics parameters (`par`), for instance, you would type

```
?par
```

If you do not know the exact name of the function, use the `help.search` facility with the name you are looking for in double quotes, like this:

```
help.search("regression")
```

and a list of the relevant functions involving regression will appear in a pop-up window. You can then use `?` to look up the function names that look most relevant. Be aware that R's on-line help is fairly cryptic, so it may be better to ask me or use a third-party help source (see next section) for additional assistance.

Some tips. These tips will help make your R interface a bit more palatable:

1. **Use a text editor.** Typing in R scripts directly into the command line in the R console can be tedious, especially if you are entering many lines of script. It is easier to use a text editor to "compose" your scripts, and then you can just cut and paste them as a block into the R console. Errors in your scripts are easier to fix and re-submit this way.

   You have two good options for a text editor:

   a. **Use R's built-in text editor**. In R, go to the **File** menu and choose **New script**. A window will open for you to begin composing commands. When ready, cut and paste them into the R console for execution.
   b. **In Windows, use Notepad as your text editor.** *Note:* I do not recommend using Word, because Word's special formatting (and other bells and whistles) can create problems in R.

2. Even if you type scripts directly into the R console, you may retrieve your command history by issuing the command

   ```
   history()
   ```

3. **Use your arrow keys.** Frequently, you will want to re-submit a line of R script (perhaps with minor changes) that you already submitted. Instead of retyping, use your up and down arrow keys to recall earlier scripts. This saves a lot of time!

4. **Workspace management.** When you open an R session, you are creating a *workspace*. The *working directory* is the directory on your computer in which R will by default look for files and save files. The *workspace* is the collection of R objects that are listed upon typing `objects()`. Objects that the user creates are by default stored in the user's workspace.

   When you quit R (see below), you will be asked to save the working directory. If you choose not to save it, you will lose all record of everything you did in the current session (all scripts, etc). So, if you ever intend to return to a project to continue working, it is to your benefit to save the current workspace.

   The default location of a saved workspace is the working directory (in Windows, this is usually your **My Documents** folder). You may want to change this to better organize your work. To do so, go to **File** and choose **Change dir…**.

   To save a workspace:

   - First, remove any objects you no longer want using `rm()`.
   - From the R console menu, choose **File > Save workspace...**
   - Choose a directory from the **Save in:** window, and then meaningfully name the workspace of your current project, i.e. `project1.Rdata`. You need to physically type `.Rdata` as the file type.
   - Click **Save**.

   To load a previously saved workspace:

   - Start R.
   - From the R console menu, choose **File > Load workspace...**
   - Find the .Rdata file you want to load, select it and open it.

5. **Quitting R.** Easy: just click the close box × on the R console window, or type `q()`.

   When you quit R, you will be asked to save the workspace. This saves the workspace into the default working directory (see above). If you do this, this default workspace will be automatically loaded the next time you start R. *This is generally not recommended.*

1.11 Additional references and help sources

These notes are meant to just scratch the surface of R. If you want more detailed introductory treatments of R and its use in basic statistical work, I suggest the following online sources available either as "browseable" HTML or PDF downloads on CRAN:

At `http://cran.r-project.org/manuals.html`:

- **An Introduction to R**: gives an introduction to the language and how to use R for doing statistical analysis and graphics.

At `http://cran.r-project.org/other-docs.html`:

- **Using R for Data Analysis and Graphics - Introduction, Examples and Commentary** by John Maindonald
- **Simple R** by John Verzani
- **IcebreakeR** by Andrew Robinson
- **The R Guide** (version 2.3) by Jason Owen
- **R reference card** by Tom Short

The following are excellent textbook references that use R:

- **Using R for Introductory Statistics** by John Verzani. Chapman and Hall, 2005.
- **Data Analysis and Graphics Using R: An Example-Based Approach, 2nd ed.** by John Maindonald and John Braun. Cambridge University Press, 2007.
- **An R Companion to Applied Regression** by John Fox and Sanford Weisberg. Sage Publications, 2010.

Other valuable websites:

- **Quick-R:** `http://www.statmethods.net/`
  - *Nice introduction with basic code for many tasks. Recommended.*
- **Statistics with R:** `http://zoonek2.free.fr/UNIX/48_R/all.html`
  - *Very comprehensive with lots of explicit script examples.*

Lecture 1 Practice Exercises

Use R to do each of the following. Use R code instructions that are as general as possible, and also as efficient as possible. Use the Quick-R website for help on finding commands.

1. Enter the following values into a data vector named `testdata`:

   45.4 44.2 36.8 35.1 39.0 60.0 47.4 41.1 45.8 35.6

2. Calculate the difference between the 2nd and 7th entries of this vector using only reference indices.

3. Calculate the median of `testdata`.

4. Sort the values in this data vector from highest to lowest, and save the sorted version as a vector named `sortHL.testdata`.

5. What does `sortHL.testdata[-4]` do?

6. What does `testdata[-c(2,7,9)]` do? Why do you think the `c()` function necessary here?

7. Suppose a random variable X has a normal distribution with a mean of $\mu = 60$ and a standard deviation of $\sigma = 4$. Find the following using R functions:

   a. The probability that X is less than or equal to 66.
   b. The probability that X is between 50 and 60.
   c. The probability that X is greater than 68.

8. Refer to 7c. There are two ways to find this probability using R. If you did it one way in part 7c, find another way to do it here. Do a bit of detective work re. options in R for help. You should get the same answer either way.

STA333 Lecture 2
The bare essentials of R (part 2)

2.1 Logical values and expressions

Let's begin with the `temps` data vector in its most recent form from the earlier example:

```
> temps
[1]  76.1  88.8  95.0 100.0 105.0  84.6  87.0  84.8  86.0  83.3  88.5
```

When we interact with R, we ask it a question, and it (hopefully) responds with an answer. Frequently our questions have a numeric answer:

```
> mean(temps)
[1] 89.00909
```

Or, it may be a vector of numeric answers:

```
> temps.celsius <- 5/9*(temps - 32)
> temps.celsius
[1] 24.50000 31.55556 35.00000 37.77778 40.55556 29.22222 30.55556
[8] 29.33333 30.00000 28.50000 31.38889
```

But what if we asked something like "Is a weekly temperature over 85?" The answer to this question isn't a number, but rather a *yes* or *no*. In such cases, we can provide R with a logical expression, and R will check if and where the expression is true or false. R uses the words `TRUE` and `FALSE` to indicate the result. For example,

```
> temps
[1]  76.1  88.8  95.0 100.0 105.0  84.6  87.0  84.8  86.0  83.3  88.5
> temps > 85
[1] FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
```

This second result is an example of a logical vector. The R script `temps > 85` asks of each element of `temps`, "Is the value of the element greater than 85?" Now, the output of `TRUE`'s and `FALSE`'s isn't very useful as displayed, but it can be quite useful when used as the indices of a data vector. When `x` is a data vector and `vec` is a logical vector of the same length as `x`, then `x[vec]` returns *only* the values of `x` for which `vec`'s values are true. These indices can be found using the `which()` function. For example,

```
> temps
[1]  76.1  88.8  95.0 100.0 105.0  84.6  87.0  84.8  86.0  83.3  88.5
> temps[temps > 85]
[1]  88.8  95.0 100.0 105.0  87.0  86.0  88.5
> which(temps > 85)
[1]  2  3  4  5  7  9 11
```

In the example above, the result of `temps > 85` is a logical vector (all elements having values `TRUE` or `FALSE`). Indexing with such a vector picks out the elements for which the index is `TRUE`. The result of `temps[temps > 85]` is just the vector of elements of `temps` whose values exceed 85.

R uses the following operators to do logical comparisons:

>	strictly greater than
>=	greater than or equal to
<	strictly less than
<=	less than or equal to
==	equal to
!=	not equal to

What happens if you apply a numeric function to a logical vector? Some functions can provide very useful results. In such cases, the logical values `TRUE` and `FALSE` are coerced to numeric values of 1 and 0 respectively, i.e. a `TRUE` is interpreted as the value 1 and `FALSE` is interpreted as the value 0.

```
> sum(temps > 85)
[1] 7
```

The `sum()` function coerces the `TRUE`'s and `FALSE`'s into 1's and 0's, and R sums the 1's and 0's to get 7. In other words, there were 7 weeks where the average high temps were over 85° F.

```
 > mean(temps > 85)
[1] 0.6363636
```

The `mean()` function coerces the `TRUE`'s and `FALSE`'s into 1's and 0's, and R averages the 1's and 0's to get 0.636. In other words, this is a proportion: 0.636 (63.6%) of the weeks had an average high temperature over 85° F. *(Question: How can you get R to calculate the mean of only those elements of* `temps` *that are above 85?)*

2.2 Missing values and computer arithmetic in R

Sometimes data from an individual is lost or not available. R indicates such occurrences using the value `NA`. Missing values are important to account for, because we cannot simply ignore them when performing functions that involve all values. For example, we can't find the mean of the elements of a data vector when some are set to `NA`:

```
> x <- c(14, 18, 12, NA, 22, 16)
> mean(x)
[1] NA
```

Many R functions have an argument `na.rm=`, which is set to `FALSE` by default. It can be set to `TRUE` in order to remove the `NA`s. For example,

```
> mean(x, na.rm=TRUE)     # mean of the non-missing elements
[1] 16.4
```

The function `is.na()` can be used to create a logical vector to look for missing values:

```
> x
[1] 14 18 12 NA 22 16
> is.na(x)
[1] FALSE FALSE FALSE  TRUE FALSE FALSE
> x >= 20
[1] FALSE FALSE FALSE    NA  TRUE FALSE
```

The last logical expression `x >= 20` illustrates that if data are missing, even logical comparisons cannot be carried out. In particular, R cannot determine if the missing element of `x` is greater than or equal to 20.

The number system used by R has the following ways of accommodating results of calculations on missing values or extreme values:

`NA`: Not Available. Any data value, numeric or not, can be `NA`. This is what you use for missing data. Always use `NA` for this purpose.

`NaN`: Not a Number. This is a special value that only numeric variables can take. It is the result of an undefined operation like `0/0`.

`Inf`: Infinity. Numeric variables can also take the values `-Inf` and `Inf`. These are produced by the low level arithmetic of all modern computers by operations such as `1/0`. (You shouldn't think of these as *real* infinities, but rather that the correct calculation, if the computer could do it would be *extremely* large, larger than the largest numbers the computer can hold (about 10^{300})).

Scientific notation. This is a shorthand way of displaying very small or very large numbers. For example, if R displays `3e-8` as a value, it really means $3 \times 10^{-8} = 0.00000003$.

2.3 Data frames

Up to now, we have been entering data values into a data vector. While this is fine for entering the values of *n* observations on one variable (e.g. GPA), it does not accommodate data sets with multiple variables, which is more common in practice.

A statistical dataset in R is known as a data frame, and is a rectangular array of information. Rows of a data frame constitute different observations in the data, and the columns constitute different variables measured. For example, here is the data frame `stackloss` from the `datasets` package (which loads automatically upon starting R):

```
> stackloss

   Air.Flow Water.Temp Acid.Conc. stack.loss
1        80          27         89         42
2        80          27         88         37
3        75          25         90         37
4        62          24         87         28
5        62          22         87         18
6        62          23         87         18
7        62          24         93         19
8        62          24         93         20
9        58          23         87         15
10       58          18         80         14
11       58          18         89         14
12       58          17         88         13
13       58          18         82         11
14       58          19         93         12
15       50          18         89          8
16       50          18         86          7
17       50          19         72          8
18       50          19         79          8
19       50          20         80          9
20       56          20         82         15
21       70          20         91         15
```

This data frame contains 21 observations on 4 different variables. The top row containing the variable names is known as the `header` in R.

Issuing the command `data()` lets you see what datasets are currently in the search path.

I will maintain an archive of datasets for STA333 on my public web space on the MU server. The URL is

`http://www.users.muohio.edu/hughesmr/sta333`

2.4 Importing datasets into R: `read.table()`

In practice you will most likely encounter datasets from other external sources. The best format for importing dataset files into R is *tab-delimited text* (file extension .txt). R can handle other formats, but .txt is the easiest to work with.

You may import these data files into R for analysis when needed. There are two ways to accomplish this, both of which require the `read.table()` command

1. **Read a file directly from the web.**
   For example, this command imports the file `univadmissions.txt` from our data repository into the current R session, and assigns the name `uadata` to the data frame object:

```
> site <- "http://www.users.muohio.edu/hughesmr/sta333/univadmissions.txt"
> uadata <- read.table(site, header=TRUE)
> uadata
         id   gpa.endyr1   hs.pct   act   year
  1       1        0.980       61    20   1996
  2       2        1.130       84    20   1996
  3       3        1.250       74    19   1996
  4       4        1.320       95    23   1996
. . . lines deleted . . .
703     703        4.000       97    29   2000
704     704        4.000       97    29   2000
705     705        4.000       99    32   2000
```

2. **First download the file to your computer, then read it.**
   You could go to the URL above and save the data file *to your default working directory* on your computer. This approach also works if I send you a data file as an email attachment. After saving it, type the following into the R console:

```
> uadata <- read.table("univadmissions.txt", header=TRUE)
> uadata
         id   gpa.endyr1   hs.pct   act   year
  1       1        0.980       61    20   1996
  2       2        1.130       84    20   1996
  3       3        1.250       74    19   1996
  4       4        1.320       95    23   1996
. . . lines deleted . . .
703     703        4.000       97    29   2000
704     704        4.000       97    29   2000
705     705        4.000       99    32   2000
```

   If you save the file to a directory on your computer *other* than the default directory, you must provide the path in the `read.table()` command:

```
> uadata <- read.table("c:/My Documents/My STA333 Files/univadmissions.txt",
+      header=TRUE)
```

The function `read.table()` automatically converts a tab delimited text file to a data frame in R. The `header=TRUE` option tells R that the top line in the text file contains the variable names rather than data values.

2.5 Creating your own data frames in R

Two ways to do this, really: either use Excel, or stay native within R. Here are details for each method.

Using Excel. If you are working with your own data, I usually suggest entering it into an Excel spreadsheet. This is the easiest method since Excel is universally available and it is easy to use.

Steps:

1. Enter your data in Excel, putting variable names in the top row according to R naming guidelines. Different variables should be in different columns, and different cases in different rows. Remember that missing data entries should be indicated by `NA`.

2. Save your Excel file as a tab-delimited text file using the following menu choice:

   File > Save As...

3. From the Save as type options, choose Text (Tab delimited). Save the file to the desired directory on your computer.

4. Once you are back in R, use the `read.table()` command to import your text file (see directions in section 2.4).

Interactive data frame entry in R. R also has a built-in spreadsheet that you can use to enter data frames interactively, bypassing the need for Excel altogether. To access it, you must use the `data.frame()` and `edit()` commands in R.

For example, here is how you could set up a data frame directly in R's spreadsheet utility. This example sets up a data frame with three variables: `id`, `sex`, and `gpa`. The `sex` variable is specified to be a character variable, while the other two are set to be numeric:

```
> mydata <- data.frame(id=numeric(0), sex=character(0), gpa=numeric(0))
> mydata <- edit(mydata)
```

Try this example and see what happens.

2.6 Referencing data in a data frame

We can access parts of a data frame using indices much like we did for data vectors. The key difference is that data frames are two-dimensional (i.e. they have rows and columns), so each element has a `[row, col]` index pair. We can access the entry in the i^{th} row and j^{th} column of a data frame x by calling `x[i,j]`.

The examples below use the data frame `uadata` from section 2.4. First, we can check the dimensions of the data frame. `uadata` has 705 rows and 5 columns:

```
> dim(uadata)
[1] 705   5
```

We can list the names of the variables in the data frame:

```
> names(uadata)
[1] "id"          "gpa.endyr1" "hs.pct"     "act"         "year"
```

Important: To make variables in a data frame accessible by name within the R session, you must first invoke the `attach()` function:

```
> attach(uadata)
```

Here are several examples of selecting and accessing data in the data frame. Try these out yourself to see what happens. Some results are printed here.

```
> uadata[2,]           # select the second row only

> uadata[,4]           # select the fourth column only

> uadata[63,4]         # selects the entry in the 63rd row, 4th column
> uadata[c(1,2,7),]    # selects rows 1, 2, and 7
  id gpa.endyr1 hs.pct act year
1  1       0.98     61  20 1996
2  2       1.13     84  20 1996
7  7       1.67     67  28 1996

> uadata[80:88,]       # selects rows 80 through 88
   id gpa.endyr1 hs.pct act year
80 80       3.14     59  24 1996
81 81       3.14     76  24 1996
82 82       3.15     77  20 1996
83 83       3.15     73  25 1996
84 84       3.15     95  25 1996
85 85       3.16     55  22 1996
86 86       3.16     97  29 1996
87 87       3.19     45  20 1996
88 88       3.19     88  22 1996
```

We can still use "–" to indicate "everything but":

```
> uadata[,-c(1,2)]      # select all data except columns 1 and 2
    hs.pct act year
1       61  20 1996
2       84  20 1996
3       74  19 1996
4       95  23 1996
... stuff deleted ...
703     97  29 2000
704     97  29 2000
705     99  32 2000
```

You may extract a variable from a data frame using the call `dfname$varname`: For example, we can extract the variable `act` from the data frame and place it into a vector of its own:

```
> actvec <- uadata$act
> actvec
  [1] 20 20 19 23 28 23 28 20 22 24 27 21 23 27 21 23 20 19 20 29 27 28 24 21 27
 [26] 19 24 25 20 28 25 32 20 29 25 23 23 26 21 22 23 29 21 24 29 22 23 25 22 25
 [51] 21 28 18 21 21 23 24 24 28 17 21 24 25 30 22 29 23 23 24 22 31 26 18 20 22
 [76] 24 23 27 27 24 24 20 25 25 22 29 20 22 27 17 25 26 22 27 31 25 25 26 31 17
... stuff deleted ...
[601] 23 31 17 25 16 31 18 26 22 28 24 18 24 22 24 30 25 18 26 19 25 18 26 28 28
[626] 28 28 24 25 21 22 26 22 23 25 25 28 28 22 28 31 22 18 30 28 26 28 22 27 24
[651] 26 28 24 20 20 22 29 20 24 26 25 23 22 28 24 28 25 22 30 30 27 25 22 23 23
[676] 24 25 26 27 25 26 30 25 29 31 21 27 26 23 26 28 30 26 26 29 21 24 27 27 27
[701] 29 26 29 29 32
```

We may also want to select the subsets on the basis of some criterion, e.g. which cases have a freshman end-of-year GPA greater than 3.9.

```
> attach(uadata)
> uadata[gpa.endyr1 > 3.9,]
     id gpa.endyr1 hs.pct act year
138 138      3.910     99  28 1996
139 139      3.920     93  28 1996
140 140      4.000     99  25 1996
141 141      4.000     92  28 1996
142 142      4.000     99  33 1996
266 266      3.920     85  15 1997
267 267      3.920     63  23 1997
268 268      3.920     98  23 1997
269 269      3.930     99  29 1997
270 270      3.960     99  29 1997
271 271      3.970     92  26 1997
272 272      4.000     99  20 1997
273 273      4.000     92  29 1997
422 422      3.920     77  28 1998
423 423      3.920     97  31 1998
424 424      3.930     82  31 1998
425 425      3.960     69  21 1998
426 426      4.000     97  27 1998
427 427      4.000     99  32 1998
565 565      3.913     45  23 1999
```

2.7 Writing your own functions

I've already mentioned a few of the built-in R functions. There are gazillions of them. By built-in functions, I mean those that you don't have to do anything special to use (e.g. calling `mean(x)` directly calculates the mean of the vector `x`).

Strictly speaking, R doesn't really have any "built-in" functions. In truth, they are just functions that someone wrote up that were placed in the software's core code. In this sense, *any function is like any other function*, whether someone else wrote it or you wrote it. If you have a specific task you'd like R to perform for you, you can write your own R function to do it.

You can "customize" the software by programming your own tasks into an R function. A function basically serves to bundle a set of commands together so that you may execute them later with a single function call.

The basic syntax for a function is

```
function(argument-list) {
     expressions
     return(value)
}
```

This may look strange, so let's try a simple illustration.

Defining Functions. The R function `function` defines new functions. For example,

```
> trim <- function(x, lower = 0.0, upper = 1.0) {
+              indicies <- x >= lower & x <= upper
+              return(x[indicies])
+ }
```

creates a new function named `trim` that trims off the values of the argument `x` that are below or above the arguments `lower` and `upper`, respectively. The `lower = 0.0` and `upper = 1.0` in the definition specify *default values* for these arguments that are used when the user does not supply values. The `&` is a logical operator that requires both conditions to be met. Let's check it out.

```
> trim <- function(x, lower = 0.0, upper = 1.0) {
+            indicies <- x >= lower & x <= upper
+            return(x[indicies])
+ }
>
> x <- sort(rnorm(20))
> x
> y <- trim(x)
> y
> z <- trim(x, - 1/2, 1/2)
> z
```

As the assignment suggests, an R function is just an R object like any other. In this example, `trim` is an R object that happens to be a function and `x` is an R object that happens to be a

numeric vector. This allows functions to be passed as arguments to other functions, a very useful technique that we will use often (that's the main reason we often want to define our own functions).

Returned Value: A `return` statement is not strictly necessary. Functions return the value of the last expression if there is no return statement. The curly brackets are not necessary if there is only one statement. Thus,

```
> trim <- function(x, lower = 0.0, upper = 1.0) x[x >= lower & x <= upper]
```

works just as well as the other definition (but it might be harder to read).

Local Variables: *Local variables* are variables defined inside a function. They exist only inside the function and have no influence on anything outside the function. The following example shows this behavior. Run it and see what you get:

```
> x <- 42
> fred <- function(y) {
+    x <- y
+    return(y + x)
+ }
> fred(13)
> x
```

Inside the function `x` is defined to be the value of `y`, but outside the function `x` is unchanged.

Global Variables: *Global variables* are variables defined outside a function. They are not defined inside the function, either in the argument list or in the body. They can, however, be used inside the function:

```
> x <- 42
> fred <- function(y) {
+    return(y + x)
+ }
> fred(13)
```

This is sometimes very convenient, but can lead to confusing code.

► **Example:** Create a function in R that "rolls" a fair die *n* times and returns to total number of dots from all the rolls.

We only need one argument here: `n`, the number of rolls. This will be variable and must be specified by the user when the function is called to execute. Here is the function, which I've assigned to an R object named `sumdice` (that is, I've named the function `sumdice`):

```
sumdice <- function(n) {
     k <- sample(1:6, size=n, replace=TRUE)
     return(sum(k))
}
```

Explanation. Before we move on to execute the function, I should talk a little about the built-in R function `sample()`, which I use as a "function inside my function". `sample()` is an R function that lets you sample items from a vector, a data frame, or as in this case, just an arbitrary set of integers (1 through 6, representing the die faces). You specify the number of items to sample, as well as whether or not you wish to sample with or without replacement.

The command `k <- sample(1:6, size=n, replace=TRUE)` draws a sample of size `n` from the integers 1 through 6 with replacement (meaning it is possible to draw the same value more than once) and assigns the values to the object `k`. `k` will be a data vector of length `n`. The values in `k` are then totaled using the `sum()` function.

To illustrate, I type in the function and then execute it four times:

```
 > sumdice <- function(n) {
+      k <- sample(1:6, size=n, replace=TRUE)
+      return(sum(k))
+ }
> sumdice(2)          # roll a pair of die (i.e., n=2)
[1] 4
> sumdice(2)          # roll again (random chance makes a different result)
[1] 10
> sumdice(50)         # roll 50 die
[1] 174
> wow <- sumdice(50)  # save the sum of 50 rolls into an R object named "wow"
> wow                 # call up "wow"
[1] 185
```

Functions can be very elaborate and elegant, and we will use them in STA333 to illustrate some concepts in nonparametric statistics as the semester progresses. ◀

Question: *On average*, how many dots would you expect to see when you roll a fair six-sided die? How can you use your function `sumdice` to get a reliable estimate of the average ("expected") number of dots?

2.8 Looping control structures

Those who find computer programming frightening may rest assured that the programming we do will be very simple, involving no more than writing your own functions and perhaps using the control structure described in this section.

One thing computers are much better at than people is mindless repetition. The `for` looping control construct is the main way mindless repetition is done in R. For example, the R expression

```
for (i in 1:100) {
    insert some R statements that do some work here
}
```

does the same thing (whatever is done by the R statements inside) 100 times.

► **Example.** Suppose we want to examine the sample median as an estimator of the mean of a normal distribution. The following code simulates the median of a random sample of size `n` from a standard normal distribution, and we do this repeatedly `nsim` times.

```
> n <- 20
> nsim <- 1000
> lotsa.medians <- numeric(nsim)
> for (i in 1:nsim) {
+    x <- rnorm(n)
+    lotsa.medians[i] <- median(x)
+ }
> lotsa.medians
> mean(lotsa.medians)
> hist(lotsa.medians)
```

Explanation:

- The statement `lotsa.medians <- numeric(nsim)` creates an "empty" vector of length `nsim` to hold the simulation results (that we have not done yet). Each time the `for` loop repeats, it calculates one result (i.e. one median from a random sample of size `n`).
- The statement `x <- rnorm(n)` generates a random sample of size `n` from the standard normal $N(0, 1)$ distribution.
- The expression `median(x)` calculates the median of the sample just simulated, and the whole statement `lotsa.medians[i] <- median(x)` assigns this calculation to the `i`th element of the vector `lotsa.medians`. To understand this, one needs to know that each time the loop is executed, the variable `i` takes a different value from the list specified in the `for` statement, which in this case is `1:nsim`, the vector of integers 1, 2, …, `nsim`.
- When the loop finishes the vector `lotsa.medians` contains `nsim` random (and independent) replications of the sample median of a sample of size `n`. The last three commands print out the 1000 randomly generated sample medians, calculates their average, and then draws a histogram of them.

Try this code out and see what you get. ◄

Lecture 2 Practice Exercises

Use R to do each of the following. Use R code instructions that are as general as possible, and also as efficient as possible.

1. Use the `testdata` data vector you created for the Lecture 1 practice problems:

   45.4 44.2 36.8 35.1 39.0 60.0 47.4 41.1 45.8 35.6

   a. Use logical referencing to calculate the standard deviation of only those values that are less than 45.
   b. Write an R command that will determine how many of the vector entries are less than 45.
   c. Write an R command that will determine how many vector entries are greater than 40 but less than 55 (i.e. how many entries are between 40 and 55).
   d. Write an R command that will calculate what proportion of the data vector has values exceeding 40.

2. This exercise uses an existing R data frame called `cigsales` that contains state-level data regarding cigarette sales and other variables. Go to our data repository to open the workspace. See INSR section 2.6 for guidance on this exercise.

   a. The variable `black` indicates the percentage of a given state that is African-American. Using logical referencing, select from this data frame only those states that have over a 15% African-American population. What states get selected?
   b. Extract the variable `price` from this data frame, and place it into a vector of its own called `price.vec`.
   c. Use logical referencing to create two separate vectors in R:
      a. A vector called `poor` containing only the `income` values that fall below the median `income` value for all the states; and
      b. A vector called `rich` containing only the `income` values that fall above the median `income` value for all the states.

3. I want you to run all the R code examples in INSR section 2.7 and carefully examine what each example does. Please contact me (phone/email/visit) if you have questions.

4. Create an R function named `diff` that calculates the absolute difference between the mean and median of a sample of values stored in a vector. Then, run your function on the `price.vec` vector from part 2b.

5. Here is a very simple loop in R. Run this code and tell me what it does:

```
for (i in 1:20) {
  print(i+3)
}
```

STA333 Lecture 3
Review of some basic statistical concepts

3.1 Goals of a statistical analysis

Statistics can be broadly defined as the study of variation in data.

We collect sample data from a parent population of interest typically because, even though we are ultimately interested in characteristics about the population, it is not feasible to collect data from every member. Good statistical practice demands that we collect sample data from the population in such a manner so as to ensure that the sample is *representative*, i.e., it is not presenting a systematic bias in its representation.

There are two issues that arise with respect to variation in such data:

1. Measurements of the same attribute will vary from sample member to sample member. This is natural and is to be expected.

2. Any summary characteristic computed from the sample (known as a *sample statistic*) will only be an estimate of the corresponding summary characteristic for the population as a whole (known as a *population parameter*), since only a small subset of the population is used in its calculation. Thus, sample statistics will vary from their corresponding population parameters. In fact, different samples from the same population will produce different estimates of the same parameter, so sample statistics also vary from sample to sample.

This variation gives rise to the concept of *uncertainty* in any findings that we make about a population based on sample data. A meaningful statistical analysis helps us quantify this uncertainty.

Statistics starts with the formulation of a problem, continues with the collection of data, proceeds with a data analysis, and then finishes with conclusions. To formulate a problem correctly, you must:

1. **Understand something about the background.** Statisticians often work in collaboration with researchers in other disciplines and need to understand something about the subject area.

2. **Understand the objectives.** Run a focused analysis based on clearly stated objectives. Beware of "fishing expeditions": if you look hard enough, you'll almost always find something, but that something may just be a coincidence of no consequence. In some cases, you may find that simple descriptive statistics are all that are needed.

3. **Put the problem into statistical terms.** This can be the most challenging step and where harmful errors are often made. How do you translate the original research question into one that can be addressed using a statistical method?

The mere fact that a statistical method can be employed is not enough. The results may be totally meaningless if they are addressing the wrong question. So, some knowledge of the context of the problem is crucial to fully understanding results of an analysis.

Goals of a statistical analysis

- To make sense of data in the face of uncertainty.
- To be able to make reliable inferences/generalizations about the parent population(s) from which your sample(s) is/are drawn.

3.2 Descriptive statistics ("preliminary" data analysis)

Looking at some descriptive sample statistics is an important step that should be performed prior to a formal analysis. It looks simple but it is vital. Descriptive statistics are comprised of:

- **Numerical summaries** (means, medians, SDs, five-number summaries)
- **Graphical summaries**
  - *One variable:* boxplots, histograms, etc.
  - *Two variables:* scatterplots
  - *Many variables:* scatterplot matrices, interactive graphics

Look for outliers, data-entry errors and skewed or unusual distributions. Are the data distributed as you expect? Getting data into a form suitable for analysis by cleaning out mistakes and aberrations is often time consuming. It often takes more time than the data analysis itself! Let's look at an example.

► **Example: University admissions.** The director of admissions at a state university wanted to determine how accurately students' grade point averages at the end of their freshman year could be predicted by entrance test scores and high school class rank. The academic years covered 1996 through 2000. This example uses the `univadmissions.txt` dataset from our repository. The variables in the dataset are as follows:

`id`	Identification number
`gpa.endyr1`	Grade point average following freshman year
`hs.pct`	High school class rank (as a percentile)
`act`	ACT entrance exam score
`year`	Calendar year the freshman entered university

I will presume you have read in the data to an R data frame named `uadata`. Many basic numeric summaries can be obtained through the generic `summary()` function:

```
> summary(uadata)

       id         gpa.endyr1        hs.pct           act
 Min.   :  1   Min.   :0.5100   Min.   : 4.000   Min.   :13.000
 1st Qu.:177   1st Qu.:2.6090   1st Qu.:68.000   1st Qu.:22.000
 Median :353   Median :3.0500   Median :81.000   Median :25.000
 Mean   :353   Mean   :2.9773   Mean   :76.953   Mean   :24.543
 3rd Qu.:529   3rd Qu.:3.4700   3rd Qu.:92.000   3rd Qu.:28.000
 Max.   :705   Max.   :4.0000   Max.   :99.000   Max.   :35.000
      year
 Min.   :1996
 1st Qu.:1997
 Median :1998
 Mean   :1998
 3rd Qu.:1999
 Max.   :2000
```

Presented here are the five number summary (minimum, 1st quartile, median, 3rd quartile, and maximum) and the mean for each numeric variable in the data frame. For example, the mean freshman year-end GPA over the period 1996-2000 was 2.977; the median over the same period was 3.05, suggesting the GPA distribution is skewed left (not surprising). The middle half of the GPA distribution ranged from 2.609 to 3.470. The median ACT score for entering freshman was 25, and one-fourth of then scored 28 or higher.

This display introduces two points you need to be aware of:

- Since we applied `summary()` to the entire data frame, we also got a "summary" of the variable `id`, even though it is meaningless to do so. [*Moral:* just because a computer spits it out at you, that doesn't mean it's useful.]

- If you look carefully, you'll see that `year` is (of course) entered as a number (e.g. 1996, 1997, etc.). However, that doesn't necessarily mean it should be treated as a numeric variable! It is really meaningless to find numeric summaries like means, medians, etc, for `year`, because it is really a qualitative (categorical) variable in the context of these data. [*Moral:* just because a computer spits it out at you, that doesn't mean it's correct.]

  This is a common mistake made by the uninitiated. It's as simple as this: when R encounters a variable whose values are entered as numbers in a data frame, R will treat the variable as if it were a numeric variable. If R encounters a variable whose values are character strings, R will treat the variable as a categorical factor. But, in a case like `year` above, you have to explicitly tell R that the variable is really a factor, as follows:

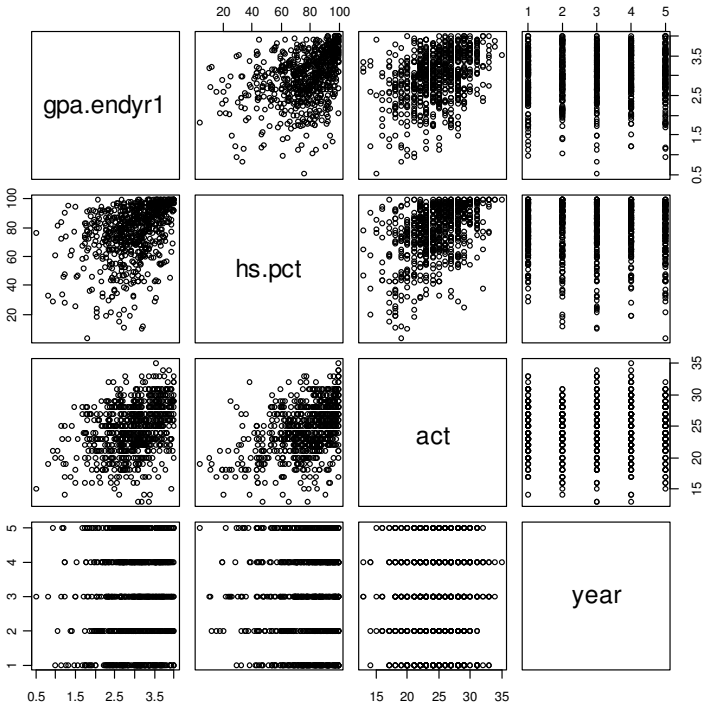
```
> uadata$year <- factor(uadata$year)
> summary(uadata)
```

```
       id         gpa.endyr1        hs.pct           act          year
 Min.   :  1   Min.   :0.510   Min.   : 4.00   Min.   :13.00   1996:142
 1st Qu.:177   1st Qu.:2.609   1st Qu.:68.00   1st Qu.:22.00   1997:131
 Median :353   Median :3.050   Median :81.00   Median :25.00   1998:154
 Mean   :353   Mean   :2.977   Mean   :76.95   Mean   :24.54   1999:141
 3rd Qu.:529   3rd Qu.:3.470   3rd Qu.:92.00   3rd Qu.:28.00   2000:137
 Max.   :705   Max.   :4.000   Max.   :99.00   Max.   :35.00
```

  Now the summary of year consists of frequency counts instead of numeric summary statistics. The number of sampled students in 1996 was 142; 131 in 1997, etc.

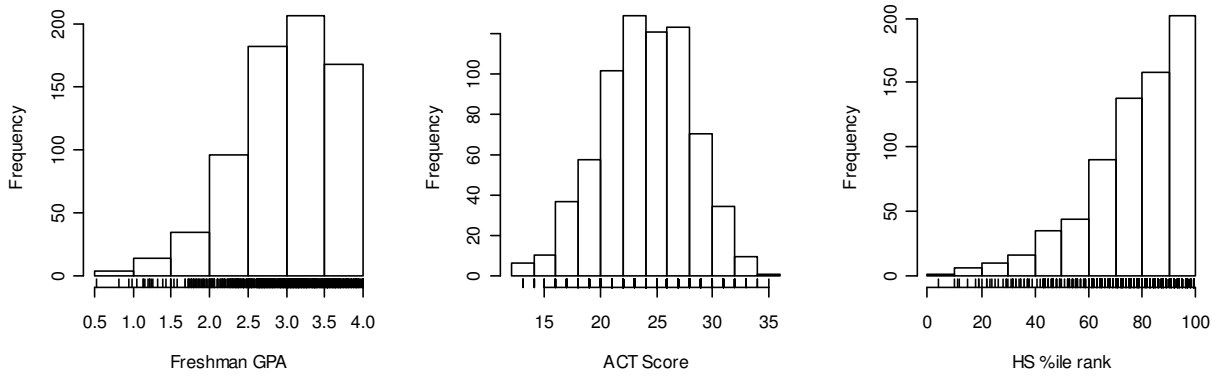
The `pairs()` function applied to a data frame will return a **scatterplot matrix**, displaying the bivariate relationships among the chosen variables:

```
> pairs(uadata[,2:4])
```



This lets us have a glimpse at any relationships or associations between the variables in the data frame. We can also look at univariate distributional shapes by looking at **histograms**:

```
> par(mfrow=c(1,3))    # set up a plotting window for 3 side-by-side plots
> attach(uadata)       # necessary to call up variable names alone
> hist(gpa.endyr1, main=" ", xlab="Freshman GPA")
> rug(uadata$gpa.endyr1)
> hist(act, main=" ", xlab="ACT Score")
> rug(uadata$act)
> hist(hs.pct, main=" ", xlab="HS %ile rank")
> rug(uadata$hs.pct)
> par(mfrow=c(1,1))    # reset plotting window to default (1 plot per window)
```



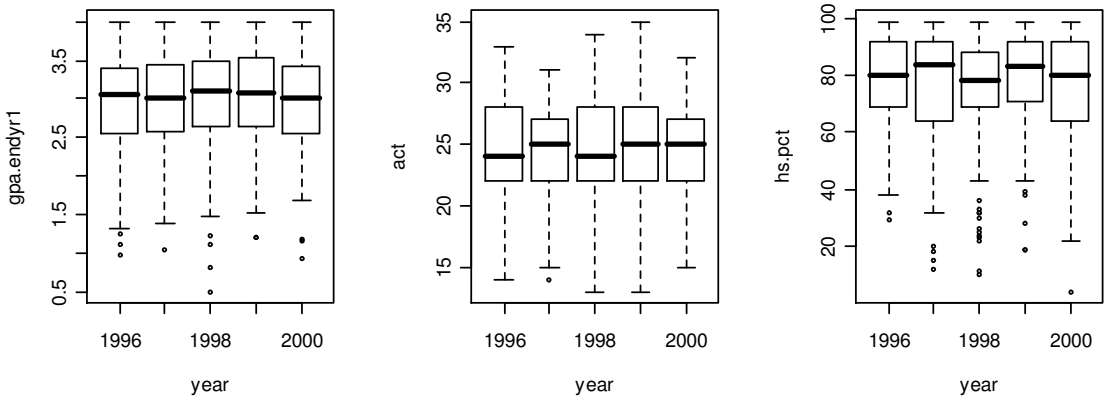
We can see that GPA and high school percentile rankings are skewed left, and that ACT scores are reasonably symmetrically distributed. `par(mfrow=c(1,3))` is a command that sets graphic output parameters so that multiple graph requests may appear in a single display window; in this case, graphs are paneled in 1 row and 3 columns. The `rug()` function places tick marks at each data value on the horizontal axis. The final `par(mfrow=c(1,1))` restores the default layout.

Descriptive statistics and visualizations by levels of a factor variable

While the above summaries are informative, they overlook the fact that we have several years of data and that it may be more interesting to see how things might be changing year to year. This is why treating `year` as a factor is important, because it enables us to split out descriptions of the numeric variables by levels of the factor.

The best visualization tool for comparing distributions is the **boxplot**, which displays information about quartiles and potential outliers. Here we construct side-by-side boxplots by year for each of the three numeric variables in the data frame:

```
> par(mfrow=c(1,3))           # set 3 plots horizontally in 1 row
> boxplot(gpa.endyr1 ~ year)
> boxplot(act ~ year)
> boxplot(hs.pct ~ year)
> par(mfrow=c(1,1))           # reset default graphic layout
```



This is a nice compact visual description of the changes in patterns of response of these three variables over time. All three variables appear pretty stable over the years; the left skews in GPA and high school percentile rankings are still evident here. Note the detection of some low outliers for `year` on these two variables.

Aside: The following command provides summary descriptive statistics by `year`, but using the `summary()` function in the base R package:

```
> by(uadata[,2:4], year, summary)
```

3.3 The nuts and bolts of variability

At the beginning of this topic, we defined statistics as the study of variation in data. The truth is that everything naturally varies:

- If you measured the same characteristic on two different individuals, you'd get two different answers (*subject variability*).
- If you measured the same characteristic twice on the same individual, you'd get two different answers (*measurement error*).
- If you measured the same characteristic on the same individual but at different times, you'd get two different answers (*temporal variability*).

Because everything varies, finding that things vary is simply not very interesting. What we need is a way of discriminating between [1] variation that is scientifically interesting, and [2] variation that just reflects the natural background noise that is always there. This is what the science of statistics is for.

The key concept is knowing what amount of variation we would expect to occur just by random chance alone, when nothing scientifically interesting is going on. If we actually observe bigger differences than we would expect by chance alone, we say the result is *statistically significant*. If instead we observe differences no larger than we would expect by chance alone, we say the result is *not statistically significant*.

Standard deviations and standard errors

A standard deviation (SD) is a measure of how a quantity varies. You may think of it (roughly) as the average distance that the values in a sample or a distribution of values are away from the mean of the sample (or distribution). The standard deviation of a sample estimate is called the standard error (SE) of the estimate:

$$\text{Sample standard deviation:} \quad s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

$$\text{Standard error of } \bar{x}: \quad \text{SE}_{\bar{x}} = \frac{s}{\sqrt{n}}$$

When considering how a statistic (like $\bar{x}$) varies around the parameter it is estimating (μ), it makes sense that the more data you build the statistic with, the better it should perform as an estimator. This is precisely why the SE of $\bar{x}$ is inversely related to the sample size. As n increases, the variation in the estimate decreases, i.e. the estimate will be closer to the thing it is estimating.

3.4 Sampling distributions

It is not enough to know just the standard deviation of an estimate. That is just a measure of the inconsistency in an estimate's value from sample to sample of a given size n.

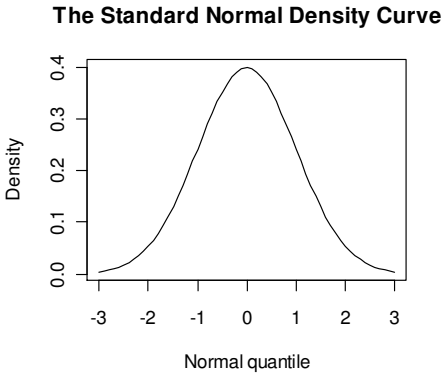
If we wish to make a *confident* inference about a population parameter using a sample statistic (e.g. $\bar{x}$), we also need to know something about the general pattern of behavior that the statistic's value would take on if we had observed it over many, many different potential random samples. This long-run behavior pattern is described via the probability distribution of the statistic, also known as the **sampling distribution of the statistic**.

It is important to remember that we usually only collect one sample, so only one sample mean $\bar{x}$ will be available in practice. However, knowledge of the sampling distribution of our statistic or estimate is important because it will enable us to assess the *reliability* or *confidence* we can place in any generalizations we make about the population using our estimate.

Two common sampling distributions: Normal and *t*

You should be familiar with normal distributions and *t*-distributions from your introductory statistics course. In particular, both serve as useful sampling distributions to describe the behavior of the sample mean $\bar{x}$ as an estimator for a population mean μ. We briefly review them here, and illustrate the use of R functions to replace old-fashioned normal tables and *t*-tables.

Normal distributions. A variable X that is normally distributed is one that has a symmetric bell-shaped density curve. A normal density curve has two parameters: its mean μ and its standard deviation σ. Notationally, we indicate that X is normal by writing $X \sim N(\mu, \sigma)$.



At right is a picture of the *standard normal curve*, which is the normal curve with μ = 0 and σ = 1, i.e., $N(0, 1)$.

Probabilities. `pnorm(q,μ,σ)` is the R function that calculates the probability $P(X \le q)$, where $X \sim N(\mu, \sigma)$. In other words, it returns the area to the left of `q` under the normal curve with mean μ and SD σ. The value `q` is a quantile of the distribution:

```
> pnorm(1.96, 0, 1)       # find area to left of 1.96 on N(0,1)
[1] 0.9750021
> pnorm(25, 33, 4)        # find area to left of 25 on N(33,4)
[1] 0.02275013
```

Quantiles. `qnorm(p,μ,σ)` is the R function that returns the quantile q from $N(\mu, \sigma)$ such that $P(X \le q) = p$. In other words, it finds the location of the slice `q` through the normal curve with mean μ and SD σ such that the area to the left of the slice is `p`:

```
> qnorm(0.975, 0, 1)      # find value with area to L of 0.975 on N(0,1)
[1] 1.959964
> qnorm(0.02275, 33, 4)
[1] 24.99999
> qnorm(0.025, 13.3, 2.5)
[1] 8.40009
```

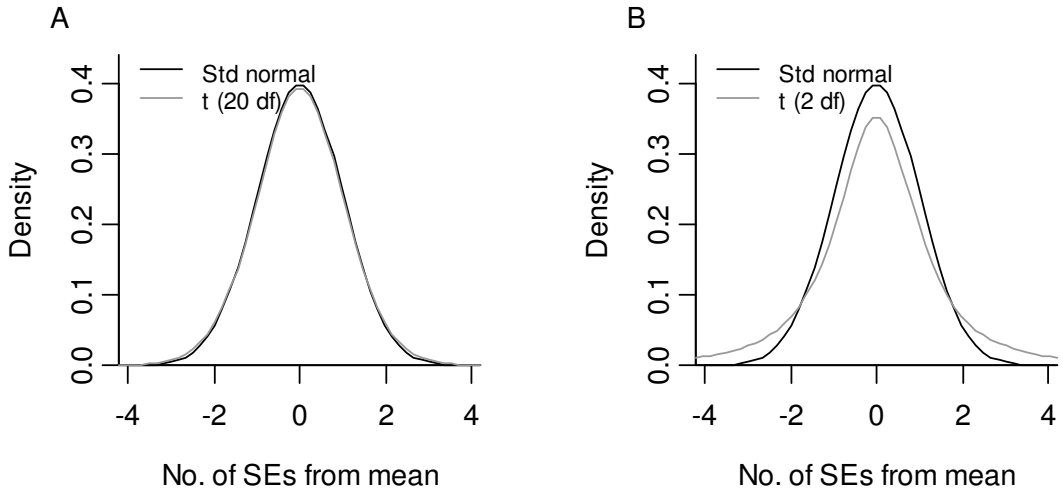
The normal distribution serves as a good sampling distribution model for the sample mean $\bar{x}$ because of the powerful **Central Limit Theorem**, which states that if n is sufficiently large, the sampling distribution of $\bar{x}$ will be approximately normal, regardless of the shape of the distribution of the original parent population from which you collect your sample.

***t* distributions.** The formula given by

$$t = \frac{\bar{x} - \mu}{\text{SE}_{\bar{x}}}$$

counts up the number of standard error units between the true value of μ and its sample estimate $\bar{x}$; it can be thought of as the standardized distance between a true mean value and its sample mean estimate. If the original population was normally distributed, then the quantity t will possess the t-distribution with $n - 1$ degrees of freedom. *Degrees of freedom* is the only parameter of a t-distribution. Notationally, we indicate this by writing $t \sim t(\text{df})$.

t-distributions look a lot like the standard normal distribution $N(0, 1)$: they both center at 0 and are symmetric and bell-shaped. The key difference is due to the fact that the quantity t includes two sample summaries in its construction: the sample mean $\bar{x}$, and $\text{SE}_{\bar{x}}$, which incorporates the sample standard deviation s. Because of this, t is more variable than $N(0, 1)$. However, as n (and hence degrees of freedom) increases, s becomes a better estimate of the true population SD, so the distinction ultimately vanishes with larger samples. This is shown in the figure below:



Plot A superimposes $N(0, 1)$ over $t(20)$, while plot B superimposes $N(0, 1)$ over $t(2)$.

From introductory statistics, you are probably familiar with looking up values in a *t*-table. With R, using such tables is a thing of the past:

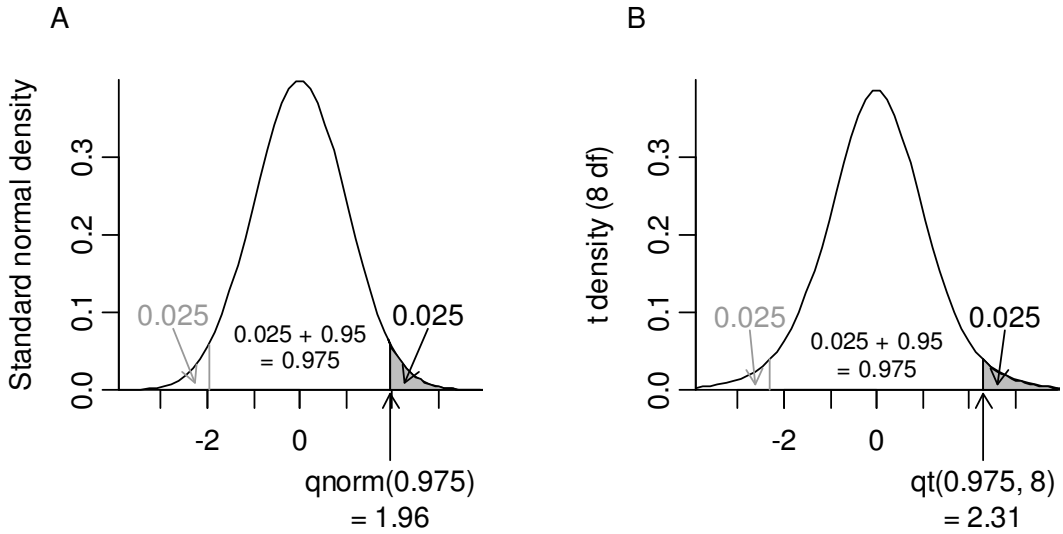
Probabilities. `pt(q,df)` is the R function that calculates the probability $P(t \leq q)$, where $t \sim t(\text{df})$. In other words, it returns the area to the left of `q` under the *t*-curve with degrees of freedom given by `df`. Check these examples against a traditional *t*-table:

```
> pt(1.372, 10)          # area to L of 1.372 on t-curve w/ df = 10
[1] 0.8999723
> 1 - pt(2.145, 14)      # area to R of 2.145 on t-curve w/ df = 14
[1] 0.02499008
> 1 - pt(2.048, 28)      # area to R of 2.048 on t-curve w/ df = 28
[1] 0.02502126
```

Quantiles. `qt(p,df)` is the R function that returns the quantile q from $t(\text{df})$ such that $P(t \leq q) = p$. In other words, it finds the location of the slice `q` through the *t*-curve with degrees of freedom given by `df` such that the area to the left of the slice is `p`:

```
> qt(0.10, 10)       # slice through t(10) such that area to L is 0.1
[1] -1.372184
> qt(0.90, 10)       # slice through t(10) such that area to L is 0.9
[1] 1.372184
> qt(0.5883, 17)     # slice through t(17) such that area to L is 0.5883
[1] 0.2266485
```

If the notion of quantiles is still a bit confusing to you, consider the following figure:



Both plots are finding the quantile value on a sampling distribution that places an area of 0.975 to its left. This quantile is also known as the **97.5th percentile** of its respective distribution. For the $N(0, 1)$ distribution (plot A), this quantile value is 1.96; for the $t(8)$ distribution (plot B), this quantile value is 2.31. The same quantile for the *t*-curve is farther out because the *t*-curve is more spread out.

3.5 Confidence intervals (CIs)

A **confidence interval** is the tool that researchers use to ***estimate*** **unknown population parameters.** A confidence interval is some interval of values within which we are highly confident the true value of some population parameter resides.

Typically, we first use a sample statistic as a *point estimate* of the unknown parameter, and because this estimate has uncertainty, we build a formal estimate of the parameter by forming an interval around the point estimate that accounts for the *margin of error* that the point estimate has. The *confidence level* for a CI sets the likelihood (prior to sampling) that the interval will cover the true parameter value, and is determined by the researcher. The general form of any traditional confidence interval is given by

point estimate ± margin of error

One-sample *t*-based CI for a mean μ

If the variable being studied has a normal distribution, recall that we can form a 95% confidence interval for μ by using the formula

$$\bar{x} \pm t_{0.025} \times \text{SE}_{\bar{x}}, \quad \text{or equivalently,} \quad \bar{x} \pm t_{0.025} \frac{s}{\sqrt{n}}$$

where $t_{0.025}$ is conventional notation for the quantile of the *t*-distribution with $n - 1$ df that puts an area of 0.025 to its right. In R, this is `qt(0.975,n-1)`. Time for an example:

► **Example: Honda Civic mileage.** A random sample of $n = 35$ 2008 Honda Civic sedans were used in testing for average fuel mileage in city driving. Each car was filled three times, and the total miles driven on the three tankfulls was recorded, then divided by the total gallons of fuel consumed to obtain their MPG. The data appear in the file `civicmpg.txt` as follows:

25.32	27.97	25.69	30.93	28.79	33.64	29.93	27.06	27.56	30.58	23.89	29.24
26.55	27.86	31.68	29.43	27.24	29.95	26.77	25.16	26.23	27.59	31.93	27.24
30.50	25.58	32.30	33.13	25.74	25.16	26.61	23.85	24.08	30.72	30.25	

Find a 95% CI for μ, the true mean MPG for 2008 Honda Civic sedans.

```
> attach(civicmpg)
> xbar <-mean(mpg)
> se.xbar <- sd(mpg)/sqrt(length(mpg))
> t.multiplier <- qt(0.975,length(mpg)-1)
> CI <- xbar + c(-t.multiplier*se.xbar, t.multiplier*se.xbar)
> CI
[1] 27.24 29.11
```

We can be 95% confident that the true mean mileage is between 27.24 and 29.11 miles per gallon. (*Note*: There may be assumptions required to validate this analysis.) ◄

3.6 Hypothesis tests: the basic ingredients

Hypothesis tests are the means by which statisticians test conjectured statements about unknown population parameters. We use a sample statistic as a *point estimate* of the unknown parameter, and then use a probability argument to determine whether or not to believe some conjectured hypothesis made about the true parameter value.

There are three essential ingredients required for *any* hypothesis test:

1. A hypothesis to be tested (called the **null hypothesis**, labeled H_0)
2. Some statistic calculated from a collected sample that is useful for testing the hypothesis (called the **test statistic** for the test)
3. Knowledge of the expected random behavior of the statistic if H_0 were actually true (called the **null distribution** of the test statistic)

It's probably wise at this point to flesh out some general issues in hypothesis testing.

Hypotheses

What is a "good" hypothesis? Testing a hypothesis means seeing if the hypothesis can stand up to scrutiny. If the hypothesis doesn't stand up to scrutiny, then it should be rejected. **Thus, a good hypothesis is a falsifiable hypothesis.** To illustrate, consider the following two assertions:

1. There are WMDs (weapons of mass destruction) in Iraq.
2. There are no WMDs in Iraq.

Both statements involve the same essential idea, but one is refutable and the other is not. Ask yourself how you would refute option 1. You travel through Iraq and you look for WMDs, but you don't find any. Of course, this doesn't mean that there *aren't* any. They could be very well hidden. They could be buried underground. They could be constantly moved to other locations when you aren't looking. So no matter how long or how hard you look, you *cannot* refute the hypothesis. All you can do is say: "I went out and I didn't find any WMDs". This situation illustrates a most important scientific principle: absence of evidence is not evidence of absence.

Option 2, however, is fundamentally different. You reject hypothesis 2 the first time you see a WMD. Until the time you do see your first WMD, you are operating on the assumption that the hypothesis is true. But if you do see a WMD, the hypothesis is clearly false, so you reject it. So, option 2 is a good hypothesis to "test".

Null hypotheses. In statistical analyses, the null hypothesis says 'nothing is happening'. For instance, when we are comparing two population means, the null hypothesis says there is no difference between the means. If we are investigating whether or not your ACT score is a predictor of your freshman GPA, the null hypothesis says that there is no association between ACT score and GPA. Null hypotheses are always specific and refutable. We reject the null hypothesis when our data shows that the null is sufficiently implausible, and the sample evidence supports a contradictory hypothesis known as the *alternative hypothesis*.

In practice, we label our null hypothesis H_0 and our alternative hypothesis H_a.

"Significance" – a common word stolen by statisticians

When a null hypothesis is rejected, we say that the result is "*statistically significant*." What does this really mean?

The usual dictionary definitions of significant are "having meaning" or "of a noticeably or measurably large amount", but in statistics we mean something very specific. **Statistical significance means that a result was "unlikely to have occurred by chance alone".** In particular, we mean that a result was unlikely to have occurred by chance alone if the null hypothesis was really true.

Statisticians have an agreed-on convention about what typically constitutes "unlikely". We usually say an event is unlikely if it occurs less than 5% of the time. You have seen this in your introductory statistics course through the setting of a quantity called **α**. When you ran a hypothesis test at $\alpha = 0.05$, what you were doing was setting a standard of plausibility for rejecting your null hypothesis.

p-values

A *p*-value is the probability that a particular observed sample result (or a result more extreme than that observed) could have occurred by chance if the null hypothesis were true. In short, **the *p*-value measures of the plausibility of the null hypothesis**. The following are the reasonable conclusions you could make:

- If $p \leq 0.05$, the null hypothesis is implausible in light of the sample data, and hence we should *reject* H_0 in favor of our alternative hypothesis H_a. There is sufficient evidence to conclude that H_a is true.

- If $p > 0.05$, the null hypothesis is reasonably plausible in light of the sample data, and hence we should *not reject* H_0. There is insufficient evidence to conclude that H_a is true.

In the second option above, you should note what I did not say. I did not say that because H_0 is reasonably plausible that I conclude H_0 is true. Remember that absence of evidence is not evidence of absence. Insufficient evidence to find for H_a doesn't substantiate H_0.

In reality, it is a bit artificial to pick an arbitrary boundary like 0.05 and then pretend that it builds a magical fence between significant and insignificant findings. Significance is not a black and white issue: there are "shades of gray" to consider which we will discuss.

3.7 One-sample *t*-test and CI for a mean μ using `t.test()` in R

We can use a *t*-based approach for testing a hypothesis about a population mean μ. A two-sided test has hypothesis of the form H_0: $\mu = \mu_0$ versus H_a: $\mu \neq \mu_0$ using the *t*-statistic

$$t = \frac{\text{point estimate} - \text{hypothesized value}}{\text{standard error of point estimate}} = \frac{\bar{x} - \mu_0}{\text{SE}_{\bar{x}}},$$

and the *p*-value is determined by finding an appropriate area under the relevant sampling distribution (in this case, a *t*-distribution with $n - 1$ df). The R function `t.test()` runs this test. Now, an example using the Honda Civic sedan mileage data:

► **Example: Honda Civic mileage.** A random sample of $n = 35$ 2008 Honda Civic sedans were used in testing for average fuel mileage in city driving. In 2007 Civic sedans, the mean mileage was cited as being 28.7 MPG. Based on this sample, can you conclude that the mean mileage has changed in 2008?

The correct hypotheses are

$$H_0: \mu = 28.7 \quad \text{versus} \quad H_a: \mu \neq 28.7$$

where μ is the unknown true mean mileage of 2008 Honda Civic sedans. Note that H_0 is specific and refutable. We run the test in R:

```
> attach(civicmpg)
> t.test(mpg, mu=28.7)

        One Sample t-test

data:  mpg
t = -1.1429, df = 34, p-value = 0.2611
alternative hypothesis: true mean is not equal to 28.7
95 percent confidence interval:
 27.24348 29.10795
sample estimates:
mean of x
 28.17571          ← Note: this is the sample mean, not μ!
```

The *t* test statistic value (on 34 df) is –1.143, and the *p*-value is 0.2611. Since $p > 0.05$, we fail to reject H_0. There is insufficient evidence to conclude that the true mean mileage for 2008 Civic sedans has changed since the previous year. ◄

Note that `t.test()` also provides the 95% CI for μ, negating the need to find it using the "do-it-yourself" method employed at the end of section 3.5.

3.8 Independent samples *t*-test and CI using `t.test()` in R

In *comparative inference*, we compare multiple populations with respect to some common parameter (e.g. the population means). For example, if we are comparing two populations, we can either *estimate the difference* between the two population parameters (using a CI), or *test a hypothesis about the difference* between two population parameters.

If we are comparing two population means μ_1 and μ_2, *t*-based tests and CIs usually accommodate the problem. We frequently collect random samples from each population independently, so that the values observed in one sample have no bearing or impact on what the values might be in the other sample.

The hypotheses of usual interest are H_0: $\mu_1 = \mu_2$ versus H_a: $\mu_1 \neq \mu_2$, or if equivalently expressed as differences,

$$H_0: \mu_1 - \mu_2 = 0 \quad \text{versus} \quad H_a: \mu_1 - \mu_2 \neq 0$$

The *t* test statistic and 95% CI are given respectively by

$$t = \frac{\text{point estimate} - \text{hypothesized value}}{\text{standard error of point estimate}} = \frac{(\bar{x}_1 - \bar{x}_2) - 0}{\text{SE}_{\bar{x}_1 - \bar{x}_2}}$$

and

$$(\bar{x}_1 - \bar{x}_2) \pm t_{0.025} \times \text{SE}_{\bar{x}_1 - \bar{x}_2}$$

► **Example: Offspring of diabetic mothers.** Previous research indicates that children borne by diabetic mothers may suffer from obesity, high blood pressure and glucose intolerance. Independent random samples of adolescent offspring of diabetic and non-diabetic mothers were evaluated for potential differences in vital measurements, including blood pressure (measured in mmHg). The data are in the file `diabeticoffspring.txt` in our data repository.

Use this data to (a) test to see if there is a difference in mean systolic BP between diabetic and nondiabetic offspring, and if so (b) estimate the true mean difference using a 95% CI.

```
> attach(diabeticoffspring)
> names(diabeticoffspring)
[1] "sys.bp" "mother"
> t.test(sys.bp ~ mother, var.equal=TRUE)

        Two Sample t-test

data:  sys.bp by mother
t = 4.5586, df = 177, p-value = 9.58e-06
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  4.543792 11.481208
sample estimates:
   mean in group diabetic mean in group nondiabetic
                 118.0000                  109.9875
```

The data frame consists of a numeric column of blood pressure measurements (`sys.bp`), and a factor column (`mother`) indicating the type (diabetic or not) mother. We invoke the `var.equal=TRUE` option in `t.test()` to pool the sample variances.

The *t* test statistic value (on 177 df) is 4.558, and the *p*-value is 0.00000958 (very small values in R are expressed using scientific notation: see Lecture 2). Since $p < 0.05$, we reject H_0. There is a significant difference in mean systolic BP between diabetic and non-diabetic adolescent offspring.

The corresponding 95% CI for $\mu_{diabetic} - \mu_{nondiabetic}$ is (4.54, 11.48). A formal interpretation of this result is that we can be 95% confident that the true mean systolic BP in offspring from diabetic mothers is between 4.54 to 11.48 mmHg higher than in offspring from non-diabetic mothers. ◄

3.9 Confidence intervals vs. hypothesis tests

In each of the preceding examples, R provided us with results of both a hypothesis test and a confidence interval. It is useful to look at what the results of each example say, and accordingly determine what worth you can place in a hypothesis test finding and what worth you can place in a CI finding.

Hypothesis tests
Basically, a hypothesis test result only tells you whether or not you have plausible evidence to support some specific claim. For example, in the diabetic offspring example, all the null hypothesis stated was "there is no difference in mean systolic BP between offspring from diabetic versus non-diabetic mothers". When we rejected that hypothesis, the finding was (accordingly) "there is a significant difference in mean systolic BP between offspring from diabetic and non-diabetic mothers". While that is a useful finding, it isn't exactly illuminating. A more enlightening question is: "*How* different are they?" That's where a CI has a distinct advantage.

Confidence intervals
The advantage of CIs is that not only will they tell you if a difference or effect exists, but they will also estimate the size of that effect. Remember, we found a confidence interval for the *difference* between two population means. That implies that anything inside the interval is a highly plausible value for the true mean difference. Since 0 was not inside the CI, it is implausible that the true mean difference is zero … and that's why we rejected the null hypothesis! **But the CI goes further: it tells you what the plausible values for the true difference are.** It is highly plausible that the true mean systolic BP in offspring from diabetic mothers is between 4.54 to 11.48 mmHg higher than in offspring from nondiabetic mothers.

Lecture 3 Practice Exercises

Use R to do each of the following. Use R code instructions that are as general as possible, and also as efficient as possible.

1. This is an exercise using looping and functions in R

    a. Write R code that uses the `for` looping construct to calculate the sum of the first 10 positive integers, i.e. calculates 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55. Show your code and the result of its execution. *Hint:* you might find that starting with a global variable set to 0 will be useful.

    b. Once you get the basic code to work in part a, write an R function named `sum.to.n` that uses looping to sum the first *n* positive integers: 1 + 2 + 3 + …+ *n*. The function should use `n` as its single argument. Show your code, that then execute the function for the following 3 `n` values: 10; 54; 675,919.

2. This exercise uses the existing R data frame called `uwecsample` that contains student-level data for a sample of 600 students at the University of Wisconsin at Eau Claire (i.e. UWEC). Do the following:

    d. What does the command `head(uwecsample)` do?
    e. What does the command `names(uwecsample)` do?
    f. Create side-by-side boxplots to compare the `gpa` distributions of freshmen, sophomore, juniors and seniors (see variable `class`). You should create <u>one</u> display that contains four boxplots. Comment on the GPA distributions (i.e. compare and contrast what you see).
    g. The variable HSP contains each student's high school percentile rank. Find the mean, median and standard deviation of the HSP values. What does a comparison of the mean and median values suggest?
    h. Make a histogram of the high school percentile values.
    i. Run a one-sample *t*-test to see if the true mean ACT math score (`MATH`) is different than 24, i.e. test H_0: $\mu = 24$ versus H_a: $\mu \neq 24$. Use a 5% significance level (i.e. use $\alpha = 0.05$). Interpret the *p*-value and make a conclusion in context. See INSR section 3.7 for details.

STA333 Lecture 4
So what exactly is "nonparametric" statistics?

4.1 A simple illustration

► **Example: Holiday sales data.** Average weekly sales of new cell phones at three Verizon cell phone kiosks in area malls are collected from October through December. The average number of phones sold per kiosk is recorded for each of 12 weeks and are given below:

45 32 39 29 64 55 38 212 187 124 320 188

Last year, the average weekly holiday sales per kiosk was 100 units. Is there sufficient evidence to conclude that sales this year exceeds last years' sales? Test at $\alpha = 0.05$.

The usual type of test to run for such a question is a one-sample *t*-test. For the given problem, the correct hypotheses are

$$H_0: \mu = 100 \quad \text{versus} \quad H_a: \mu > 100$$

where μ is the true mean weekly sales of new cell phones at all Verizon cell phone kiosks during October-December. We run the test in R:

```
> sales <- c(45,32,39,29,64,55,38,212,187,124,320,188)
> t.test(sales, mu=100, alternative="greater")

        One Sample t-test

data:  sales
t = 0.4048, df = 11, p-value = 0.3467
alternative hypothesis: true mean is greater than 100
```

At first glance, the test result tells us that there is insufficient evidence to reject the null hypothesis, i.e., there is not sufficient evidence to conclude that the true mean weekly sales per kiosk has increased (test statistic $t = 0.4048$, df = 11, p-value > 0.05).

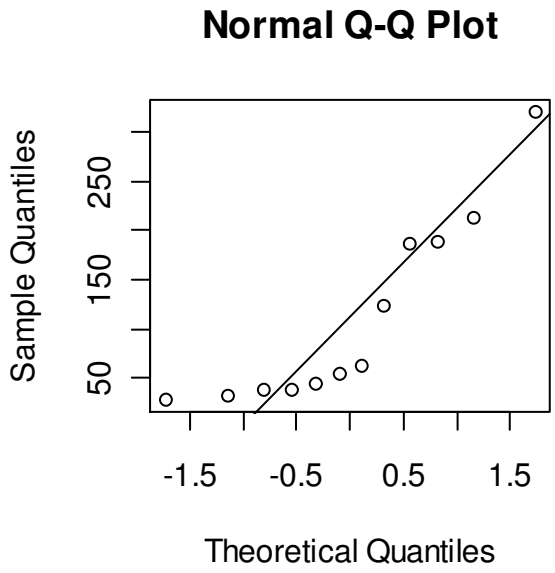
So are we finished? No. Remember that t-tests have underlying assumptions required to validate their results. Namely, the one sample t-test results are only valid if either

1. we randomly sampled from a normally distributed population, or
2. the sample size was sufficient large (usually $n > 30$ is used as a "guide").

Obviously we do not have over 30 observations, so we must check to see if our sample could have conceivably come from a population of values that is normally distributed. The best way to check this is with a visualization known as a normal quantile-quantile (or normal Q-Q) plot.

In a normal Q-Q plot, the observed sample values (the "sample quantiles") are plotted against the corresponding quantiles from a reference normal distribution (the "theoretical quantiles"). If the parent population is normally distributed, then the sample values should reasonably match up one-to-one with normal reference quantiles, resulting in a plot of points that line up in a linear (straight line) fashion. Here's the normal Q-Q plot for the `sales` vector:

```
> qqnorm(sales)    # plots the points
> qqline(sales)    # draws the reference line for matching up quantiles
```



The observed points do not line up with normal reference quantiles (note the severe bowing), so we cannot reasonably assume that the population was normal. So, we cannot trust the t-test.

Hmm. So what can we do? ◄

4.2 So what is it?

Nonparametric statistics is a strange name for a subject. Very negative. It defines the subject by what it's not. It's not *parametric statistics*.

Most modern statistics is parametric. What you learn in other statistics courses at a similar level as this one, such as STA363 (regression and experimental design) or STA365 (quality control) is parametric statistics.

So what *is* nonparametric statistics? The "counterculture"? Well … sort of.

Assumptions

Most statistical methods require assumptions to be made about the format of the data to be analyzed. For example, the usual independent samples *t*-test you learned about in introductory statistics has the following assumptions:

1. Both populations from which we drawn samples are normally distributed.
2. The samples are collected at random using simple random sampling.
3. The variances (or standard deviations) of the two populations are the same, i.e. equal.

This sounds like a pretty rigorous set of requirements. Fortunately, these assumptions are often valid in practice. And in some cases, if the sample sizes are sufficiently large, we can relax the normality assumption because the Central Limit Theorem will help ensure that the sampling distribution of our test statistic is nearly normal, regardless of the shape of the parent population.

However, there *will* be situations in which one or more of these assumptions may be violated, and in these cases it may be inappropriate to use traditional (parametric) methods of analysis.

Conventional statistical methods such as the *t*-test are known as **parametric methods** because they require estimation of the parameters that define the underlying distribution of the data. In the case of the *t*-test, for instance, these parameters are the mean μ and standard deviation σ that define the normal distribution.

Nonparametric statistics provides an alternative series of statistical methods that require no or very limited assumptions to be made about the data. There is a wide range of methods that can be used in different circumstances.

Unlike their parametric counterparts, non-parametric methods make no specific assumptions about the distribution of the data, nor do they necessarily rely on estimates of population parameters (such as the mean) in order to describe a variable's distribution. For this reason, nonparametric statistics are also often referred to as *distribution-free* statistics.

The advent of high-speed computing and software like R has also had the effect of spawning new nonparametric techniques as well as approaching traditional nonparametric methods in a new light. We will see some of these as the semester progresses.

4.3 Efficiency vs. robustness

OK, OK … assumptions, assumptions. What's the big deal?

Efficiency

When we are willing to assume a specific distribution for the population, this has very strong consequences. Statistical theory has proven that the sample mean and sample SD are the *best possible estimators* of the parameters μ and σ, *when the assumed population distribution is normal*. By "best", we mean that these estimators have smaller asymptotic uncertainty than any others, where asymptotic refers to the limit as the sample size *n* goes to infinity.

This concept is known as *asymptotic efficiency*. Note (and this is very important) that this efficiency property depends crucially on the assumed population distribution.

Robustness

But what if the assumed distribution for the population is wrong? Then the whole efficiency story goes out the window, and what was "best" (asymptotically efficient) can become the *worst*.

We say an estimator is *robust* when it tolerates departures from assumptions. One of the simplest measures of robustness is the **breakdown point**, which can be defined as the asymptotic fraction of the data that can be complete junk while the estimator stays sensible. This point is usually explained in intro stats without being explicit about the terminology. Consider:

The sample mean $\bar{x}$ has breakdown point zero. Why?
The sample median has breakdown point 50%. Why?

There is a trade-off. Efficiency and robustness generally go in opposite directions. The mean is most efficient and least robust. The median is most robust and least efficient (when the assumed population is normal). Trimmed means are in between on both robustness and efficiency.

In short, if you have perfect data (no errors, no outliers, fits the assumed normal population model), then you should use the sample mean and related procedures (*t* tests, linear regression, ANOVA *F*-tests, etc).

Nonparamateric statistics is about what you do when you don't have perfect data (or at least don't want to take a chance on imperfect data).

4.4 The zoo of nonparametric tests

Nonparametric (NP) statistical tests may be identified based on their "parametric" counterparts. For example, a nonparametric analog to the two-independent samples *t*-test is known as the Mann-Whitney-Wilcoxon test. I will certainly present new NP procedures alongside their parametric partners, but for the purpose of learning what these techniques do, it is also informative to classify NP procedures according to the statistical principles on which they are based.

In STA333, the zoo of nonparametric statistical methods can be largely classified into five broad areas as follows:

1. **Methods based on the binomial sampling distribution.** You may have already learned about the binomial distribution in an earlier stat course. (Or maybe not.) If not, you'll learn it soon enough. These methods are useful for one sample problems involving hypotheses or confidence intervals about population percentiles (e.g. the median is the 50^{th} percentile).

2. **Methods based on permutations of the data.** Many NP procedures are based upon the notion of re-arranging, or "permuting" the data values. Oftentimes, these methods involve only dealing with the ranks of the data values rather than values themselves.

3. **Methods based on "bootstrapping" the data.** When building confidence intervals using traditional parametric statistics, we require knowledge of the sampling distribution of the point estimate in order to calculate the upper and lower limits of the CI. Theory or assumption usually dictates what we use as the sampling distribution. But what if we don't know the theory, or if the theory doesn't apply? "Bootstrapping" is a way of building the sampling distribution of the point estimate ourselves simply by resampling from the observed data we have in hand.

4. **Methods based on "smoothing" the data.** For example, in traditional (parametric) regression analysis, you specify a functional form for the relationship between two variables x and y, e.g. a linear relation is modeled via a straight line function given by $y = \beta_0 + \beta_1 x$, where β_0 is the y-intercept of the fitted line and β_1 is the slope of the fitted line. Smoothing methods are a nonparametric way of fitting curves to data *without* explicitly specifying any functional form for the curves. For example, imagine plotting the price of a stock vs. time and then drawing a smooth curve through the data to indicate the trend over time. This curve may have bumps or unique features based on the observed trend that a linear function would miss.

5. **Classification and regression trees.** Trees help us explore the structure in a set of data, while developing easy to visualize decision rules for predicting a categorical or continuous outcome from a set of predictor variables. It is the nonparametric analog to multiple regression, and is a fundamental tool used in data mining.

4.5 Advantages and disadvantages of nonparametric methods

Given that NP methods make less stringent demands on the data, one might wonder why they are not used more often. There are a few reasons:

1. **Tradition.** Most traditional statistical analyses are parametric. Old habits are hard to break.

2. **Loss of information.** Many nonparametric procedures discard information. For example, if we convert severely skewed interval data into ranks, we are discarding the actual values and only retaining their "order". Because vital information is discarded, nonparametric tests can be limited in their ability to detect real effects as compared to parametric methods. (This also means that nonparametric tests typically require comparably larger sample sizes in order to demonstrate an effect when it is present.)

3. **Limitations.** There are certain types of information that only parametric statistical tests can provide.

For these reasons, you will usually see nonparametric analyses used primarily on an as-needed basis as a substitute for parametric tests when their assumptions are grossly violated, e.g., when a distribution is severely skewed.

Advantages of NP statistics:

- They can be used with non-normally distributed data.
- They can be used with discrete data (nominal or ordinal measurement level), whereas many parametric methods require continuous data (interval, ratio measurement level)
- Methods can be highly customizable.

Disadvantages of NP statistics:

- Many NP methods "waste"' information or discard features of the data
- They are usually not as "efficient" as their parametric counterparts when the assumptions can be met.

Lecture 4 Practice Exercises

Use R to do each of the following. Use R code instructions that are as general as possible, and also as efficient as possible.

1. The following are the gasoline mileages (per gallon) obtained with eight tankfulls each of two kinds of gasoline. The experiment used 16 cars of the same make and model.

Gasoline A:	26.3	24.6	26.7	26.7	27.1	26.3	26.1	25.0
Gasoline B:	27.2	26.7	26.6	27.3	29.9	28.7	28.1	26.4

   a. What might be some secondary reasons (i.e. reasons not under study) that might introduce variability into these measurements? How could the experimenter attempt to control these sources of variability?
   b. Run an independent samples *t*-test to see if the true mean mileage differs between brands A and B. Cite all elements of the test (null and alternative hypotheses expressed in terms of population parameters, test statistic value, *p*-value, and decision/conclusion in context).
   c. If the result in part b is statistically significant, find and interpret a 95% confidence interval for the true mean difference in mileage between the brands.
   d. What are the assumptions underlying the analyses in parts b and c? How can we check them?

2. On page 60, the issue of "breakdown point" is discussed. Respond to the two questions posed on that page:

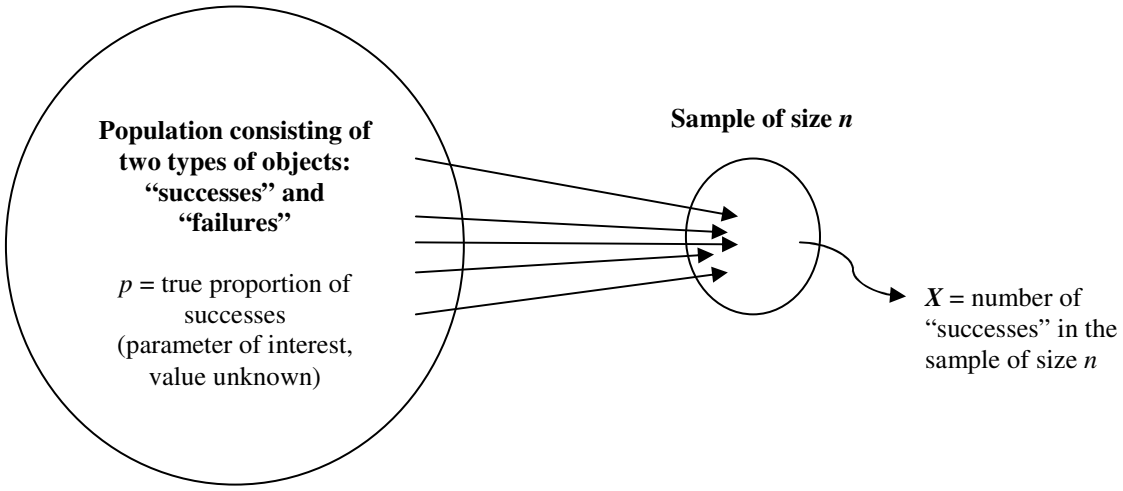
   a. *The sample mean $\bar{x}$ has breakdown point zero. Why?*
   b. *The sample median has breakdown point 50%. Why?*

STA333 Lecture 5
The binomial sampling distribution

5.1 The basic premise

What gives rise to the binomial sampling distribution? Consider the following:

1. We have a large population that, as far as we are concerned, consists of only two types of objects, which we generically refer to as *successes* and *failures*.
2. We randomly sample n items from the population. The sample statistic of interest is X = the number of successes in the sample.
3. We wish to use X to make inferences about the parameter p = the true proportion of successes in the population. p is unknown.



What is the probability distribution of X? How can we calculate probabilities associated with X?

The binomial sampling distribution can be used to model likelihoods and probabilities associated with the number of observed successes in a sample. It has wide ranging applicability. Before I derive it, I provide several diverse examples, all of which use the binomial distribution.

► **Example: Coin flips.** Suppose you toss a fair coin 10 times. What is the chance that you see exactly four heads in the ten tosses? Exactly nine heads? All heads?

► **Example: Random guessing.** Suppose you are to take a quiz that has five multiple-choice questions, where each question has 4 possible responses (of which only one is correct). You fail to study for the test, so you just randomly guess on every question. What is the chance you get exactly two of the questions right? If 60% is the minimum passing grade, what is the chance you fail the test?

► **Example: Public opinion polling.** Suppose that Obama's public approval rating is truly 62%, i.e. 62% of the American public approves of how Obama is conducting his presidency. If you randomly sampled 1000 Americans, what is the chance that your sample would produce an estimate of Obama's approval rating that is correct to within a margin of error of ± 3%, i.e. produces a sample estimate between 59% and 65%?

► **Example: Quality control.** A manufacturing process is deemed to be "in control" if it produces no more than 5% defects. A quality control engineer randomly samples 40 items and detects 3 defective items. Should the process be declared "out of control"?

► **Example: Holiday sales data.** Median weekly sales of new cell phones at three Verizon cell phone kiosks in area malls are collected from October through December. The average number of phones sold per kiosk is recorded for each of 12 weeks and are as follows: 45, 32, 39, 29, 64, 55, 38, 212, 187, 124, 320, 188. Last year, the median weekly holiday sales per kiosk was 100 units. Is there sufficient evidence to conclude that sales this year exceeds last years' sales?

5.2 Derivation of the binomial distribution

Let's start with the random guessing example above. When we guess at random ten times, each outcome will be a success or failure. Suppose we define a correct answer to be the "success" outcome. Then the probability of getting a correct answer on any one question is

$$p = P(\text{success}) = P(\text{correct answer}) = ¼ = 0.25$$

From the population perspective, you may think of p as the true proportion of times that you would guess the correct answer if you had continued answering test questions ad nauseum.

OK, so now what? Take the test once, randomly guessing $n = 5$ times. You'll see a sequence of correct answers (C) and wrong answers (W). Let X = the number of correct answers on the quiz. Suppose you see this happen:

$$WWCCW$$

What's the chance of this sequence happening? Well, since it is a sequence of independent outcomes, basic probability rules can be used to show that

$$\begin{aligned} P(WWCCW) &= P(W) \times P(W) \times P(C) \times P(C) \times P(W) \\ &= 0.75 \times 0.75 \times 0.25 \times 0.25 \times 0.75 \\ &= (0.25)^2(0.75)^3 \\ &= 0.02636. \end{aligned}$$

So, is this the probability of getting two correct out of five, i.e. is this $P(X = 2)$? No. Why not? *Because this is only one sequence where we could see two Cs and three Ws.* In particular, the sequence *WWCCW* refers only to the sequence where we get questions 3 and 4 correct. What if we got questions 1 and 4 correct? Or 2 and 5? Or 3 and 5? Each of these are cases where $X = 2$ also.

Combinations. In order to correctly calculate the probability of seeing two correct answers and three wrong answers, we must figure out how many ways we can "choose" two successes from the five possible questions. Here are all the possible sequences:

CCWWW	*CWCWW*	*CWWCW*	*CWWWC*	*WCCWW*
WCWCW	*WCWWC*	*WWCCW*	*WWCWC*	*WWWCC*

The mathematical way to "count" how many combinations there are is through use of the *binomial coefficient:*

$$\binom{n}{x} = \frac{n!}{x!\,(n-x)!}$$

where $n! = n(n-1)(n-2)\cdots 3 \cdot 2 \cdot 1$. (Note: 0! = 1 by definition). This is read "n choose x". In particular, the number of ways to "choose" 2 successes from five questions is given by

$$\binom{5}{2} = \frac{5!}{2!\,(5-2)!} = \frac{5!}{2!\,3!} = \frac{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{2 \cdot 1 \cdot 3 \cdot 2 \cdot 1} = 10.$$

Thus, the probability of seeing exactly two correct answers among the five questions is

$$P(X = 2) = \binom{5}{2}(0.25)^2(0.75)^3 = 10(0.02636) = 0.2636,$$

i.e. there is a 26.4% chance you would get exactly two out of five correct by guessing. ◄

Hopefully you can see the structure in where the probability comes from. If so, it is easy to find the **true** probabilities for <u>all</u> the possible outcomes in the example, i.e. the probabilities associated with $X = 0, 1, 2, 3, 4$, and 5. Here they are:

$$P(\text{none correct}) = P(X = 0) = \binom{5}{0}(0.25)^0(0.75)^5 = 0.2373$$
$$P(\text{one correct}) = P(X = 1) = \binom{5}{1}(0.25)^1(0.75)^4 = 0.3955$$
$$P(\text{two correct}) = P(X = 2) = \binom{5}{2}(0.25)^2(0.75)^3 = 0.2636$$
$$P(\text{three correct}) = P(X = 3) = \binom{5}{3}(0.25)^3(0.75)^2 = 0.0879$$
$$P(\text{four correct}) = P(X = 4) = \binom{5}{4}(0.25)^4(0.75)^1 = 0.0146$$
$$P(\text{all correct}) = P(X = 5) = \binom{5}{5}(0.25)^5(0.75)^0 = 0.0009$$

Note that these probabilities sum to 1, confirming that this is a legitimate probability distribution. All of these probabilities together collectively constitute the binomial sampling distribution when $n = 5$ and $p = 0.25$.

The binomial sampling distribution

- We sample n items at random from a large population consisting only of successes and failures, where p = true proportion of successes in the population.
- X = the number of observed successes in the sample.

If these are satisfied, then X is said to possess the binomial sampling distribution with parameters n and p. The form of the probability distribution is given by

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x} \quad \text{for } X = 0, 1, 2, \ldots, n.$$

In shorthand, we denote this by writing $X \sim Bin(n, p)$.

Mean ("expected value") and standard deviation of $X \sim Bin(n, p)$

The mean of a binomial sampling distribution is $\mu = np$.
The standard deviation of a binomial distribution is $\sigma = \sqrt{np(1-p)}$.

5.3 Using the binomial distribution calculator in R

In R, we can work with the binomial distribution using a choice of different functions:

- `dbinom(x,n,p)` will calculate $P(X = x)$ where $X \sim Bin(n, p)$. In other words, this function returns specific probabilities associated with individual values of X.

- `pbinom(x,n,p)` will calculate $P(X \leq x)$ where $X \sim Bin(n, p)$. In other words, this function returns cumulative probabilities for all values of X up to and including x. This is called the binomial cumulative distribution function, or CDF.

- `rbinom(nreps,n,p)` will randomly generate `nreps` values of X from the $Bin(n, p)$ distribution. This is for the purpose of simulation studies.

Here are a few examples.

► **Example: Random guessing.** Suppose you are to take a quiz that has five multiple-choice questions, where each question has 4 possible responses (of which only one is correct). You fail to study for the test, so you just randomly guess on every question.

What is the chance you get exactly two of the questions right?

```
> dbinom(2,5,0.25)        # P(X=2) where X~Bin(5,0.25).  We did this earlier.
[1] 0.2636719

> dbinom(0:5,5,.25)       # All individual probability values for Bin(5,0.25)
[1] 0.2373046875 0.3955078125 0.2636718750 0.0878906250 0.0146484375
[6] 0.0009765625
```

If 60% is the minimum passing grade, what is the chance you fail the test?

We'd fail if we got fewer than 3 out of 5 correct, i.e. if $X = 0$, 1 or 2 where $X \sim Bin(5, 0.25)$. So, let's use R to find $P(X \leq 2)$ where $X \sim Bin(5, 0.25)$:

```
> pbinom(2,5,0.25)        # P(X<=2) where X~Bin(5,0.25)
[1] 0.8964844
```

What is the probability of passing this test?

```
> 1-pbinom(2,5,0.25)      # P(X>=3) = 1-P(X<3) = 1-P(X<=2)
[1] 0.1035156
```

Note that X is discrete, so we must be mindful of specifying the probability correctly. ◄

► **Example: Public opinion polling.** Suppose that Obama's public approval rating is truly 62%, i.e. 62% of the American public approves of how Obama is conducting his presidency. If you randomly sample 1000 Americans, what is the chance that your sample would produce an estimate of Obama's approval rating that is correct to within a margin of error of ± 3%, i.e., it produces a sample estimate between 59% and 65%?

Solution. Let X = the number of persons polled out of 1000 who approve of Obama. Then, $X \sim Bin(1000, 0.62)$. If between 59% and 65% of our sample approves of Obama, that translates to seeing a value of X somewhere between 590 (59% of 1000) and 650 (65% of 1000). So, we want to calculate $P(590 < X < 650)$.

However, we must express this in the form of a CDF. So, we rewrite it as follows:

$$P(590 < X < 650) = P(591 \leq X \leq 649) = P(X \leq 649) - P(X \leq 590)$$

Now, putting this into R yields:

```
> pbinom(649,1000,0.62) - pbinom(590,1000,0.62)
[1] 0.9454456
```

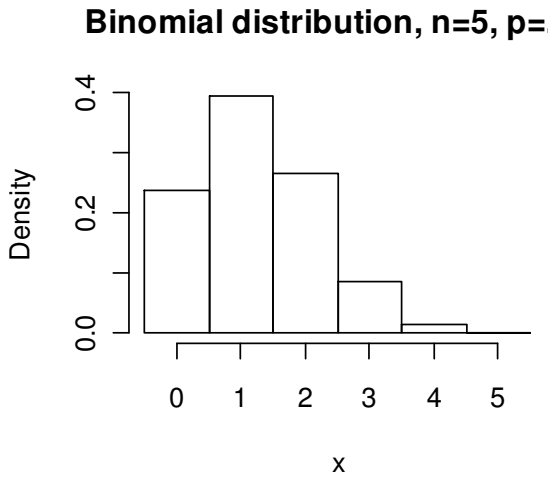
There is nearly a 95% chance that the sample estimate will be within ± 3% of the true approval rating. That's a pretty good estimate. (In statistical jargon, we'd say that we have 95% confidence in such an estimate). ◄

At this point, it might be instructive to look at a few different binomial distributions graphically. This will help give us a sense of how the overall probability patterns for binomials look. We can do this in R by **simulation** – having the computer randomly generate large numbers of random samples and calculating the value of X for each of them. The simulation steps are as follows:

1. Randomly draw a sample of size n from the population with proportion of successes p.
2. Calculate X = number of successes observed in the sample of size n.
3. Repeat steps 1 and 2 a very large number of times, say 10,000.
4. Make a histogram out of the 10,000 observed values of X. This will be a very close approximation of the shape of the *Bin(n, p)* sampling distribution.

► **Example: Random guessing.** Let's simulate taking the test 10,000 times and seeing the sampling distribution of how our scores would come out. We randomly generate 10,000 samples from the $Bin(5, 0.25)$ distribution and look at the probability pattern:

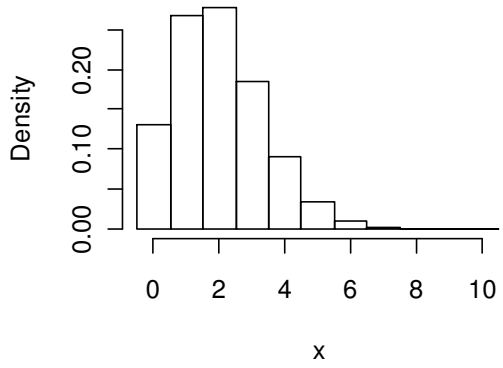
```
> nreps <- 10000
> n <- 5
> p <- .25
> x <- rbinom(nreps,n,p)
> hist(x,
+      breaks=seq(-0.5,n+0.5,1),
+      probability = TRUE,
+      main = 'Binomial distribution, n=5, p=.25')
```



You can see that the distribution is highly discrete (since X can only take on the values 0, 1, 2, 3, 4, 5). It is skewed to the right since $p = P$(success) is relatively low. Note that the probabilities from this simulation are virtually spot-on the true values (see secton 5.2). ◄

► **Example: Quality control.** (See the fourth example in section 5.1) Let's simulate taking 10,000 samples, each with 40 items, from a production process that is barely in control, i.e. is producing exactly 5% defects, and see how the sampling distribution for X = number of defects in a sample of size 40 comes out. We randomly generate 10,000 samples from the *Bin*(40, 0.05) distribution and look at the probability pattern:

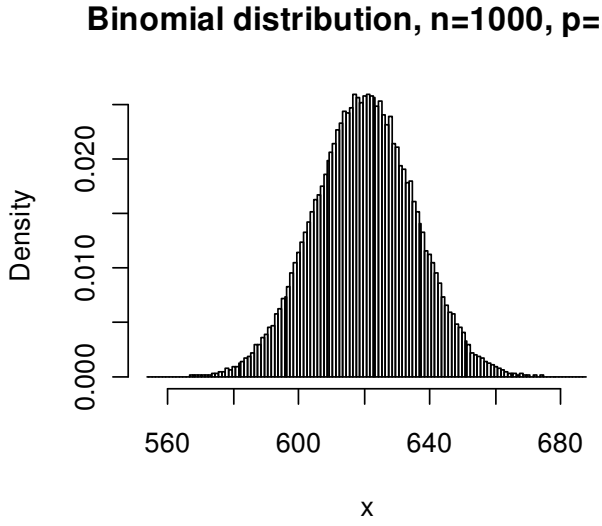
```
> nreps <- 10000
> n <- 40
> p <- .05
> x <- rbinom(nreps,n,p)
> hist(x,
+      breaks=seq(min(x)-0.5,max(x)+0.5,1),
+      probability = TRUE,
+      main = 'Binomial distribution, n=40, p=.05')
```



What do you see? The distribution is skewed right. The distribution is less "granular", i.e., since each sample had 40 items in it, there are more possible values of X (0, 1, 2, 3, …, 40). But the most important thing to see is that it is quite likely to see more 3 or more defective items in a sample of size 40, even if the process is in control. This should affect your decision about whether or not to call the process "out of control" if you actually observe 3 defects! ◄

► **Example: Public opinion polling.** OK, let's simulate the random drawing of 10,000 samples, each containing 1000 Americans, and find X = number out of 1000 who approve of Obama for each sample. Here goes:

```
> nreps <- 10000
> n <- 1000
> p <- .62
> x <- rbinom(nreps,n,p)
> hist(x,
+      breaks=seq(min(x)-0.5,max(x)+0.5,1),
+      probability = TRUE,
+      main = 'Binomial distribution, n=1000, p=.62')
```



Hmm. Is this a binomial distribution? Sure is. *But it also looks a lot like a normal distribution!* This example provides us with some valuable insight about the binomial:

Normal approximation to the binomial distribution

Suppose X has the *Bin(n, p)* sampling distribution. If both $np > 10$ and $n(1-p) > 10$, then the *Bin(n, p)* distribution is well approximated by the normal distribution with mean $\mu = np$ and standard deviation $\sigma = \sqrt{np(1-p)}$, i.e.,

$$X \xrightarrow{approx} N\left(\mu = np, \sigma = \sqrt{np(1-p)}\right).$$

► **Example: Public opinion polling.** Let's revisit the Obama's public approval rating question. Earlier, we found using the binomial sampling distribution and R that

$$P(590 < X < 650) \;=\; P(X \leq 649) - P(X \leq 590) \;=\; 0.9454456$$

Let's redo this using the normal approximation to the binomial. Since $np = 1000(0.62) = 620 > 10$ and $n(1-p) = 1000(0.38) = 380 > 10$, this will yield a good approximation. The approximating normal curve will be

$$X \xrightarrow{approx} N\left(\mu = 1000(0.62), \sigma = \sqrt{100(0.62)(0.38)}\right) = N(\mu = 620, \sigma = 15.35)$$

Using the `pnorm()` function first mentioned in the lecture 1 notes, here is the normal approximation:

```
> pnorm(649,620,15.35)-pnorm(590,620,15.35)
[1] 0.9452438
```

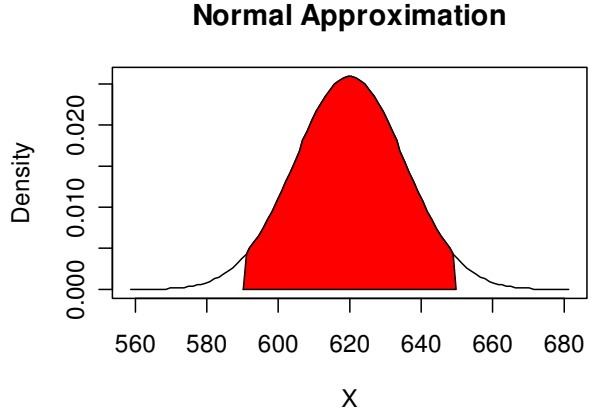
Pretty good, eh?

We can even improve the approximation a bit by incorporating a *continuity correction*. This can help since X is truly a discrete variable (binomials always are) but we are using a continuous distribution (the normal) to approximate a probability. The continuity correction is just a "fudge factor" of 0.5 correctly applied to the cutpoints of 590 and 650:

```
> pnorm(650-0.5,620,15.35)-pnorm(590+0.5,620,15.35)
[1] 0.9453721
```

The value $X = 650$ is not to be included in the probability calculation, so we "nudge" the boundary down to 649.5 when we calculate the area to the left using `pnorm()`. Also, the value $X = 590$ is not to be included … see if you can reason why the continuity correction meant nudging the boundary up to 590.5 in this case.

Below is a histogram picturing the normal curve area used as the binomial approximation. ◄



Lecture 5 Practice Exercises

Use R to do each of the following. Use R code instructions that are as general as possible, and also as efficient as possible.

1. Suppose you collect a random sample of ten tosses of a fair coin. Let X = the number of heads that you see.

    a. What are the possible values of X?
    b. What is the distribution of X? Be sure to also give the values of the parameters for the sampling distribution.
    c. What is the chance that you see exactly 4 heads in the ten tosses?
    d. What is the probability that you see at most 4 heads in the ten tosses?
    e. What is the chance that you get eight or more heads?
    f. What is the probability that, just by chance alone, you see all heads in your ten tosses?
    g. Refer to part f. Suppose you randomly flip what you believe to be a fair coin ten times, and you actually observe ten heads. Can you make an inference about the coin?

2. Look around the room today. How many ways are there for me to randomly select a sample of five students to come up to the board to publically embarrass?

3. The following scenario does not qualify Y as a binomial sampling problem. Can you determine why it violates the binomial requirements?

    "Among 12 male applicants for a job with the postal service, nine have working spouses. Suppose that two of the applicants are randomly chosen for further consideration. Let Y = the number of the chosen applicants who have a working spouse."

4. An advertising agency conducted an ad campaign aimed at making consumers in Virginia aware of a new product. Upon completion of the campaign, the agency claimed that 20% of all consumers in VA had become aware of the product. The product's distributor surveyed 1000 consumers and found that 150 were aware of the product.

    a. Let W be the number of consumers out of 1000 sampled who are aware of the product. What is the distribution of W?
    b. Find the probability that 150 or fewer consumers in a random sample of 1000 consumers in the state would be aware of the product (that is, assuming the agency's claim of 20% is correct).

STA333 Lecture 6
Binomial test for a median or percentile

6.1 Why do we need this?

Recall the holiday cell phone sales data example from lecture 4: average weekly sales of new cell phones at three Verizon cell phone kiosks in area malls were collected from October through December, and the average number of phones sold per kiosk was recorded for each of 12 weeks as follows:

45 32 39 29 64 55 38 212 187 124 320 188

Last year, the average weekly holiday sales per kiosk was 100 units. The question was: is there sufficient evidence to conclude that sales this year exceeds last years' sales of 100 units per week per kiosk?

The usual type of test to run for such a question is a one-sample t-test. For the given problem, the correct hypotheses are

$$H_0: \mu = 100 \quad \text{versus} \quad H_a: \mu > 100$$

where μ is the true mean weekly sales of new cell phones at all Verizon cell phone kiosks during October-December. We ran the test in R earlier (here's the code again):

```
> sales <- c(45,32,39,29,64,55,38,212,187,124,320,188)
> t.test(sales, mu=100, alternative="greater")
```

But recall that we failed to meet the assumption required of the *t*-test, so the results were doubtful. What we will do now is develop a nonparametric version of a hypothesis test for location in a distribution. (The mean is a location of the distribution's center).

6.2 Binomial test for a population median M (or percentiles in general)

In nonparametric statistics, we frequently refer not to what the value of a variable is, but rather where it resides in relation to other values. For this reason, a more useful measure of distributional center in this instance is the median M rather than the mean μ. Recall that the median is a special case of a percentile:

> The p^{th} percentile of a distribution is a value such that p% of the distribution lies below it, and $(100 - p)$% lies above it.

So in the case of a median, 50% of the distribution lies below it and 50% above it. Let's call the 50^{th} percentile of a population p_{50}, i.e. $M = p_{50}$. Let's develop a test for M, and then we can easily generalize it to other percentiles (e.g. the 80^{th} percentile p_{80}, the first quartile Q_1, etc).

An "exact" test for the true median M

Hypotheses. Suppose we have a random sample $X_1, X_2, \ldots, X_n$ of size n from a population measured on a continuous scale. The hypotheses to be considered are

$H_0: M = M_0$ vs. $H_a: M > M_0$ (upper-tailed test)
$H_0: M = M_0$ vs. $H_a: M < M_0$ (lower-tailed test)
$H_0: M = M_0$ vs. $H_a: M \neq M_0$ (two-sided test)

where M_0 is a hypothesized value of the population median.

Test statistic. Let T = the number of observations in the random sample that exceed M_0.

Null distribution of the test statistic. Here's where you really need to understand the underlying process of hypothesis testing to know what is going on. If in fact $H_0: M = M_0$ were true, then each X_i in our random sample would have had a 0.50 probability of falling above M_0. Thus, if indeed H_0 were true and we collected n such observations at random and let T = the number of observations that exceed M_0, T will have the binomial sampling distribution with parameters n and $p = 0.50$. We say that the null distribution of T is $T \sim Bin(n, 0.50)$.

p-value. By definition, the p-value for any hypothesis test is the probability of seeing sample evidence at least as contradictory to H_0 (and in favor of H_a) as that which you observed in the actual sample. In this case, the p-value will be some probability from the $Bin(n, 0.50)$ null distribution. We then can compare this to some pre-determined significance level α.

Assumptions. Only two:

1. The sample is random.
2. The measurement scale for the items in the population is continuous.

Now, it's time for an example!

► **Example: Holiday sales data.** Median weekly sales of new cell phones at three Verizon cell phone kiosks in area malls are collected from October through December. Phones sales per kiosk are recorded for each of 12 weeks and are given below:

45 32 39 29 64 55 38 212 187 124 320 188

Last year, the median weekly holiday sales per kiosk was 114 units. Is there sufficient evidence to conclude that median sales this year are lower than last year? Test at $\alpha = 0.05$.

Solution. Let M = true median sales this year during the same period. The hypotheses are

$$H_0: M = 114 \quad \text{versus} \quad H_a: M < 114$$

We have a random sample of $n = 12$ weeks of data. Of these, there are 5 weeks where the sales exceed 114 units. So, our test statistic is $T = 5$. Under the null hypothesis, $T \sim Bin(12, 0.5)$. Thus, the p-value for the test is given by $P(T \leq 5)$ where $T \sim Bin(12, 0.5)$. So,

$$p\text{-value} = P(T \leq 5) = \sum_{i=0}^{5} \binom{12}{i} (0.5)^i (1 - 0.5)^{12-i}.$$

This is already in the "≤" form of a CDF, so we can go straight to R to calculate the probability:

```
> sales <- c(45,32,39,29,64,55,38,212,187,124,320,188)
> T <- length(sales[sales>114])
> pvalue <- pbinom(T,12,0.5)
> pvalue
[1] 0.387207
```

The p-value is 0.3872, so H_0 is reasonably plausible given the data. Thus, there are no grounds upon which to conclude that median sales are now below 114 units per week. ◄

We can accomplish the same thing using the built-in R function `binom.test()`. The form is

```
binom.test(x, n, p = 0.5,
           alternative = c("two.sided", "less", "greater"),
           conf.level = 0.95)
```

where `x` is the number of successes (in this case, that's T). $p = 0.5$ is the default, but you can change this if you'd like. Specify the alternative H_a choice:

```
> sales <- c(45,32,39,29,64,55,38,212,187,124,320,188)
> T <- length(sales[sales>114])
> binom.test(T,12,alternative="less")

        Exact binomial test

data:  T and 12
number of successes = 5, number of trials = 12, p-value = 0.3872
alternative hypothesis: true probability of success is less than 0.5
```

```
95 percent confidence interval:
 0.0000000 0.6847622
sample estimates:
probability of success
             0.4166667
```

Note that R expresses the hypotheses in terms of *p*, not *M*. This makes sense. If it's still unclear to you, think of it this way:

- If H_0: $M = 114$ were true, then T has the binomial distribution with $n = 12$ and $p = 0.5$.
- If H_a: $M < 114$ were true, then T has a binomial distribution with $n = 12$, but with $p < 0.5$.

So, testing the hypotheses H_0: $M = 114$ versus H_a: $M < 114$ is absolutely equivalent to testing the hypotheses H_0: $p = 0.5$ versus H_a: $p < 0.5$. (Remember: $M = p_{50}$, so in terms of percentiles, our hypotheses can be written as H_0: $p_{50} = 114$ versus H_a: $p_{50} < 114$.)

► **Example: ACT scores.** A random sample of 32 Miami students was selected, and their ACT scores recorded:

28, 27, 29, 27, 29, 31, 32, 30, 34, 30, 27, 25, 30, 32, 35, 32
23, 26, 27, 33, 33, 33, 31, 25, 28, 34, 30, 33, 28, 26, 30, 28

The ACT composite score of 25 is at the 80th percentile nationally. Run a hypothesis test at $\alpha = 0.05$ to see if there is sufficient evidence to conclude that for Miami students, the 80th percentile exceeds the national figure.

Solution. The hypotheses are H_0: $p_{80} = 25$ versus H_a: $p_{80} > 25$. Our test statistic T is the number of sampled MU students with ACT score exceeding 25. So, if H_0 is true, $T \sim Bin(32, 0.20)$. We run the binomial test in R:

```
> x<-c(28,27,29,27,29,31,32,30,34,30,27,25,30,32,35,32,23,26,27,33,33,33,31,
+      25,28,34,30,33,28,26,30,28)
> T<-length(x[x>25])
> binom.test(T,length(x),p=0.2,alternative="greater")

        Exact binomial test

data:  T and length(x)
number of successes = 29, number of trials = 32, p-value < 2.2e-16
alternative hypothesis: true probability of success is greater than 0.2
95 percent confidence interval:
 0.7751839 1.0000000
sample estimates:
probability of success
               0.90625
```

We see that our test statistic value is $T = 29$, and the p-value is ≈ 0. So if testing at $\alpha = 0.05$, there is strong evidence suggesting that the 80th percentile of ACT composite scores at MU exceeds the national figure.

Confidence interval. `binom.test()` also gives a 95% lower confidence limit for the "probability of success" p, which in our case is defined by T (i.e. a "success" is defined to be "an MU student with an ACT score exceeding 25").

In the above example, the calculated CI is (0.775, 1.000). In effect, only an effective lower limit (0.775) is supplied here because we used the `alternative="greater"` option: doing so tells R that we are interested only in determining a 95% lower confidence bound for p. (This is sometimes referred to as a **one-sided confidence interval**). You can ignore the provided "upper" limit value of 1.00 in this case. If you want the typical two-sided CI (with both effective lower and upper limits), you should instead use the `alternative="two.sided"` option.

In the previous output, the results indicate that we can be 95% confident that at least 77.5% of the MU student population has an ACT composite score exceeding 25. ◄

Lecture 6 Practice Exercises

Use R to do each of the following. Use R code instructions that are as general as possible, and also as efficient as possible.

1. Adult Americans sleep an average of 7.8 hours per day. You think college students sleep less than this on average, so you collect a random sample of fifteen college students at Miami and get the following data on their typical daily sleep amount (in hours):

6.7	4.5	6.4	8.6	5.5	8.2	5.9	7.5
4.4	6.0	6.3	8.3	7.3	5.7	10.1	

   Suppose we wish to see if there is sufficient evidence to conclude that the median amount of daily sleep for MU college students is below 7.8 hours (i.e. below the adult American average). Run a binomial test to address the research question. Give all elements of the test:

   - null and alternative hypotheses
   - test statistic
   - null distribution of test statistic
   - *p*-value
   - decision and conclusion in discipline context

2. Refer to the `uwecsample` data frame in R, which contains current data on a sample from the UWEC undergraduate population. It is known that ten years ago, the median high school percentile ranking for students at UWEC was 73.5, i.e. half the students at UWEC were below the 73.5th percentile of their high school class, and half were above the 73.5th percentile of their high school class. Since that time, UWEC administrators have claimed that they have raised admissions standards, so that they now believe the true median HSP value is higher than it was ten years ago.

   a. Test this claim using a binomial test. Use the built-in R function `binom.test()`.

   b. Repeat (i.e. duplicate) part a, but find the *p*-value directly using the binomial probability calculator `pbinom()`.

   c. When using the `binom.test()` function with the `alternative="two.sided"` option, you get a two-sided 95% confidence interval for *something*. (!) See if you can trace all that you've done, and look at the output generated, to figure out *what* this is a CI for (i.e. *what* is it that this confidence interval is estimating?). Once you know, interpret the CI in context.

STA333 Lecture 7
Confidence intervals for medians or percentiles

7.1 Old stuff: *t*-based confidence interval for the mean

We know how to find a 95% confidence interval for the mean μ using usual *t*-based formula

$$\bar{x} \pm t_{0.025} \frac{s}{\sqrt{n}}$$

This may be easily computed using the `t.test()` function in R.

However, when the usual normal-theory based assumptions are not met, we can instead resort to finding a 95% confidence interval for the population median *M* as a nonparametric alternative. This is not as simple an undertaking, but if you can follow the logic of CI formulation, it should make sense.

Also, the procedure can be easily generalized to percentiles other than the median (e.g. we could find a CI for the first quartile, the 80th percentile, etc.)

7.2 A nonparametric confidence interval for *M*

Recall the basic notion of confidence interval construction: to estimate some population parameter θ using a 95% confidence interval, we need to find two constants c_1 and c_2 such that

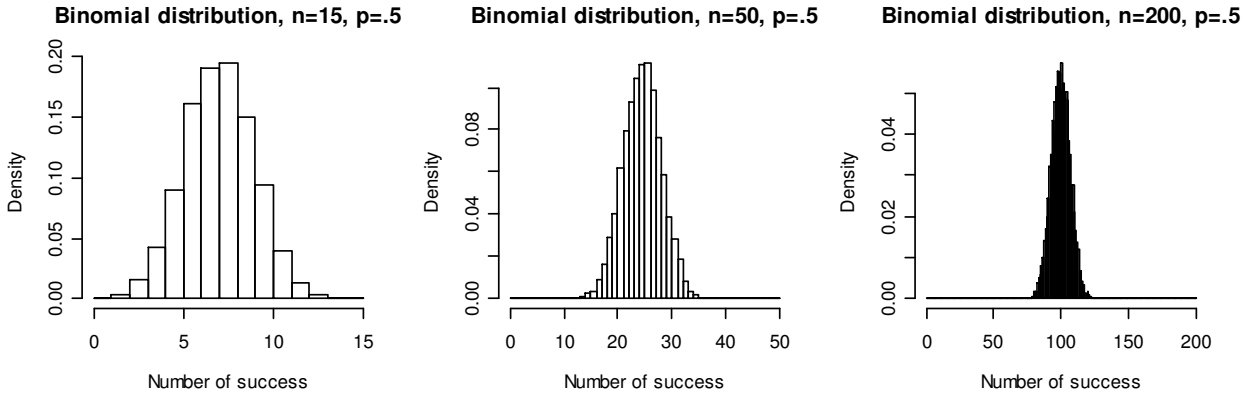
$$P(c_1 < \theta < c_2) = 0.95.$$

The interval (c_1, c_2) is said to be a 95% confidence interval for θ. In the case of the population median M, we want to find lower and upper limits M_L and M_U such that

$$P(M_L < M < M_U) = 0.95.$$

Suppose our sample will consist of n observations $X_1, X_2, \ldots, X_n$. We know that there is a 0.50 probability that any one of these observations will fall above (or below) the median M. We have already established that T, the number of items in the sample falling above the median, will be a random variable possessing the $Bin(n, 0.50)$ sampling distribution.

To get started on finding the boundaries of a confidence interval for M, we can take advantage of the fact that the $Bin(n, 0.50)$ sampling distribution is a symmetric distribution due to $p = 0.5$, i.e. it is just as likely for any observation to be above M as it is below it. (This is <u>not</u> true if $p \neq 0.5$.) To illustrate this point, here are pictures of three different binomial sampling distributions, all with $p = 0.5$:



We first sort the sample values in increasing order. Let $X^{(1)}, X^{(2)}, \ldots, X^{(n)}$ be the <u>ordered</u> X observations (i.e. $X^{(1)} < X^{(2)} < \cdots < X^{(n)}$). **Due to the symmetry, the endpoints of the CI for *M* will be the same number of values in from each end when the data values are put in sorted order.** This will result in a finite number of possible intervals to consider since there are only a finite number of observations in the sample. So the <u>only</u> possibilities for (M_L, M_U) are

$$(X^{(1)}, X^{(n)}),\ (X^{(2)}, X^{(n-1)}),\ (X^{(3)}, X^{(n-2)}),\ \ldots \text{ etc.}$$

Let's look at an example.

► **Example: Sleep patterns.** In a study of sleep patterns, Agnew et al. measured the percentage of total sleep time spent in stage 0 sleep by 16 mentally and physically healthy males between the ages of 50 and 60. Here are the data:

0.07	0.69	1.74	1.90	1.99	2.41	3.07	3.08
3.10	3.53	3.71	4.01	8.11	8.23	9.10	10.16

Find a 95% confidence interval for the true median percentage of time spent in stage 0 sleep.

We start by entering the data into an R vector, and then sorting it:

```
> x <- c(0.07, 0.69, 1.74, 1.90, 1.99, 2.41, 3.07, 3.08, 3.10, 3.53, 3.71,
+        4.01, 8.11, 8.23, 9.10, 10.16)
> x.sorted <- sort(x)
> x.sorted
 [1]  0.07  0.69  1.74  1.90  1.99  2.41  3.07  3.08  3.10  3.53  3.71  4.01
[13]  8.11  8.23  9.10 10.16
```

The possibilities for our CI for M are (0.07, 10.16), (0.69, 9.10), (1.74, 8.23), etc. The only thing left to do is to find the confidence levels associated with each of these intervals. We want as our answer the narrowest CI that achieves the stated confidence level.

Logic. To show how this works, let's start by considering the interval (0.07, 10.16). This is the widest possible interval and it will fail to cover M only if M is not within the range of the data. If this happens, then either all the sample values are above M or they are all below it, so the number of observations in the random sample that exceed M must be either $T = 0$ (none do) or $T = 16$ (they all do). Since $T \sim Bin(16, 0.5)$, this probability is

```
> dbinom(0,16,.5) + dbinom(16,16,.5)
[1] 3.051758e-05
```

Since this is the "failure to cover" probability, the coverage probability, or confidence level, is

```
> 1 - (dbinom(0,16,.5) + dbinom(16,16,.5))
[1] 0.9999695
```

The interval (0.07, 10.16) is a 99.99695% CI for M. This achieves our desired 95% confidence level, but it may be too wide.

What if we move to the next widest interval (0.69, 9.10)? Since it is narrower, the confidence level will be lower. But if it still above 95%, then it is a better interval than (0.07, 10.16). By using the same logic as applied above, the interval (0.69, 9.10) will fail to cover M only if $T \leq 1$ or $T \geq 15$. The coverage probability for (0.69, 9.10) is

```
> 1 - sum(dbinom(c(0,1,15,16),16,0.5))
[1] 0.9994812
```

Hopefully you see the trend developing. If we continue this process, we get these results:

Cutpoint locations	*Interval endpoints*	*Confidence level*
0 in from each end	(0.07, 10.16)	0.9999695
1 in from each end	(0.69, 9.10)	0.9994812
2 in from each end	(1.74, 8.23)	0.9958190
3 in from each end	(1.90, 8.11)	0.9787292
4 in from each end	(1.99, 4.01)	0.9231873

So, the 95% CI for M should be (1.90, 8.11). We can be at least 95% confident that the true median percentage of time spent in stage 0 sleep for healthy men aged 50 to 60 is between 1.9% to 8.11% of total sleep time. ◄

Some things to observe:

1. When n is small and the sampling distribution $Bin(n, 0.5)$ is highly discrete, the actual confidence level may be somewhat different than 95%. (A recent approach known as *fuzzy confidence intervals* developed by Charles Geyer and Glen Meeden at the University of Minnesota attempts to address this, but I will not elaborate on it here.)

2. The above method is somewhat conservative, i.e. it yields an interval that has a confidence level that will never be less than the stated confidence level. However, this can result in an interval that is wider (less precise) than necessary.

3. The procedure can be generalized for finding confidence intervals for percentiles other than the median (50^{th} percentile). Here is custom-built R function that automates the above process, and finds a generalized nonparametric CI for <u>any</u> percentile:

```
pctile.ci <- function( x, p = 0.5, conf.level = 0.95 ) {
  # Produces an exact confidence interval on the 100*pth percentile,
  #    based on the binomial test, where tied values are excluded.
  # `x' is the vector of observations.
  # `p' is the percentile of interest (e.g. p = 0.5 -> 50th percentile = median).
  # `conf.level' is the confidence level (between 0 and 1) for the returned CI.
  delta <- (max(x)-min(x))/1e10
  xgrid <- c(x, x+delta, x-delta)
  value.in.ci <- rep(NA, length(xgrid))
  for (iii in 1:length(xgrid)) {
     x1 <- c( sum( x<xgrid[iii] ), sum( x>xgrid[iii] ) ); n <- sum(x1)
     value.in.ci[iii] <-
        binom.test(x1, n, p, alternative = "two.sided", conf.level)$p.value>=1-conf.level
  }
  ci <- c( min( xgrid[value.in.ci] ), max( xgrid[value.in.ci] ) )
  result <- as.data.frame(list(percentile = p, lower = ci[1], upper = ci[2]))
  class(result) <- "table"
  result
}
```

Here is the earlier example executed after loading this R function:

```
> x <- c(0.07, 0.69, 1.74, 1.90, 1.99, 2.41, 3.07, 3.08, 3.10, 3.53,
+        3.71, 4.01, 8.11, 8.23, 9.10, 10.16)
> pctile.ci(x)

percentile      lower      upper
       0.5        1.9       8.11
```

4. The above procedure could break down (i.e. not be able to produce confidence limits) if *n* is too small and/or the requested percentile is too extreme. For example, it may be possible for a given data set to calculate a CI for the population median, but it may not be possible to find a CI for the 98th percentile.

5. These methods can be crude if *n* is small, but they are still useful.

7.3 CIs for medians/percentiles using a large-sample normal approximation

Recall that if $T \sim Bin(n, p)$ and both $np > 10$ and $n(1-p) > 10$, then the following "normal approximation" holds:

$$T \xrightarrow{approx} N\left(\mu = np, \sigma = \sqrt{np(1-p)}\right)$$

Thus with large samples, we can use a normal distribution's 95% capture "area" to determine where the boundaries of a 95% CI should be. Here is the procedure:

1. Check that both $np > 10$ and $n(1-p) > 10$.
2. Sort the observations into ascending order (denoted by $X^{(1)} < X^{(2)} < \cdots < X^{(n)}$).
3. To calculate a 95% CI for the p^{th} percentile of the population, find the following indices:
   - $L = np - 1.96\sqrt{np(1-p)}$. Round L up to the next highest integer.
   - $U = np + 1.96\sqrt{np(1-p)}$. Round U up to the next highest integer.
4. The 95% CI for the p^{th} percentile is $(X^{(L)}, X^{(U)})$.

► **Example: Crime rates.** A criminologist studying the relationship between level of education and crime rate in medium-sized US counties collected data for a random sample of $n = 84$ counties. Two variables were measured: the percentage of individuals in the county having at least a high school diploma, and the crime rate (reported as number of crimes per 100,000 residents). The data appear in the text file `crimerate.txt` in our repository. Find and interpret a 90% confidence interval for the third quartile (75th percentile) of the distribution of crime rates in all medium-sized US counties.

Solution. We read the text file into an R data frame named `crimerate`, check for the names of the two variables, and then extract the crime rate variable into its own vector:

```
> site <- "http://www.users.muohio.edu/hughesmr/sta333/crimerate.txt"
> crimerate <- read.table(site, header=TRUE)
> names(crimerate)
[1] "rate"        "pct.diploma"
> rate <- crimerate$rate
```

We now proceed step by step for the CI for the 75th percentile:

1. Check that both $np > 10$ and $n(1-p) > 10$:

```
> length(rate)*0.75 > 10
[1] TRUE
> length(rate)*(1-0.75) > 10
[1] TRUE
```

2. Sort the observations into ascending order:

```
> sort.rate <- sort(rate)
```

3. Find the ordered indices corresponding to the endpoints of the 90% CI:

```
> L <- length(rate)*0.75 + qnorm(0.05)*sqrt(length(rate)*0.75*(1-0.75))
> U <- length(rate)*0.75 + qnorm(0.95)*sqrt(length(rate)*0.75*(1-0.75))
> L
[1] 56.47219
> U
[1] 69.52781
> ceiling(c(L,U))     # the R function ceiling() always rounds up
[1] 57 70
```

4. Find the 90% CI:

```
 > indices <- ceiling(c(L,U))
 > sort.rate[indices]
[1] 8220 9697
```

We can be 90% confident that the 75th percentile of the crime rates for all medium-sized US counties is between 8220 to 9697 crimes per 100,000 residents. ◄

Note: Here is the same CI done using the exact (binomial) approach described in section 7.2:

```
> pctile.ci(rate,p=0.75,conf.level=0.90)
percentile      lower      upper
      0.75       8179       9697
```

So, the normal approximation was pretty good. The larger *n* gets, the better the approximation.

Lecture 7 Practice Exercises

Use R to do each of the following. Use R code instructions that are as general as possible, and also as efficient as possible.

1. Adult Americans sleep an average of 7.8 hours per day. You think college students sleep less than this on average, so you collect a random sample of fifteen college students at Miami and get the following data on their typical daily sleep amount (in hours):

6.7	4.5	6.4	8.6	5.5	8.2	5.9	7.5
4.4	6.0	6.3	8.3	7.3	5.7	10.1	

   a. Sort these observations from lowest to highest, and calculate the confidence level associated with using (4.5, 8.6) as a confidence interval for the true median amount of daily sleep for MU college students.

   b. Find a 90% confidence interval for M. Use the built-in R function `pctile.ci()`. Interpret the interval in context.

2. Refer to the `uwecsample` data frame in R, which contains current data on a sample from the UWEC undergraduate population. In particular, we are again interested in student high school percentile ranking (data frame variable `HSP`).

   a. Find a 95% confidence interval for the true **mean** high school percentile ranking for all UWEC undergrads. Interpret the CI in context.

   b. Find a 95% confidence interval for the true **median** high school percentile ranking for all UWEC undergrads. Use the built-in R function `pctile.ci()`. Interpret the CI in context.

   c. How can you explain the difference in the results between parts a and b?

   d. Find a 95% confidence interval for the 70^{th} percentile of the high school percentile rankings for all UWEC undergrads. (I know that sounds convoluted, but think about it a minute.) Interpret the CI in context.

STA333 Lecture 8
Tests for paired data: The sign test

8.1 Paired samples

We are now going to look at a couple of methods for analyzing *paired* (or *matched*) samples. We collect random samples from two populations in such a way so that each value in one sample may be meaningfully paired or matched with one specific value in the other sample. This is usually accomplished by measuring the same attribute twice (under two different conditions) on a single set of subjects.

► **Example: Monkey stimulation.** A physiologist wants to know if monkeys prefer stimulation of brain area *A* to stimulation of brain area *B*.

In the experiment, 14 rhesus monkeys are taught to press two bars. When a light comes on, presses on Bar 1 always result in stimulation of area *A*; and presses on Bar 2 always result in stimulation of area *B*. After learning to press the bars, the monkeys are tested for 15 minutes, during which time the frequencies for the two bars are recorded. The higher the frequency, the higher the preference.

The data appear at right. This is an example of paired data, because each subject (monkey) was measured twice.

Subject	*Bar 1*	*Bar 2*
1	20	40
2	18	25
3	24	38
4	14	27
5	5	31
6	26	21
7	15	32
8	29	38
9	15	25
10	9	18
11	25	32
12	31	28
13	35	33
14	12	29

8.2 Old stuff: paired samples *t*-test and CI using `t.test()`

In the conventional (parametric) approach to analysis of this type of data, the hypotheses of interest are one of the following comparisons of **population means**:

$$H_0: \mu_1 - \mu_2 = 0 \text{ vs. } H_a: \mu_1 - \mu_2 \neq 0 \quad \text{(two-sided test)}$$
$$H_0: \mu_1 - \mu_2 = 0 \text{ vs. } H_a: \mu_1 - \mu_2 > 0 \quad \text{(upper-tailed test)}$$
$$H_0: \mu_1 - \mu_2 = 0 \text{ vs. } H_a: \mu_1 - \mu_2 < 0 \quad \text{(lower-tailed test)}$$

where μ_1 = the true mean of the first population and μ_2 = the true mean of the second population. Recall that with paired data, we can form sample differences for each matched pair $d_i = x_{1i} - x_{2i}$. Then the difference in population means is estimated by $\bar{d}$, and we construct the standard error $\text{SE}_{\bar{d}}$ accordingly.

The *t* test statistic and 95% CI are given respectively by

$$t = \frac{\text{point estimate} - \text{hypothesized value}}{\text{standard error of point estimate}} = \frac{\bar{d} - 0}{\text{SE}_{\bar{d}}}$$

and

$$\bar{d} \pm t_{0.025} \times \text{SE}_{\bar{d}}$$

These may be done in R using `t.test()` using the `paired=TRUE` option.

► **Example: Anorexia therapy.** Anorexia nervosa is a serious eating disorder among young women. Data in the file `anorexiatherapy.txt` provide the weights (in pounds) of 17 anorexic young women both before and after receiving a family therapy treatment for anorexia nervosa. Does family therapy have a significant effect on mean weight of anorexic young women?

Solution. No direction is specified in the research question, so we will apply a two-sided test. After reading the file into an R data frame (which I named `anorexiatherapy`), we run the test:

```
> site <- "http://www.users.muohio.edu/hughesmr/sta333/anorexiatherapy.txt"
> anorexiatherapy <- read.table(site, header=TRUE)
> attach(anorexiatherapy)
> t.test(wt.before, wt.after, paired=TRUE)

        Paired t-test

data:  wt.before and wt.after
t = -4.1849, df = 16, p-value = 0.0007003
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -10.944712  -3.584700
sample estimates:
mean of the differences
              -7.264706
```

The data frame consists of two numeric columns (`wt.before` and `wt.after`). We invoke the `paired=TRUE` option in `t.test()` to perform a paired *t*-test. The *t* statistic value (on 16 df) is –4.185, and the *p*-value is 0.0007. Since $p < 0.05$, we reject H_0. There is a significant difference in mean weight before and after family therapy.

The corresponding 95% CI for $\mu_{\text{wt.before}} - \mu_{\text{wt.after}}$ is (–10.94 lb, –3.58 lb).

The interval for $\mu_{\text{wt.before}} - \mu_{\text{wt.after}}$ is entirely negative (–, –), so we can be confident that the true mean weight before the therapy is between 3.58 lb to 10.96 lb lower than after the therapy. ◄

But remember….

The paired t-test is a <u>parametric</u> test. Why? Because the validity of the findings depends on the normality assumption for the population of difference scores. In nonparametric statistics, however, we want to come up with a way of comparing the two populations (e.g. weights before therapy vs weights after therapy; stimulation level of brain area *A* vs brain area *B*; etc.) that does <u>not</u> require this assumption. That is the topic of this lecture.

8.3 The sign test for paired samples

The sign test is one of the simplest nonparametric tests. It is for use with paired samples of repeated measurements on the same subjects (as in the two examples given above). The only assumptions the sign test has are:

1. The sample is collected randomly. The data are in the form of (X, Y) pairs, where:
   - X_i is the value from the first population in the i^{th} pair
   - Y_i is the value from the second population in the i^{th} pair
2. The measurement scale is at least **ordinal**. (This means that individual values at least have some sense of ordering of magnitude).

Logic. For each subject, subtract the 2nd score from the 1st, and then write down the *sign* of the difference. (That is, write "–" if the difference score is negative, and "+" if it is positive.) If the values are the same, they are said to be **tied**: usually we delete such pairs from the test ... but more on that later.

$X_i - Y_i > 0 \quad \rightarrow$ assign a (+) to the pair
$X_i - Y_i < 0 \quad \rightarrow$ assign a (–) to the pair
$X_i - Y_i = 0 \quad \rightarrow$ the values are tied: usually delete this pair from the analysis

The usual null hypothesis for this test is that there is no difference between the two population medians M_X and M_Y. If this is so, then the number of + signs (or – signs, for that matter) should have a **binomial sampling distribution** with n = number of subjects and $p = .5$. In other words, the sign test is just a binomial test with + and – in place of "success" and "failure".

Hypotheses. The hypotheses to be considered are

$H_0: M_X = M_Y$ *vs.* $H_a: M_X \neq M_Y$ (two-sided test)
$H_a: M_X > M_Y$ (upper-tailed test)
$H_a: M_X < M_Y$ (lower-tailed test)

As always, pick the appropriate alternative H_a based on the relevant research question.

Test statistic. Let T = the number of +'s in the random sample.

Null distribution of the test statistic. If the null hypothesis H_0 were true, then $T \sim Bin(n, 0.50)$, where n = the number of untied pairs.

p-value. By definition, the p-value for any hypothesis test is the probability of seeing sample evidence at least as contradictory to H_0 (and in favor of H_a) as that which you observed in the actual sample. In this case, the p-value will be some probability from the $Bin(n, 0.50)$ null distribution. We then can compare this to some pre-determined significance level α.

The sign test for paired samples is just the binomial test with $p = 0.5$.

► **Example: Monkey stimulation.** A physiologist wants to know if monkeys prefer stimulation of brain area *A* to stimulation of brain area *B*. The researcher did not predict a particular outcome in this case, but wanted to know if the two conditions differed. Therefore, the alternative hypothesis is non-directional, i.e., two-sided:

H_0: There is no monkey preference between stimulation areas *A* and *B*
H_a: There is a preference between stimulation areas *A* and *B*

- Since the data are paired and numeric, a sign test comparing the medians may be used to test the hypotheses.
- A middling number of + signs (in the neighborhood of $np = 14(0.5) = 7$) would be consistent with the null hypothesis H_0.
- Too many or two few +'s would be consistent with H_a in this case.

We build the test statistic:

Subject	*Bar 1*	*Bar 2*	*Difference*	*Sign of difference*
1	20	40	–20	–
2	18	25	–7	–
3	24	38	–14	–
4	14	27	–13	–
5	5	31	–26	–
6	26	21	+5	+
7	15	32	–17	–
8	29	38	–9	–
9	15	25	–10	–
10	9	18	–9	–
11	25	32	–7	–
12	31	28	+3	+
13	35	33	+2	+
14	12	29	–17	–

So, $T = 3$. There are no ties, so the null distribution of T is $Bin(14, 0.5)$. We show how we can do all this in R using the `binom.test()` function:

```
> x <- c(20,18,24,14,5,26,15,29,15,9,25,31,35,12)
> y <- c(40,25,38,27,31,21,32,38,25,18,32,28,33,29)
> d <- x-y
> d
 [1] -20  -7 -14 -13 -26   5 -17  -9 -10  -9  -7   3   2 -17
> T <- length(d[d>0])
> T
[1] 3
> binom.test(T, length(d[d != 0]), alternative="two.sided")
```

```
        Exact binomial test

data:  T and length(d[d != 0])
number of successes = 3, number of trials = 14, p-value = 0.05737
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.04657929 0.50797568
sample estimates:
probability of success
             0.2142857
```

The *p*-value for the test is 0.0574, which is borderline statistically significant. That is, there is marginally significant evidence to suggest that there is a preference in monkeys between stimulation areas *A* and *B*. Since the number of +'s was small, this suggests that Bar 2 had a higher frequency, and thus area *B* was preferred. ◄

8.4 Ties and the "zero fudge"

What if you have tied *X* and *Y* observations for a given subject? Many sources recommend the following procedure for dealing with zero differences in the sign test. After defining a vector `d` of differences, we can do this:

```
d <- d[d != 0]
n <- length(d)
```

This approach treats zero differences as if they were not part of the data (and the sample size *n* is reduced accordingly). This is called the **zero fudge**.

The best that can be said for this is:

- Most nonparametric statistics books recommend it (or at least describe it first).

- From a theoretical point of view, it is a valid test of the hypotheses

$$H_0: P(X_i < Y_i) = P(X_i > Y_i)$$
$$H_a: P(X_i < Y_i) \neq P(X_i > Y_i)$$

  (or the analogous one-sided alternatives). But these hypotheses are *not what you want to test!* What you want to test is the hypotheses that the medians are the same or different.

  To illustrate the point, consider modifying the data in the monkey example by adding a million zero differences to the data set. The "zero fudge" says we throw out those zeros and do exactly the same analysis getting *p*-value = 0.0574, a marginally significant result. But the whole data set says we get *exactly the same response* in the two brain areas in 1,000,000 cases and a different response in only 14 cases. This is overwhelming evidence

in favor of the null hypothesis comparing the medians. It is highly significant evidence *against* the tricky null hypothesis of the "zero fudge" test.

The moral of the story: In interpreting a test of significance, it's not enough that we have a p-value < 0.05. It's even more important *what the null hypothesis is*. Rejecting a null hypothesis of no scientific interest whatsoever is worthless.

Hence the zero fudge is a form of "honest cheating". Widely accepted, but bogus. The reason everyone likes it is because it produces a p-value < 0.05 more often than other strategies, and everyone likes to get "statistically significant" results … even if bogus.

So, what to do? Well, even after all that, applying the "zero fudge" when the number of ties is really small (say, less than 5% of the data) is usually OK. (In other words, no one notices if you cheat just a little.)

There are other approaches to handle cases where there are many ties. Consider these options:

- If there are only 2 subjects with tied scores, make one a (+) and one a (–).
- In general, if there is an even number of subjects with tied scores, make half of them (+) signs, and half (–) signs.
- **Jittering.** Ties are usually the result of dealing with responses that are measured on a discrete scale. So, a more savvy strategy to get around this might be to just *add an infinitesimally small random shift* (or "jitter") *of your own to every observation*, where there is a 50/50 chance that the jitter would be positive or negative. For example, consider the monkey data again:

```
> x <- c(20,18,24,14,5,26,15,29,15,9,25,31,35,12)
> y <- c(40,25,38,27,31,21,32,38,25,18,32,28,33,29)
> d <- x-y
> d
[1] -20  -7 -14 -13 -26   5 -17  -9 -10  -9  -7   3   2 -17
```

Let's add some jitter:

```
> d <- d + rnorm(length(d), mean=0, sd=1e-5)    # add random jitter
> d
[1] -19.999998  -7.000000  -13.999983  -13.000007  -26.000020  4.999998
[7] -17.000000  -8.999992  -10.000011  -8.9999830   -6.999997  2.999995
[13]  1.999998 -17.000013
```

Now run the sign test, but on the jittered data:

```
> T <- length(d[d>0])
> T
[1] 3

> binom.test(T, length(d), alternative="two.sided")
```

```
        Exact binomial test

data:  T and length(d)
number of successes = 3, number of trials = 14, p-value = 0.05737
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.04657929 0.50797568
sample estimates:
probability of success
             0.2142857
```

Note that we have changed each of the `d` vector's entries very slightly by adding random amounts symmetric around 0, so ties should no longer be a problem (not that they were before for these data). The sign test result is the same here. If we had ties, however, all the subjects would now participate in the test.

Also, you should take note of the difference in the `binom.test()` statement between the "jitter" approach and the "zero fudge" approach.

Lecture 8 Practice Exercise

We will use the sign test as a mechanism to compare the taste of two different kinds of cookie (labeled *A* and *B* … identity to be revealed later). We proceed with the experiment as follows:

1. Each participant will receive two cookies labeled *A* and *B*. You do not know the "identities" of the cookies from the labeling (i.e. this is a **blind** experiment).
2. The order in which you eat the cookies will be randomized. Some will eat *A* first and then *B*, the rest *B* then *A*.
3. After eating each cookie, rate the taste on a scale from 0 to 10 (0=lowest possible taste score, 10=highest possible taste score)

My taste rating for cookie A :

My taste rating for cookie B:

Testing the hypothesis to compare the taste of the two types of cookie:

What are the population parameters (characteristics) of interest?

What are the hypotheses of interest?

H_0: H_a:

What is a useful test statistic for testing these hypotheses?

What is the null distribution of the test statistic?

Using the null distribution, calculate the p-value for the test.

State your conclusion.

STA333 Lecture 9
Tests for paired data: Wilcoxon Signed-Ranks Test

9.1 Wilcoxon signed-ranks test

Let's revisit the monkeys:

► **Example: Monkey stimulation.** A physiologist wants to know if monkeys prefer stimulation of brain area *A* to stimulation of brain area *B*.

In the experiment, 14 rhesus monkeys are taught to press two bars. When a light comes on, presses on Bar 1 always result in stimulation of area *A*; and presses on Bar 2 always result in stimulation of area *B*. After learning to press the bars, the monkeys are tested for 15 minutes, during which time the frequencies for the two bars are recorded. The higher the frequency, the higher the preference.

The data appear at right. This is an example of paired data, because each monkey was measured twice. The data are in the form of (X, Y) pairs.

Subject	*Bar 1 (X)*	*Bar 2 (Y)*
1	20	40
2	18	25
3	24	38
4	14	27
5	5	31
6	26	21
7	15	32
8	29	38
9	15	25
10	9	18
11	25	32
12	31	28
13	35	33
14	12	29

We did the sign test on these data in the last lecture.

One obvious problem with the sign test is that it **discards** a lot of information about the data. It takes into account the *direction* of the difference, but not the *magnitude* of the difference between each pair of scores.

At this point, I will discuss another method for comparing two populations using paired samples. This new method, called the **Wilcoxon signed-ranks test**, is *not* a binomial-based test, so we will have to think of things a little differently.

The **Wilcoxon signed-ranks test** is another nonparametric test that can be used for paired samples of repeated measurements on the same subjects. Unlike the sign test, it *does* take into account (to some degree, at least) the *magnitude* of the measurements.

Nature of the test

Suppose we have a random sample of n pairs of measurements (X_1, Y_1), (X_2, Y_2), …, (X_n, Y_n) measured on an interval scale. Interval scale measurements are measurements where it is meaningful to think of the distance between measurements. Let $D_i = X_i - Y_i$, the difference score for each pair (just like in the sign test).

Hypotheses. The hypotheses to be tested are identical to those in the sign test in Lecture 8, i.e. we are comparing the true medians M_X and M_Y of the two populations:

$$H_0: M_X = M_Y \quad vs. \quad \begin{array}{l} H_a: M_X \neq M_Y \text{ (two-sided test)} \\ H_a: M_X > M_Y \text{ (upper-tailed test)} \\ H_a: M_X < M_Y \text{ (lower-tailed test)} \end{array}$$

As always, you pick the appropriate alternative H_a based on the relevant research question.

Motivation and derivation of a test statistic. Let's use the same monkey stimulation data we used to illustrate the sign test. The 14 difference scores (D_i's) were:

$$-20, -7, -14, -13, -26, +5, -17, -9, -10, -9, -7, +3, +2, -17$$

First, delete any observation where $D_i = 0$. From those remaining, we sort these D_i values *on the basis of their absolute values* (i.e., disregarding the sign) and then assign **ranks** to each value. This results in the following:

D_i	+2	+3	+5	–7	–7	–9	–9	–10	–13	–14	–17	–17	–20	–26
Rank	1	2	3	4.5	4.5	6.5	6.5	8	9	10	11.5	11.5	13	14

Rank 1 goes with the smallest value (in absolute value), rank 2 with the next largest, etc. If two values have the same difference score, we usually assign **average ranks** to those observations. See the next section for more on this.

The **Wilcoxon signed-rank statistic** V is the
sum of the ranks associated with the positive difference scores.

In this example, $V = 6$ (i.e., 1 + 2 + 3).

► **In-class exercise: Mileage comparisons for 5-day vs 4-day work week.** Do flexible schedules reduce the demand for resources? The Lake County, Illinois Health Department experimented with a flexible four-day workweek. For a year, the department recorded the mileage driven by 7 field workers on an ordinary five-day workweek. Then it changed to a flexible four-day workweek and recorded mileage for another year. The goal of the study is to determine if there is less driving as a result of the four-day schedule.

Employee	**5-day mileage**	**4-day mileage**	**Difference D_i**	**Absolute difference $\lvert D_i \rvert$**	**Ranks of $\lvert D_i \rvert$**	**Ranks of $\lvert D_i \rvert$ where sign is +**
Jeff	2798	2914				
Betty	7724	6112				
Roger	7505	6177				
Tom	838	1102				
Aimee	4592	3281				
Greg	8107	4997				
Larry	1228	1695				

Calculate the Wilcoxon signed-rank statistic V.

OK, now we have to figure out how to get a p-value. To do so, we should first get a handle on how V behaves under different circumstances:

- **If the null hypothesis H_0: $M_X = M_Y$ were true,** the sum of the "positive" ranks and the sum of the "negative" ranks would be roughly equal. In this case, $V \cong n(n+1)/4$.
- **If (for instance) H_a: $M_X > M_Y$ were true,** it would have resulted in seeing a prevalence of pairs with $X_i > Y_i$, and hence more pairs where $D_i > 0$. So, the larger ranked differences would tend to be positive, and thus we'd see the value of V increase.
- Similar arguments can be made for lower-tailed and two-tailed alternatives.

Null distribution of the test statistic V. (Just a reminder: it ain't binomial). The sampling distribution of V can be obtained by looking at all possible arrangements of (+) and (–) signs, and then calculating the sum of the ranks associated with the (+) scores for every possible arrangement. If the null hypothesis H_0 is true, then every possible arrangement of (+) and (–) signs is equally likely.

► **In-class exercise.** Suppose we have $n = 4$ (X, Y) pairs with no tied ranks, so we have four unique values of D_i. Find the complete null distribution of V = sum of the ranks associated with the positive difference scores.

p-value. By definition, the p-value for any hypothesis test is the probability of seeing sample evidence at least as contradictory to H_0 (and in favor of H_a) as that which you observed in the actual sample. In this case, the p-value will be some probability from the null distribution of V.

We will let R compute this for us. We then can compare this to some pre-determined significance level α.

Assumptions.

1. The sample is random.
2. The measurement scale for the items in the populations is continuous.
3. The distribution of differences is symmetric.

9.2 Issues, issues: dealing with ties and zeros

Assumption 2 essentially insures that all the observations will be unique, i.e. no duplicate values anywhere. But of course, people round off values, use discrete scales, etc. That's the real world. So what happens if the continuity assumption is false and there are zero D_i values, or tied absolute D_i values?

There are two ways in which a violation of the continuity assumption can affect results:

- **The zero fudge.** What we called the zero fudge in the context of the sign test (because it is fairly bogus there) makes much more sense in the context of the signed rank test. In the Wilcoxon signed-ranks test, zero values for the $D_i = X_i - Y_i$, differences should get the smallest possible absolute rank, because zero is smaller in absolute value than any other number. We might as well give them rank zero, starting counting at zero rather than at one. Then they make no contribution to the sum of ranks statistic V.

- **Tied ranks.** The preceding issue deals with instances where an individual gives a pair of values that are identical. But there's a second kind of violation that causes a different problem: what if we have two individuals with tied D_i values? (Take a look at the monkey data: this situation is pretty prevalent there.)

  Assigning ranks in this situation is not a problem: as stated before, it is customary to deal with tied ranks in the following manner: Give the tied scores the *mean of the ranks they would have if they were not tied*.

  For example, if you have two tied scores that would occupy positions 3 and 4 if they were not tied, give each one a rank of 3.5. If you have 3 scores that would occupy positions 7, 8, and 9, give each one a rank of 8. This procedure is preferred to any other for dealing with tied scores, because the sum of the ranks for a fixed number of scores will be the same regardless of whether or not there are any tied scores. In fact, the built-in R function `rank()` does this by default:

```
> d <- c(12,43,43,19)
> rank(d)
[1] 1.0 3.5 3.5 2.0
```

  However, the main problem that arises is that the null distribution of V changes in the presence of ties. Because of this, the p-value for the test may change. Since this could potentially alter the outcome of the test, it is pretty important to account for!

  ► **Exercise (on your own).** Suppose we have $n = 4$ (X, Y) pairs as before, but instead we have one tied pair of ranks between the 2nd and 3rd observations, so that the ranks were instead 1, 2.5, 2.5, 4. Find the null distribution of V if this is the case, and see how the null distribution differs from what we got in the no-tied-ranks case.

9.3 Wilcoxon signed-ranks test using R

We can perform the correct Wilcoxon signed-ranks test in R using the `wilcox.exact()` function in the R add-on package `exactRankTests`. *You must first download and install this add-on to your R installation in order to use it.* See section 1.3 for more info. You only need to install it once, and then you have it for good.

► **Example: Monkey stimulation.** The hypotheses are

$$H_0: M_X = M_Y \quad \text{vs.} \quad H_a: M_X \neq M_Y$$

where M_X = the true median number of presses on bar 1, and M_Y = the true median number of presses on bar 2. The null hypothesis is essentially stating that there is no preference between the bars, but the signed-ranks test expresses this hypothesis in term of the difference in medians.

```
> x <- c(20,18,24,14,5,26,15,29,15,9,25,31,35,12)
> y <- c(40,25,38,27,31,21,32,38,25,18,32,28,33,29)
> library(exactRankTests)
> wilcox.exact(x, y, paired=TRUE, alternative="two.sided")

        Exact Wilcoxon signed rank test

data:  x and y
V = 6, p-value = 0.001465
alternative hypothesis: true mu is not equal to 0
```

The `library(exactRankTests)` command loads the package into the current R session. The fourth line above runs the two-sided test.

The test statistic value is $V = 6$, and the two-sided alternative's p-value is 0.0014. Since the p-value < 0.05, there is significant evidence of a preference. The preference is for area *B*. ◄

Important note: there is another built-in function for running the Wilcoxon signed-ranks test in R's base package named `wilcox.test()`. It is more widely used, but it ignores ties when calculating the p-value. If you have ties, `wilcox.test()` gives you the wrong p-value:

```
> # NOT RECOMMENDED!!!!
> wilcox.test(x, y, paired=TRUE, alternative="two.sided")

        Wilcoxon signed rank test with continuity correction

data:  x and y
V = 6, p-value = 0.003854
alternative hypothesis: true location shift is not equal to 0

Warning message:
In wilcox.test.default(x, y, paired = TRUE, alternative = "two.sided") :
  cannot compute exact p-value with ties
```

At least it warns you it's screwing up, but it still gives a screwed-up result!

9.4 Some comments

Here are some observations about the Wilcoxon signed-ranks test that you should be aware of:

1. The conclusion we make when running the Wilcoxon signed-ranks test is a stronger one than our conclusion when we analyzed the same data using the sign test. **The reason is that the Wilcoxon signed-ranks test is more *powerful* than the sign test.** Why is it more powerful? Because it makes use of more information than does the sign test. But, note that it too does discard some information by using *ranks* rather than the scores themselves (like the paired *t*-test does).

   Here's a look at the three competing procedures:

***t*-tests (parametric)**	**Sign test (Lecture 8)**	**Signed-ranks test**
• *Uses the actual numbers, both size and sign* • *Highly non-robust (breakdown point 0%).*	• *Completely ignores the size of the numbers, using only their sign (i.e., "Are the differences positive or negative?").* • *Highly robust (breakdown point 50%).*	• *Uses both size and sign, but doesn't use the actual magnitudes of the numbers, only the ranks of their magnitudes* • *Moderate robustness (breakdown point 29.3%).*

2. *Symmetry.* We added the assumption of symmetry in the difference scores to this procedure (which was *not* an assumption for the sign test). This has two implications:

   a. If a distribution is symmetric, the mean coincides with the median because both are located exactly in the middle of the distribution (at the line of symmetry). So, adding the assumption of symmetry implies that any inferences concerning the median are also valid statements for the mean.

   b. A second consequence of adding the assumption of symmetry is that the required scale of measurement is changed from ordinal to interval. With an ordinal scale of measurement, two observations need only be distinguished on the basis of which is larger and which is smaller. It is not necessary to know which one is farthest from the median, such as when two observations are on opposite sides of the median. If the assumption of symmetry is a meaningful one, then the distance from the median is a meaningful measurement and, therefore, the distance between two observations is a meaningful measurement. As a result, the scale of measurement is more than just ordinal, it is interval.

   c. *Coding.* Sometimes in practice, people will score an ordinal attribute using a numeric coding scale (e.g. "on a scale from 0 to 10…"). It is not uncommon in such instances to treat the numeric coding as though there were measured on an interval scale.

3. *What the…?* You may be wondering why R has a `wilcox.test()` function that does the wrong thing and a `wilcox.exact()` function that does the right thing. My guess is because the wrong `wilcox.test()` function has been around for several years and uses decades-old ideas. It's tradition. However, `wilcox.exact()` uses ideas that are very recent research. It may eventually replace the other function, but has to wait a while.

4. *Large sample approximation.* Why use an approximation when the computer can do it exactly? Well, because even a computer has a hard time (if not an impossible time) trying to do it exactly for really large n. The reason is because it's a herculean task to try to figure all the values of V for all the possible arrangements of (+) and (–) signs for a big n.

   So, when $n > 50$, `wilcox.exact()` defaults to using a Central Limit Theorem-based asymptotic approximation for the p-value:

```
> x <- rnorm(200,12,2)
> y <- rnorm(200,12.5,3)
> wilcox.exact(x, y, paired=TRUE, alternative="less")

        Asymptotic Wilcoxon signed rank test

data:  x and y
V = 8287, p-value = 0.01573
alternative hypothesis: true mu is less than 0
```

Lecture 9 Practice Exercise

Here are data from an Oreo comparison taste test done in a previous semester of STA333. Use it and R to run a Wilcoxon signed-ranks test to see if there is a difference in the median taste test rating between regular and low-fat Oreo cookies.

Subject	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Regular	3	7	5	7	8	10	9	6	10	5	9	8	8	8	6	5	6	8	7
Low-fat	6	4	8	8	6	6	5	4	7	7	5	7	6	6	8	7	4	6	5

STA333 Lecture 10
Tests for paired data: McNemar's Test

10.1 Correlated proportions

We now consider (finally) a final variation of the binomial test. In this one, the paired data are dichotomous, that is, it is in the form of categorical responses where one may be classified into one of two categories. These classifications are typically measured on the same individuals at two points in time to see if a change in opinion or status has occurred. Here is an example.

► **Example: Obama approval.** The Gallup organization occasionally runs tracking polls, where the same individuals are polled at different times to assess if they have changed their opinion regarding some political or social issue. 1600 randomly selected persons were polled regarding their approval or disapproval of Barack Obama both immediately after the election and again after the passage of the economic stimulus package of 2009. The results of the two surveys are given in the table below.

		Post-stimulus package passage survey		
		Approve	*Disapprove*	**Total**
Post-election survey	*Approve*	794	150	**944**
	Disapprove	86	570	**656**
	Total	**880**	**720**	**1600**

Is there sufficient evidence to conclude that Obama's approval rating has significantly dropped following passage of the stimulus bill? Test using $\alpha = 0.05$.

We'll come back to this problem later after laying out some basics in the next section.

10.2 Structure of the test

McNemar's Test assesses the significance of the difference between two correlated proportions, such as might be found in the case where the two proportions are based on the same sample of subjects or on matched-pair samples.

An illustration with some numbers. Suppose that 100 subjects are each assessed with respect to two dichotomous categorical variables, *A* and *B*. If the temporal sequence of the two responses is relevant, variable *A* can be defined as a "before" response, and variable *B* as an "after" response. The results may be coded as **1** for those subjects that display the property defined by the variable in question and as **0** for those that do not display that property.

Here are some phony data for illustration:

		Subject's response on variable *B*		
		1	**0**	**Total**
Subject's response on variable *A*	**1**	25	5	**30**
	0	15	55	**70**
	Total	**40**	**60**	**100**

The sample marginal proportions in this example are $p_A = 30/100 = 0.30$, and $p_B = 40/100 = 0.40$. In other words, 30% of the subjects display the characteristic defined by variable *A,* and 40% display the characteristic defined by variable *B*.

Generalization. We can generalize the structure in the counts in the above example as follows:

		Subject's response on variable *B*		
		1	**0**	**Total**
Subject's response on variable *A*	**1**	*a*	*b*	***a + b***
	0	*c*	*d*	***c + d***
	Total	***a + c***	***b + d***	***n = a + b + c + d***

McNemar's Test examines the difference between the *proportions* that derive from the marginal sums of the table:

$$\hat{p}_A = (a + b)/n$$
$$\hat{p}_B = (a + c)/n$$

The question in McNemar's Test is: do the true proportions p_A and p_B significantly differ? And the answer must take into account the fact that the two proportions are *not* independent. Why? Because the correlation of $\hat{p}_A$ and $\hat{p}_B$ is induced by the fact that both of them include the quantity *a* in the upper left cell of the table. In other words, *a* subjects out of the sample of *n* subjects responded with a **1** response to both variables.

The core insight of McNemar's test is two-fold:

1. The difference between $\hat{p}_A$ and $\hat{p}_B$ reduces entirely, both algebraically and conceptually, to the difference between b and c in the off-diagonal cells of the table.
2. b and c belong to a binomial distribution with $n = b + c$ and $p = 0.5$. In other words, the exact null distribution for testing H_0: $p_A = p_B$ is the $Bin(b + c, 0.5)$ distribution. (Can you reason why?)

Details

Hypotheses. The parameters of interest are p_A and p_B, the true population proportions giving the response of interest at two different times A and B. The hypotheses to be considered are

H_0: $p_A = p_B$ vs. H_a: $p_A > p_B$ (upper-tailed test)
H_0: $p_A = p_B$ vs. H_a: $p_A < p_B$ (lower-tailed test)
H_0: $p_A = p_B$ vs. H_a: $p_A \neq p_B$ (two-sided test)

Test statistic. We could actually choose from several different valid test statistics here, but the most commonly used version in practice is a large-sample approximation statistic:

$$\chi^2 = \frac{(b-c)^2}{b+c}$$

This type of statistic is known as a **chi-square statistic** and is frequently found in statistics when analyzing count data. But please note the following:

- This test statistic works well as long as both b and c are greater than 10, and can only be used to test the two-sided alternative hypothesis.
- If either b or c are too small, or if you are testing a one-sided alternative hypothesis, you should base your finding on the binomial distribution instead of the chi-square.

p-value. By definition, the p-value for any hypothesis test is the probability of seeing sample evidence at least as contradictory to H_0 (and in favor of H_a) as that which you observed in the actual sample. In this case, the p-value will be some probability from the chi-square distribution with one degree of freedom, or the $Bin(b + c, 0.5)$ distribution if using the exact form. We will let R compute this for us in either case. We then can compare this to some pre-determined significance level α.

Assumptions.

1. The sample is random.
2. The measurement scale is at least nominal (i.e. variable values differ at least in kind, but not magnitude).

10.3 Example using R

► **Example: Obama approval.** 1600 randomly selected persons were polled regarding their approval or disapproval of Barack Obama both immediately after the election and after the passage of the economic stimulus package of 2009. The results of the two surveys are given in the table below.

		Post-stimulus package passage survey		
		Approve	*Disapprove*	**Total**
Post-election survey	*Approve*	794	150	**944**
	Disapprove	86	570	**656**
	Total	**880**	**720**	**1600**

Is there sufficient evidence to conclude that Obama's approval rating has significantly dropped following passage of the stimulus bill? Test using $\alpha = 0.05$.

An exact solution. The correct hypotheses to test are

$$H_0: p_A = p_B \quad \text{vs.} \quad H_a: p_A > p_B$$

where p_A = Obama's true approval rating immediately following the election, and p_B = Obama's true approval rating immediately following passage of the stimulus package. Our point estimates of these parameters show that there was an observed 4% drop in his sample approval rating:

$$\hat{p}_A = \frac{794+150}{1600} = 0.59 \qquad \hat{p}_B = \frac{794+86}{1600} = 0.55$$

Is this statistically significant? To answer this exactly, we recognize that if the null hypothesis H_0: $p_A = p_B$ were true, then the 150 + 86 = 236 "changed" opinions would have been equally distributed between those who changed from approval to disapproval (table cell *b*) and those who changed from disapproval to approval (table cell *c*). "Equal" distribution means we would have expected 236/2 = 118 in each of these cells. As it stands, we see many more in cell *b* (150) than we would have expected if the null hypothesis were true. So, we can calculate the *p*-value for the test as follows:

$$\begin{aligned} p\text{-value} &= \text{probability of seeing sample evidence at least as favorable to } H_a \text{ as that which we observed in the actual sample, given } H_0 \text{ true} \\ &= P(X \geq 150) \text{ where } X \sim Bin(236, 0.5) \end{aligned}$$

We know how to find this is R:

```
> 1 - pbinom(149, 236, 0.5)
[1] 1.857968e-05
```

The exact p-value is 0.00001857, so we safely reject H_0. There is significant evidence that Obama's approval rating in the entire population has dropped since just after the election. ◄

Large sample approximation solution. The correct hypotheses to test are the same as before:

$$H_0: p_A = p_B \quad \text{vs.} \quad H_a: p_A > p_B$$

We can use the built-in R function `mcnemar.test()` to run the large sample approximation test. Keep in mind that this only tests the two-sided alternative, so we will need to divide the given p-value by 2 to get the correct p-value.

We need to first enter all the counts in the table into an R matrix. I do so below, print out the table, and then run the `mcnemar.test()` function:

```
> rate.bo <- matrix(c(794, 86, 150, 570),
+            nrow = 2,
+            dimnames = list("1st Survey" = c("Approve", "Disapprove"),
+                            "2nd Survey" = c("Approve", "Disapprove")))
> rate.bo

            2nd Survey
1st Survey   Approve Disapprove
  Approve        794        150
  Disapprove      86        570

> mcnemar.test(rate.bo, correct=FALSE)

        McNemar's Chi-squared test

data:  rate.bo
McNemar's chi-squared = 17.3559, df = 1, p-value = 3.099e-05
```

The given p-value is for the two-sided alternative. Since our test is strictly looking to ask if Obama's approval rating has dropped (a one-sided alternative), divide the given p-value by 2:

$$p\text{-value} = 0.00003099/2 = 0.000015495$$

This is very close to the exact p-value obtained by the binomial distribution. Again, we reject H_0. ◄

10.4 A confidence interval for $p_A - p_B$

We can estimate the size of the "gap" between p_A and p_B using a CI. This can be more informative than the result of a hypothesis test alone.

The CI is only reasonable when the sample size n is large enough to use the large sample approximation statistic. **The formula uses a standard error that recognizes the fact that the two proportions are correlated.**

We use $\hat{p}_A$ and $\hat{p}_B$ from before. Referencing the quantities a, b, c and d from the table layout in section 10.2, we can write the CI formula as follows:

$$(\hat{p}_A - \hat{p}_B) \pm z_{\alpha/2}\sqrt{\frac{\hat{p}_A(1-\hat{p}_A) + \hat{p}_B(1-\hat{p}_B) - 2(\frac{ad-bc}{n^2})}{n}}$$

I know … yuck. But it isn't that bad to code in R. Here's an example to illustrate.

► **Example: Obama approval**. In R, build a 90% CI for $p_A - p_B$, the proportional loss in approval for Obama from post-election to post-economic stimulus package passage. As a reminder, here's the tabled data:

		Post-stimulus package passage survey		
		Approve	*Disapprove*	**Total**
Post-election survey	*Approve*	$a = 794$	$b = 150$	**944**
	Disapprove	$c = 86$	$d = 570$	**656**
	Total	**880**	**720**	**1600**

```
> a <- 794
> b <- 150
> c <- 86
> d <- 570
> n <- a+b+c+d                                          # compute n
> Pa <- (a+b)/n                                        # compute Pa-hat
> Pb <- (a+c)/n                                        # compute Pb-hat
> se <- sqrt((Pa*(1-Pa)+Pb*(1-Pb)-2*(a*d-b*c)/n^2)/n)   # compute std error
> lower.limit <- (Pa-Pb) + qnorm(0.05)*se              # compute lower limit
> upper.limit <- (Pa-Pb) + qnorm(0.95)*se              # compute upper limit
> ci <- cbind(lower.limit, upper.limit)                # bind into 1 object
> ci                                                   # print CI
     lower.limit upper.limit
[1,]  0.02429294  0.05570706
```

We can be 90% confident that Obama's approval rating has dropped between 2.4% to 5.5%. ◄

Lecture 10 Practice Exercise

A lab experiment is done to compare two different media (*A* and *B*) for culturing Mycobacterium tuberculosis. The lab researcher believes that medium *B* is more conducive to culture growth than medium *A*.

Fifty suspect sputum specimens were divided and plated up on both media, and it was observed if culture growth occurred. Of the 50 specimens, 20 experienced growth on both *A* and *B*, 16 experienced no growth on either *A* or *B*, 2 experienced growth on *A* but not on *B*, and 12 experienced growth on *B* but not on *A*.

You will use this information to run a statistical test to try to verify the researcher's claim. Answer the following:

1. Build a two-way table displaying the information above.

2. Run McNemar's test at $\alpha = 0.05$ to see if there is sufficient evidence to conclude that medium *B* is more likely to produce culture growth than medium *A*. Use the binomial distribution to find your *p*-value.

3. Repeat part 2, but use the R function `mcnemar.test()`

STA333 Lecture 11
Introduction to permutation tests (part 1)

11.1 Quick review of the "classic" two-sample t-test

We now move onto problems involving the comparison of two populations, where we collect two independent samples (one from each population). There are many nonparametric tests for such situations, all of which are a form of what is known as a *permutation test*. First, however, let's revisit the classic two-sample problem from introductory statistics, the independent samples *t*-test.

► **Example: Diet comparisons.** In a comparison of the effect on growth of two diets ***A*** and ***B***, a number of growing rats were placed on these two diets, and the following growth figures were observed after 7 weeks:

A	156	183	120	113	138	145	142			
B	109	107	119	162	121	123	76	111	130	115

Use a two-independent samples *t*-test to see if mean growth rate differs between the two diets. Test using $\alpha = 0.05$.

Solution. The hypotheses of usual interest are H_0: $\mu_A = \mu_B$ versus H_a: $\mu_A \neq \mu_B$, or if equivalently expressed as differences,

$$H_0: \mu_A - \mu_B = 0 \quad \text{versus} \quad H_a: \mu_A - \mu_B \neq 0$$

Recall that the form of the test statistic for a two-sample *t*-test is given by

$$t = \frac{(\bar{X} - \bar{Y}) - 0}{\sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

In introductory statistics, it was stated that (under certain assumptions) that t has a t-distribution with $n_1 + n_2 - 2$ degrees of freedom. It was on this t-curve that you determined the p-value for the test.

In R, it is easy to run this test using `t.test()`. See the Quick-R website if you need to refresh yourself:

```
> A <- c(156,183,120,113,138,145,142)
> B <- c(109,107,119,162,121,123,76,111,130,115)
> t.test(A, B, paired=FALSE, var.equal=TRUE)

        Two Sample t-test

data:  A and B
t = 2.302, df = 15, p-value = 0.03608
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  1.861684 48.395459
sample estimates:
mean of x mean of y
 142.4286  117.3000
```

The test statistic value is $t = 2.302$, and the p-value is 0.03608. Since this is less than 0.05, we reject H_0. There is sufficient evidence to say that there is a difference in mean growth between the two diets. ◀

Question: What assumptions are required to validate this test result?

1.
2.
3.

So, the application of the t-test comes at a price. While it is easy to produce samples from the two populations that are independent, there is no guarantee that the other two assumptions of the t-test are being met. We can check normality (using normal Q-Q plots) but this is only a cursory check to rule out evidence of non-normality. If we were to apply the t-test anyway, we run the risk of misstating the p-value of the statistical test and therefore declaring a result to be statistically significant when it is not.

11.2 Permutation tests

Permutation tests are the oldest of all nonparametric procedures, dating back to R. A. Fisher, the father of modern statistics, in the 1930s. The theory behind these tests has been developed since Fisher's inception, but it is the advent of high-speed computing that makes these methods much more easily applied today. However, we do need to cover a little bit of background so you know what is happening.

The beauty of permutation tests is that once you know what you want to test, it is simply a matter of adapting the means by which permutations are obtained. For example, the Wilcoxon signed-ranks test was a special case of a permutation test, so the subject should seem somewhat familiar.

Also, as the plural permutation tests implies, this name refers to a *class* of tests. It is not a single procedure like the sign test. Rather, it is a large class of procedures all derived from the same fundamental principle.

As with any test of statistical hypotheses, we need the following elements:

- data
- a null hypothesis
- a test statistic that is useful for assessing the null hypothesis
- the sampling distribution of the test statistic under the null hypothesis

The last item above is a complicated entity derived from the other three elements. It is a probability distribution for the *data* which in turn induces a probability distribution for the *test statistic* (because it is a function of the data). The *null hypothesis* embodies the assumptions under which this probability distribution for the data holds. (Under an alternative hypothesis, the test statistic would have a different sampling distribution.)

Permutations

For most hypothesis tests, we start with the assumptions and work forward to derive the sampling distribution of the test statistic under the null hypothesis. But for permutation tests, we will *reverse* the procedure since the sampling distribution involves the permutations which give the procedure its name and are the key theoretical issue in understanding the test.

In mathematics, a *permutation* is a reordering of the numbers 1, ..., n. For example,

1, 2, 3, 4, 5, 6, 7
1, 3, 2, 4, 5, 6, 7
4, 7, 5, 2, 6, 1, 3

are three different permutations of the numbers 1, ..., 7 (note that this includes the standard order in the first line). As one learns in basic probability, there are $n!$ permutations of n objects. In this case, $7! = 5040$, so you can see why they aren't all written out here.

In the term *permutation tests,* the notion of permutation is somewhat different. It refers to *any* of a specified class of rearrangements or modifications of the data.

The null hypothesis of the test specifies that the permutations are all equally likely. A concise way to say this is that the distribution of the data under the null hypothesis must satisfy a condition called ***exchangeability***. The sampling distribution of the test statistic under the null hypothesis is computed by forming all of the permutations, calculating the test statistic for each, and considering these values all equally likely.

► **Example: Diet comparisons.** Here are the data once again:

A	156	183	120	113	138	145	142			
B	109	107	119	162	121	123	76	111	130	115

Here is how we could build a permutation test to see if the mean growth rate differs between the two diets. The hypotheses are the same as before:

$$H_0: \mu_A - \mu_B = 0 \quad \text{versus} \quad H_a: \mu_A - \mu_B \neq 0$$

Note that diet ***A*** has 7 observations, and diet ***B*** has 10 observations. A useful test statistic for testing our null hypothesis is just the difference in the sample means, $\bar{A} - \bar{B}$. If $\bar{A} - \bar{B}$ is sufficiently large or small, then we should reject H_0 in favor of H_a.

Here is <u>a</u> procedure that will do it:

1. First, compute $\bar{A} - \bar{B}$.
2. Next, permute the 17 observations, assigning 7 of them to diet ***A*** and the remaining 10 of them to diet ***B***.
3. Compute the permuted value of $\bar{A} - \bar{B}$.
4. How many combinations exist for the 17 observations, with 7 of them assigned to diet ***A*** and 10 of them assigned to diet ***B***?
5. Compute $\bar{A} - \bar{B}$ for each possible permutation.
6. Determine the proportion of times that the permuted values of $|\bar{A} - \bar{B}|$ are at least as large as the observed value of $|\bar{A} - \bar{B}|$. <u>That</u> will be your p-value!

Question: <u>Why</u> are all the different possible permutations <u>equally likely</u> if the null hypothesis is true? *This is a vitally important question whose answer you must understand to fully use the power of permutation testing.*

11.3 A really simple example done by hand

► **Simple example.** Consider the following mutually independent observations from two different continuous populations:

A	35	25	
B	50	30	70

Test H_0: $\mu_A = \mu_B$ vs. H_a: $\mu_A < \mu_B$ at $\alpha = 0.05$, using the difference in means as the test statistic.

Solution: Let's proceed in steps:

- Compute the observed value of the test statistic $\bar{A} - \bar{B}$.

- How many combinations exist for the 5 observations, with 2 of them assigned to sample ***A*** and 3 of them assigned to sample ***B***?

- List all the possible groupings of the 5 observations, and compute the permuted value of $\bar{A} - \bar{B}$ for each (along with $\sum A_i$, just for kicks).

Permuted samples	***Population A***		***Population B***			$\bar{A} - \bar{B}$	$\sum A_i$
1	25	30	35	50	70	–24.17	55
2*	25	35	30	50	70	–20	60
3	25	50	30	35	70	–7.5	75
4	25	70	30	35	50	9.17	95
5	30	35	25	50	70	–15.83	65
6	30	50	25	35	70	–3.33	80
7	30	70	25	35	50	13.33	100
8	35	50	25	30	70	0.83	85
9	35	70	25	30	50	17.5	105
10	50	70	25	30	35	30	120

- List the permutation distribution of $\bar{A} - \bar{B}$.

$\bar{A} - \bar{B}$	-24.17	-20	-15.83	-7.5	-3.33	0.83	9.17	13.33	17.5	30
probability	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

All possible outcomes are equally likely under the null hypothesis.

- Determine the proportion of times that the permuted values of $\bar{A} - \bar{B}$ are **at least as small** as the *observed* value of $\bar{A} - \bar{B}$. This is the *p*-value for the test. Cite it and make your conclusion.

► **Simple example (revisited).** Test H_0: $\mu_A = \mu_B$ vs. H_a: $\mu_A < \mu_B$ at $\alpha = 0.05$, but this time using the sample sum from population *A* as the test statistic.

► **Simple example (revisited).** Test H_0: $\mu_A = \mu_B$ vs. H_a: $\mu_A \neq \mu_B$ at $\alpha = 0.05$, using the difference in means as the test statistic. (This is a two-tailed test: so, is the *p*-value $2 \times 0.2 = 0.4$?)

A couple of things to note:

- **The null distribution in a permutation test may not be symmetric.** If you are in a "normal distribution" or bell-curve mindset, you need to think differently now.
- **There may be more than one useful form for a test statistic** for testing a given null hypothesis. The previous example showed two versions. There may be even more!

Lecture 11 Practice Exercises

Consider the following mutually independent observations from two different continuous populations:

A	35	25	
B	50	30	70

We wish to test H_0: $\mu_A = \mu_B$ vs. H_a: $\mu_A < \mu_B$ at $\alpha = 0.05$, using the difference in means as the test statistic.

1. Compute the observed value of the test statistic $\bar{A} - \bar{B}$.

2. How many combinations exist for the 5 observations, with 2 of them assigned to sample ***A*** and 3 of them assigned to sample ***B***?

3. Below are all the possible groupings of the 5 observations, along with the permuted value of $\bar{A} - \bar{B}$ for each (ignore the last column for now):

Permuted samples	***Population A***		***Population B***			$\bar{A} - \bar{B}$	$\sum A_i$
1	25	30	35	50	70	–24.17	
2*	25	35	30	50	70	–20	
3	25	50	30	35	70	–7.5	
4	25	70	30	35	50	9.17	
5	30	35	25	50	70	–15.83	
6	30	50	25	35	70	–3.33	
7	30	70	25	35	50	13.33	
8	35	50	25	30	70	0.83	
9	35	70	25	30	50	17.5	
10	50	70	25	30	35	30	

List the permutation distribution of $\bar{A} - \bar{B}$.

$\bar{A} - \bar{B}$	
probability	

Note: this is also the **null distribution** for $\bar{A} - \bar{B}$, because probabilities were assigned to each possible value of $\bar{A} - \bar{B}$ based upon the presumption that the null hypothesis H_0 is true.

4. Why are all these different possible permutations equally likely if the null hypothesis is true? *This is a vitally important question whose answer you must understand to fully use the power of permutation testing.*

5. Determine the proportion of times that the permuted values of $\bar{A} - \bar{B}$ is **as small or smaller** (i.e. "at least as extreme") than the *observed* value of $\bar{A} - \bar{B}$. This is the *p*-value for our lower-tailed test. Cite it and make your conclusion.

Alternate form of test statistic. Re-test H_0: $\mu_A = \mu_B$ vs. H_a: $\mu_A < \mu_B$ at $\alpha = 0.05$, but this time using the sample sum from A as the test statistic. Is this statistic just as "good" to use as $\bar{A} - \bar{B}$?

A two-tailed test. Test H_0: $\mu_A = \mu_B$ vs. H_a: $\mu_A \neq \mu_B$ at $\alpha = 0.05$, using the difference in means as the test statistic. Especially important here is determining the *p*-value correctly. (The one-tailed test above had *p*-value = 0.2. So, is the *p*-value for the two-tailed test $2 \times 0.2 = 0.4$?)

STA333 Lecture 12
Introduction to permutation tests (part 2)

12.1 Randomly sampling the permutations

Prior to the computer age, the practical use of permutation tests was limited by prohibitive computations. For example, if we have two treatment groups, each with 9 observations, then the number of possible two-sample data sets that can be obtained by permuting the 18 observations (9 to a treatment) is

$$\binom{18}{9} = \frac{18!}{9!\,9!} = 48{,}620$$

This can get out of hand in a hurry. (For instance, what happens to the number of possible permutations if you were to add only two observations to each of the samples above?)

Fortunately, high-speed computing enables us to obtain an approximate *p*-value for a permutation test in such cases by using a relatively simple concept: **sampling**. *Rather than using all the possible permutations, we take a random sample from the permutations and perform the steps involved in a permutation test only on the randomly sampled permutations.*

For example, here is a 5-step procedure for obtaining an approximate *p*-value for a permutation test based on a difference of means. A similar procedure applies to permutation tests based on other test statistics.

1. First, compute the observed test statistic $\bar{X} - \bar{Y}$.
2. Suppose there are m observations from population X and n observations from population Y. Create a single vector of the $m + n$ observations.
3. Randomly "shuffle" the elements of this vector, assigning the first m to treatment 1 and the remaining n to treatment 2.
4. Compute $\bar{X} - \bar{Y}$ for the shuffled (permuted) data set.

5. Repeat steps 3 and 4 a large predetermined number of times (e.g. 1,000 or 10,000). For an upper-tailed test, the approximate *p*-value would be the proportion of "shuffled" mean differences that are greater than or equal to the observed test statistic. (Similarly, we may obtain an approximate *p*-value for a lower-tailed or two-tailed test.

Sampling the permutations is easy in R. Here is an example:

► **Example: Diet comparisons.** Here are the data once again:

A	156	183	120	113	138	145	142			
B	109	107	119	162	121	123	76	111	130	115

Run a permutation test at $\alpha = 0.05$ to see if the mean growth rate due to diet ***A*** is higher than for diet ***B*** (i.e. an upper-tailed test).

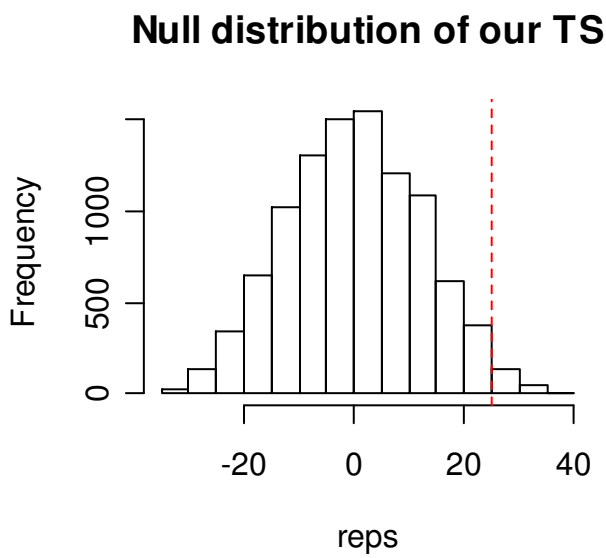
Solution. The hypotheses are H_0: $\mu_A - \mu_B = 0$ vs. H_a: $\mu_A - \mu_B > 0$. We choose to use $\bar{A} - \bar{B}$ as our test statistic. Here is an R program that runs the test:

```
A <- c(156,183,120,113,138,145,142)
B <- c(109,107,119,162,121,123,76,111,130,115)

R <- 9999                     # sets number of sampled permutations = 9999
all <- c(A,B)                 # combine both samples into one vector
k <- 1:length(all)            # create indices for the combined vector
reps <- numeric(R)            # create storage for permuted test statistics
ts <- mean(A) - mean(B)       # calculate the observed test statistic value
ts                            # print the observed test statistic

for (i in 1:R) {
  # generate m indices for the first sample
  m <- sample(k, size=length(A), replace=FALSE)
  a1 <- all[m]                    # assign permuted sample 1
  b1 <- all[-m]                   # assign the rest to sample 2
  reps[i] <- mean(a1) - mean(b1)  # calculate ith permuted test statistic
  }
pvalue <- mean(c(ts, reps) >= ts)  # calculate the approx p-value
pvalue                             # print the p-value
```

The results will be slightly different every time you run this on the same data, because every execution produces a different random sampling from the possible 19,448 permutations. In my case, my *p*-value was 0.0184, so I reject H_0. There is sufficient evidence to conclude that the mean growth rate due to diet ***A*** is higher than for diet ***B***. ◄

Below is a picture of the null distribution, in the form of a histogram of our 10,000 randomly sampled permuted values of $\bar{A} - \bar{B}$. The dashed line is the observed value of the test statistic ($\bar{A} - \bar{B} = 25.12857$). The area to the right of this on the null distribution is the *p*-value for this upper-tailed test.

```
hist(reps, main="Null distribution of our TS")
abline(v=ts, lty=2 ,color="red")
```



The beauty of this kind of approach is that:

- *You build the null distribution of the test statistic yourself from the data.* It does not depend on any underlying assumptions about the populations you originally sampled from, so it is truly a "nonparametric" procedure.

- *We can use a wide variety of different test statistics in a permutation test to test various hypotheses.* For example, we could run permutation tests using:

    - a difference of medians
    - a ratio of means
    - a ratio of variances
    - the sum of the items in one sample
    - the ranks of the values in one sample, etc.

As long as a random shuffling of the data produces outcomes that are consistent with your null hypothesis, a permutation test can be used.

Let's try a few examples.

12.2 R examples

► **Simulation example: A "permutation *t*-test".** We know how to run an independent-samples *t*-test. The purpose of this exercise is to show how well a "permutation version" of the *t*-test performs against the usual *t*-test. Here's how we will run the simulation:

1. Randomly sample $m = 25$ items from the normal distribution with $\mu_1 = 13$ and $\sigma_1 = 4$.
2. Randomly sample $n = 25$ items from the normal distribution with $\mu_2 = 15$ and $\sigma_2 = 4$.
3. It is easy to see that we are satisfying all the assumptions required by the *t*-test. **Also, it is a known fact that** H_0: $\mu_1 - \mu_2 = 0$ is false: since $\mu_1 = 13$ and $\mu_2 = 15$, we should reject our null hypothesis.
4. Run the usual *t*-test, testing H_0: $\mu_1 - \mu_2 = 0$ versus H_a: $\mu_1 - \mu_2 < 0$. Check the *p*-value.
5. Run a permutation version of the same test, using the *t*-statistic as our test statistic. Check the *p*-value and compare it to the usual *t*-test's *p*-value.

First, have R randomly generate two samples from their respective normal populations:

```
x <- rnorm(25,13,4)
y <- rnorm(25,15,4)
```

The "regular" t-test. Here is the usual parametric *t*-test as done in R:

```
t.test(x,y,var.equal=TRUE,alternative="less")

        Two Sample t-test

data:  x and y
t = -1.9249, df = 48, p-value = 0.03009
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
       -Inf -0.2583762
sample estimates:
mean of x mean of y
 13.02093  15.02878
```

Recall that the formula for the *t* test statistic is:

$$t = \frac{(\bar{X} - \bar{Y}) - 0}{\sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

My observed value of the test statistic is $t = -1.9249$ and the *p*-value is 0.03009. *In this case, the p-value is a lower-tail area under the t-distribution curve with 48 degrees of freedom.* Since the *p*-value is small, we can reject H_0. (We know this was the correct decision, since in the simulation we know that $\mu_1 = 13$ and $\mu_2 = 15$.) [Bear in mind that everyone will get a slightly different result, since everyone is generating different random samples from the two populations in the simulation.]

The "permutation" t-test. Now, let's run a permutation version of the test. This involves us randomly shuffling the samples in a manner consistent with our null hypothesis, and recomputing the same *t*-statistic for each of the random shuffles.

I choose to do 10,000 random shuffles. Here goes:

```
R <- 9999
all <- c(x,y)
k <- 1:length(all)
reps <- numeric(R)

# calculate the observed test statistic value
ts <- t.test(x,y,var.equal=TRUE)$statistic    # extracts TS value from t.test
ts                                            # print the observed TS value

for (i in 1:R) {
  # generate m indices for the first sample
  m <- sample(k, size=length(x), replace=FALSE)
  x1 <- all[m]
  y1 <- all[-m]

  # calculate ith permuted test statistic
  reps[i] <- t.test(x1,y1,var.equal=TRUE)$statistic
  }
pvalue <- mean(c(ts, reps) <= ts)             # calculate the p-value
```

After running, here's what I got:

```
> pvalue
[1] 0.0311
```

Look how close the *p*-value of the permutation version of the test comes to the usual independent-samples *t*-test *p*-value! This is to be expected, since the permutation test result is what I like to call "organic" (i.e. the data itself dictate the null distribution of the test statistic). Since all the assumptions underlying the usual *t*-test were met, it makes sense that the two approaches agree. ◄

Important question: What if the assumptions underlying the usual *t*-test were *not* met? Would you expect the two approaches to agree? If not, which approach should be more trustworthy to give you an "honest" *p*-value?

In the practice exercise that follows, you will look at a hypothesis test that could never be run using a *t*-test.

Lecture 12 Practice Exercise

A two-tailed permutation test for a mean ratio. Suppose you run an experiment where you measure the amount of a certain chemical produced in a chemical reaction. The experiment is run 10 times, where five of the runs are done using catalyst $\boldsymbol{C_1}$ and the remaining five using catalyst $\boldsymbol{C_2}$.

Here are the data:

$\boldsymbol{C_1}$	7.44	4.98	4.11	7.04	6.65
$\boldsymbol{C_2}$	4.26	4.55	5.23	5.29	5.31

The main research question involves the **mean ratio** μ_1/μ_2: the head of the research team is wondering if the mean amount of chemical produced in the reaction is the same for the two catalysts, but wants the test run by using the *ratio* as the measure of interest rather than the difference between them. Thus, if

μ_1 = true mean amount of chemical produced using catalyst $\boldsymbol{C_1}$
μ_2 = true mean amount of chemical produced using catalyst $\boldsymbol{C_2}$

then the hypotheses of interest are:

$$H_0: \frac{\mu_1}{\mu_2} = 1 \quad \text{versus} \quad H_a: \frac{\mu_1}{\mu_2} \neq 1$$

Test these hypotheses in R using a permutation test. Use 10,000 random permutations to run the test, and use $\alpha = 0.05$ to make your decision.

STA333 Lecture 13
Wilcoxon rank-sum test

13.1 Introduction

In presenting permutation tests in the last lecture, we illustrated the technique of randomly sampling the permutations in the context of an independent samples comparison of two population means μ_1 and μ_2. In that case, the hypotheses we tested were of the form

$$H_0: \mu_1 - \mu_2 = 0 \quad \text{vs.} \quad \begin{array}{l} H_a: \mu_1 - \mu_2 < 0 \\ H_a: \mu_1 - \mu_2 > 0 \\ H_a: \mu_1 - \mu_2 \neq 0 \end{array}$$

The only assumptions we really needed to test this using a permutation test was that [1] the samples were random and independently selected, and [2] that the measurement level of the measurements was on an interval scale (so that the concept of means and the distance between means is meaningful).

Another widely used nonparametric method used to compare two populations using independently collected samples attributable to Wilcoxon (of "signed-ranks test" fame) is known as the Wilcoxon rank-sum test.

Why consider another such test? It's mainly because the rank-sum test tests a different hypothesis that what you see written above. Basically, the Wilcoxon rank-sum test only uses the *ranks* of the observations as the basis of forming a test statistic. This means that only the ordinal property of the sample values comes into play in an analysis. Because of this, the Wilcoxon rank-sum test can be used on data that is ordinal by nature. (It can also be used on interval scale data, but you will lose some of the sample's information by doing so.)

13.2 Ordinal data

Data that is ordinal in nature has values that can only be ranked in terms of magnitude. It is, however, difficult or impossible to talk about how far apart the data values are. You have seen this sort of thing on surveys that employ a "Likert scale". Here is an example of such a survey question:

► **Example.** Please circle the number corresponding most closely to your opinion about the following statement:

	Strongly Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Strongly Agree
Nonparametric statistics is the coolest thing since sliced bread.	**1**	**2**	**3**	**4**	**5**

The responses on this question are ordinal. As you move from left to right on the scale, all you really know is that the amount (or "magnitude") of agreement increases. However, who can say *how much more* agreement there is between "strongly agree" and "somewhat agree"? The distance, or *interval*, between values is murky or at best very subjective.

The numbers in the scale are just there for convenience. It is important to realize that they are completely arbitrary. You could have just as easily used 2, 5, 13, 15, 29 as your scale numbers instead of 1, 2, 3, 4, 5. Because of this, the values of interval-scale based statistical measures such as $\bar{X}$ are completely arbitrary also. That doesn't stop researchers from using $\bar{X}$ in such cases anyway, but you should be aware of the shortcomings by doing so. ◄

The purpose of the Wilcoxon rank-sum test is to analyze the two independent samples problem *solely on the basis of the ordinal aspect of the data*. There are two clear cases where this might be advantageous:

1. If your data really are ordinal.
2. If your data are interval, but you have crazy outliers that could distort a means-based comparison.

The next section covers the derivation of the test, which I will follow with examples.

13.3 Derivation of the test

Assumptions. The Wilcoxon rank-sum test has the following assumptions:

1. The two samples are independent and randomly collected.
2. The measurements are at least ordinal.

Hypotheses. The hypotheses to be considered can be stated in English as follows:

H_0: *the values in populations 1 and 2 are at the same location*

vs.

H_a: *the values in population 1 locate differently than those in population 2* (two-sided test)
H_a: *the values in population 1 tend larger than those in population 2* (upper-tailed test)
H_a: *the values in population 1 tend smaller than those in population 2* (lower-tailed test)

These hypotheses are broader than what we saw with the sign test (Lecture 8) or the Wilcoxon signed-ranks test (Lecture 9). That is chiefly the result of assuming an ordinal scale for the measurements. If we apply more rigorous assumptions of having measurements on a continuous scale and distributions with the same shape, then the hypotheses may be restated as comparisons of population medians (just like in Lectures 8 and 9). But, **the main difference now is that the samples are independent**, whereas for the sign test and signed-ranks test they were paired.

Motivation for a test statistic. There are actually several perfectly valid forms for a test statistic for testing the above hypotheses, all of which are based on the ranks of the data after both sample have been combined ("joint" ranks). One popular choice for the test statistic is

W = sum of the ranks in the first sample

Here is a simple illustration of how W is calculated. Suppose you have the following two samples of values collected in a study where you are comparing two treatments:

Treatment 1	15	18	19	10	
Treatment 2	13	17	17	20	16

First, jointly rank all the observations *without regard to what sample they came from.* Use average ranks for ties. Here are the data again, with the joint ranks given in parentheses:

Treatment 1	15 *(3)*	18 *(7)*	19 *(8)*	10 *(1)*	
Treatment 2	13 *(2)*	17 *(5.5)*	17 *(5.5)*	20 *(9)*	16 *(4)*

For these data, out test statistic value is $W = 3 + 7 + 8 + 1 = 19$. (Note how we have discarded the actual data values, and retained only their relative position in the combined sample.)

How does W behave?

- **If the null hypothesis H_0 is true**, then the values in both samples should be similar, and hence we should see a mix of both large and small ranks in both samples.
- **If the null hypothesis H_0 is false**, then the values in one sample would tend to be smaller than the values in the other. So:
    - If the first treatment truly produced lower responses, then the values in sample 1 should be smaller and hence their rank sun W would be smaller.
    - If the first treatment truly produced higher responses, then the values in sample 1 should be larger and hence their rank sun W would be higher.

Null distribution and p-value. Old-fashioned statistics textbooks had endless and mind-numbing tables for this sort of thing, but we can easily estimate the null distribution and p-value using a permutation test approach. And now, the punch line for Lecture 13:

The **Wilcoxon rank-sum test** is just a permutation test using W as the test statistic.

By definition, the p-value for any hypothesis test is the probability of seeing sample evidence at least as contradictory to H_0 (and in favor of H_a) as that which you observed in the actual sample. Knowing what H_a is and what W would do if it were true is all we need to tell R to compute this for us. We then can compare this to some pre-determined significance level α.

Procedure. Here is the logic flow for running the Wilcoxon rank-sum test using a permutation test approach (by randomly sampling the permutations). Assume there are m observations in sample 1 and n observations in sample 2.

1. Combine the $m + n$ observations into one group.
2. Rank all the observations jointly from smallest to largest. Let 1 be the rank of the smallest observation, 2 the rank of the next smallest, etc. For tied observations, apply average ranks.
3. Compute the observed test statistic, W = the sum of the ranks in the first sample.
4. Randomly "shuffle" the joint ranks of the combined samples, assigning the first m of them to sample 1.
5. Compute W for the shuffled (permuted) ranks.
6. Repeat steps 4 and 5 a large predetermined number of times (e.g. 1,000 or 10,000). For an upper-tailed test, the approximate p-value would be the proportion of "shuffled" W values that are greater than or equal to the observed test statistic. (Similarly, we may obtain an approximate p-value for a lower-tailed or two-tailed test.)

Time for an example.

13.4 Wilcoxon rank-sum tests using R

We can run the test based on a modification of our generic permutation test program, or by using built-in functions provided in R. I will illustrate each of these and discuss when you can use them.

► **Example: Diet comparisons.** Here are the diet data from Lecture 11 once again:

A	156	183	120	113	138	145	142			
B	109	107	119	162	121	123	76	111	130	115

Run a Wilcoxon rank-sum test at $\alpha = 0.05$ to see if the growth rates due to diet ***A*** are higher than for diet ***B*** (i.e. an upper-tailed test).

Solution. The competing hypotheses are

H_0: the true population growth rates for diets *A* and *B* are the same
H_a: the true population growth rates for diet *A* tend to be higher than for diet *B*

Here is an R program to run the test as a permutation test. Be sure to study and understand this code:

```
A <- c(156,183,120,113,138,145,142)
B <- c(109,107,119,162,121,123,76,111,130,115)

R <- 9999                          # set number of sampled permutations
ranksall <- rank(c(A,B))           # pool both samples and jointly rank
k <- 1:length(ranksall)            # create a pooled index set
reps <- numeric(R)                 # create storage for permuted TSs
ts <- sum(ranksall[1:length(A)]) # calculate the observed TS value
ts                                 # print the observed TS value

for (i in 1:R) {
  # generate m indices for the first sample
  m <- sample(k, size=length(A), replace=FALSE)
  a1 <- ranksall[m]                        # assign ranks to permuted sample 1
  reps[i] <- sum(a1)                       # calculate ith permuted TS value
  }
pvalue <- mean(c(ts, reps) >= ts)          # calculate upper-tailed p-value
pvalue                                     # print the p-value
```

Upon running this program, I obtained an observed test statistic value of $W = 84$. The resulting *p*-value will be slightly different every time you run this on the same data, because every execution is a different random sampling from all possible permutations. My *p*-value was 0.0203, so I reject H_0. There is sufficient evidence to conclude that the true population growth rates for diet *A* are higher than for diet *B*. ◄

Note how this *p*-value (0.0203) is different than the *p*-value I got when doing the means comparison permutation test in Lecture 12 (0.0148). This difference is not because of using a

different random sample of permutations, but rather because we are using a *different test statistic* and testing a *different hypothesis*.

Using `wilcox.exact()` in R

We can perform the Wilcoxon rank-sum test using the `wilcox.exact()` function in the R add-on package `exactRankTests`. Recall that this command properly handles ties in the data.

I now "re-do" the diet data problem using `wilcox.test`. The hypotheses are the same as before. Here goes:

```
> A <- c(156,183,120,113,138,145,142)
> B <- c(109,107,119,162,121,123,76,111,130,115)
> library(exactRankTests)
> wilcox.exact(A, B, paired=FALSE, alternative="greater")

        Exact Wilcoxon rank sum test

data:  A and B
W = 56, p-value = 0.02154
alternative hypothesis: true mu is greater than 0
```

Note the use of the `paired=FALSE` option. This tells `wilcox.exact()` to run the Wilcoxon rank-sum test, rather than the Wilcoxon signed-ranks test from Lecture 9.

Note also that R computes the test statistic differently. It calculates our W as we defined it earlier, but then it subtracts $m(m + 1)/2$ from it (and then calls *that* "W"). This makes no difference to the p-value. (Confused yet?)

The test statistic value is $W = 56$, and the upper-tailed alternative's p-value is 0.02154. (Compare this to what I got using the permutation test approach). Since the p-value < 0.05, there is sufficient evidence to conclude that the true population growth rates for diet A are higher than for diet B. ◀

13.5 Some comments

Here are some observations about the Wilcoxon rank-sum test that you should be aware of:

1. *Forms for valid test statistics.* As mentioned earlier, there are actually many different forms for a valid test statistic for this test. Here are a few:

    a. W = sum of the ranks in the first sample (what we used in our permutation test).
    b. An "adjusted" sum of the ranks of the first sample. For example, R uses the test statistic $W - m(m + 1)/2$ in the function `wilcox.test()`.
    c. The sum of the ranks in the second sample.
    d. The mean of the ranks in the first sample minus the mean of the ranks in the second sample.
    e. Let X_i = any value from the first sample, and let Y_j = any value from the second sample. Look at all possible pairs of values from both samples (X_i, Y_j). Let the test statistic U = the number of (X_i, Y_j) pairs for which $X_i < Y_j$.

    These could all be used in an equivalent manner. They all deal with ranks or the ordinal properties of the data. The important thing is to know how each form of a test statistic behaves under null and alternative conditions, so that you calculate a p-value correctly.

2. *Also known as…* Two other statisticians (named Mann and Whitney) developed an essentially equivalent solution to the same testing problem as did Wilcoxon, and at almost exactly the same time! (Mann and Whitney's version of the test statistic is given above in part e.) For this reason, the procedure is sometimes also referred to as the **Mann-Whitney-Wilcoxon test**. I guess everyone wants credit.

3. *Tied scores.* Our permutation test approach is valid whether you have ties or not. If using R's built-in functions, you are always safe if you use `wilcox.exact()` from `exactRankTests` whether you have ties or not. (Recall that the built-in R function `wilcox.test()` in the base package does <u>not</u> correctly account for ties. Avoid that function in practice.)

4. *Two-tailed tests.* These are extremely easy to implement in R using the option `alternative="two.sided"` in `wilcox.exact()`. However, if you are using our version of W and a permutation test approach, figuring out how to compute the p-value correctly is a challenging problem. In this case, you might consider a different form for the test statistic that is more intuitive to work with when it comes to thinking about a two-sided hypothesis.

Lecture 13 Practice Exercise

A researcher randomly divides a class of 31 grade school students into two independent groups, and teaches one group using a traditional teaching method and the other group using a computer based tutoring tool. The primary goal is to test to see if the computer based teaching method produces higher scores in general.

Here are semester scores for the two groups:

Traditional	78	18	34	80	87	78	72	89	78	94	90	62	69	98	22	
Computer	82	89	86	77	90	95	87	84	82	80	90	86	89	85	79	91

Earlier in STA333, you ran the usual standard parametric independent samples t-test on these data to test the hypotheses H_0: $\mu_T = \mu_C$ vs. H_a: $\mu_T < \mu_C$. (The test statistic was $t = -2.3654$ with a p-value 0.01597).

Now, run a Wilcoxon rank-sum test on these same data, using two approaches:

1. Using `wilcox.exact()` in R
2. As a permutation test (use 10,000 random permutations)

Cite all elements of the test (hypotheses, test statistic, p-value, decision and conclusion in context).

STA333 Lecture 14
Wilcoxon confidence intervals for a "shift" parameter

We've talked about two different ranks-based tests to compare two populations:

1. Wilcoxon signed-ranks test for paired samples (Lecture 9)
2. Wilcoxon rank-sum test for independent samples (Lecture 13)

What this lecture covers are two corresponding confidence intervals (for each of the above) to try to estimate the *difference in location* between two populations. This could mean a difference in medians (as in the signed-ranks test) or just a general "location shift" between the populations. In either case, we will refer to the "shift" parameter using the symbol Δ.

14.1 Old stuff: CI for the difference between two population means

Let's briefly return to the parametric independent samples *t*-test for comparing two population means μ_1 and μ_2. If we were to reject the null hypothesis H_0: $\mu_1 - \mu_2 = 0$, then we conclude that one population mean is significantly different (higher or lower) than that the other. The next natural question is to then ask: *how different are they?*

We typically answer this question by finding a confidence interval for $\mu_1 - \mu_2$. A confidence interval will give us a range of values within which we can be highly confident the true difference in population means resides. (For a detailed worked out example, see section 3.8.)

We now wish to develop a similar idea, but in a **nonparametric** way. As you might expect by now, there are several ways to do this (including a more modern technique that I will cover later in the term), but for now I will introduce a method that had been traditionally used in the past.

14.2 CI for Δ when using independent samples

The following lays out the confidence interval corresponding to the Wilcoxon rank sum test for independent samples. I do this first because the rank-sum test is so fresh in your memory (Lecture 13).

Let's use (one last time, I promise) the diet data for illustration.

A	156	183	120	113	138	145	142			
B	109	107	119	162	121	123	76	111	130	115

We ran a Wilcoxon rank-sum test in the last lecture and found that there was sufficient evidence to conclude that the true population growth rates for diet *A* are higher than for diet *B*. Here's the output again to refresh you:

```
> wilcox.exact(A, B, paired=FALSE, alternative="greater")

        Exact Wilcoxon rank sum test

data:  A and B
W = 56, p-value = 0.02154
alternative hypothesis: true mu is greater than 0
```

The question now is: how much higher is the growth rate for diet *A*?

To get started, we need to think <u>not</u> specifically in terms of the difference in means (because Wilcoxon did not compare the population means), but rather a generic difference in *location*. The difference in the overall population distribution locations between diets *A* and *B* is referred to as a *shift parameter*, which we denote as Δ. Some think of Δ as being the difference between the medians of the two population distributions, but technically this is not the definition of Δ here. Our goal is to estimate Δ and find a CI for it.

Hodges-Lehmann estimate for Δ. The Hodges-Lehmann estimate is a nonparametric estimate of Δ, and is used as an alternative to the difference in the means. For independent samples, the Hodges-Lehmann estimate for Δ is defined to be the *median of all possible pairwise differences of the form* $X_i - Y_j$, where X_i is any value from the first sample, and Y_j is any value from the second sample.

Here is a simple illustration of how Δ is calculated:

► **Simple example.** Consider the following mutually independent sample observations from two different populations:

X	14	22	31	16
Y	23	19	19	

In this example, $m = 4$ and $n = 3$. To estimate Δ, we can build a 4×3 rectangular grid containing all 12 possible pairwise differences. Here it is, with the sample values in the margins:

	14	22	31	16
23	–9	–1	8	–7
19	–5	3	12	–3
19	–5	3	12	–3

Δ is estimated by the median of these 12 differences. This whole process of estimating Δ can be accomplished using the tricky R code below:

```
> x <- c(14,22,31,16)
> y <- c(23,19,19)
> diffs <- as.vector(outer(x, y, "-"))
> diffs
 [1] -9 -1  8 -7 -5  3 12 -3 -5  3 12 -3
> median(diffs)
[1] -2
```

We estimate the population shift from X to Y to be –2. ◄

***Confidence interval for* Δ.** First, let's sort the differences in the previous example:

```
> sort(diffs)
 [1] -9 -7 -5 -5 -3 -3 -1  3  3  8 12 12
```

Denote these sorted values as $D^{(1)}$, $D^{(2)}$, …, $D^{(12)}$. In a manner very similar to the confidence interval for a median as presented in Lecture 7, the confidence interval for Δ has the form

$$(D^{(k)}, D^{(mn + 1 - k)})$$

where m is the size of the first sample, n is the size of the second sample, and k denotes the k^{th} ordered slot. **What we do here (like before) is count in k from each end in the list of sorted pairwise differences to find the confidence interval.** The confidence level, however, is not determined from the binomial distribution. Without getting into the gory details of a derivation, here is code to have R determine the confidence interval for Δ:

```
conf.level <- 0.95
m <- length(x)
n <- length(y)
diffs <- sort(as.vector(outer(x, y, "-")))
mn <- length(diffs)
alpha <- 1 - conf.level
k <- qwilcox(alpha/2, m, n)
if (k == 0) k <- k+1
k
cat("Achieved confidence level:", 1 - 2*pwilcox(k-1, m, n), "\n")
c(diffs[k], diffs[mn+1-k])
```

Let's run this on the diet data:

► **Example: Diet comparisons.** (Last last time, I promise!) Let's find a 95% confidence interval for the location difference in growth between the two diets.

Solution. Here goes:

```
> A <- c(156,183,120,113,138,145,142)
> B <- c(109,107,119,162,121,123,76,111,130,115)
> conf.level <- 0.95
> m <- length(A)
> n <- length(B)
> diffs <- sort(as.vector(outer(A, B, "-")))
> mn <- length(diffs)
> HL <- median(diffs)
> HL
[1] 25
> alpha <- 1 - conf.level
> k <- qwilcox(alpha/2, m, n)
> if (k == 0) k <- k+1
> k
[1] 15
> cat("Achieved confidence level:", 1 - 2*pwilcox(k-1, m, n), "\n")
Achieved confidence level: 0.9569107
> c(diffs[k], diffs[mn+1-k])
[1]  1 47
```

The Hodges-Lehmann estimate of Δ is 25, and the 95% confidence interval for the shift between populations is (1, 47). We can be 95% confident that the location of the diet *A* population is between 1 to 47 higher than the location of the diet *B* population. (*Higher* because…?) ◄

Notes:

1. Because of the discrete nature of the determination of CI endpoints, you probably will not achieve the *exact* confidence level desired. For instance, suppose you have two samples with $m = 10$ and $n = 7$. There are 70 possible pairwise differences, so we could theoretically "march in" 35 slots from each end of the difference vector to produce a CI. You can see all the possible confidence levels associated with k values ranging from 1 to 35 by issuing these commands:

   ```
   k <- 1:35
   conf <- 1 - 2*pwilcox(k-1, m, n)
   conf[conf > 0.5]
   ```

2. There is a confidence interval option in the R function `wilcox.exact()` in the `exactRankTests` add-on package. However, even though `wilcox.exact()` runs hypothesis tests fine in the presence of ties, it doesn't handle confidence intervals correctly if you have ties (go figure!). So, avoid this function for CIs. Use the code above instead.

14.3 CI for Δ when using paired samples

The following lays out the confidence interval corresponding to the Wilcoxon signed-ranks test for paired samples (ref. Lecture 9).

Hodges-Lehmann estimate for Δ. For paired data, the Hodges-Lehmann estimate for Δ is defined differently: here, it is the *median of the "Walsh averages*," which are just the $\binom{n}{2} + n$ averages of pairs of differences of the form

$$\frac{D_i + D_j}{2} \quad \text{where } i \leq j$$

Here is a simple illustration of how Δ is calculated:

► **Simple in-class example.** Consider the following sample of three matched pairs of observations:

Subject	**1**	**2**	**3**
X	5	7	6
Y	1	4	4

Calculate the Walsh averages for these data.

***Confidence interval for* Δ.** In a manner very similar to the confidence interval for a median as presented in Lecture 7, the confidence interval for Δ here has the form

$$(W^{(k)}, W^{(nWA + 1 - k)})$$

where nWA is the number of Walsh averages, and k denotes the k^{th} ordered slot. **What we do here (like before) is count in *k* from each end in the list of sorted Walsh averages to find the confidence interval.** Here is R code to find the confidence interval for Δ for paired data:

```
conf.level <- 0.95
d <- sort(x-y)
n <- length(d)
walsh <- outer(d, d, "+")/2
walsh <- sort(walsh[lower.tri(walsh, diag = TRUE)])
nW <- length(walsh)
alpha <- 1 - conf.level
k <- qsignrank(alpha/2, n)
if (k == 0) k <- k+1
k
cat("achieved confidence level:", 1 - 2*psignrank(k-1, n), "\n")
c(walsh[k], walsh[nW+1-k])
```

Here's an example. Say hello to the monkeys again:

► **Example: Monkey stimulation.** A physiologist wants to know if monkeys prefer stimulation of brain area *A* to stimulation of brain area *B*.

In the experiment, 14 rhesus monkeys are taught to press two bars. When a light comes on, presses on Bar 1 always result in stimulation of area *A*; and presses on Bar 2 always result in stimulation of area *B*. After learning to press the bars, the monkeys are tested for 15 minutes, during which time the frequencies for the two bars are recorded. The higher the frequency, the higher the preference.

The data appear at right. This is an example of paired data, because each monkey was measured twice. The data are in the form of (*X*, *Y*) pairs.

Subject	*Bar 1 (X)*	*Bar 2 (Y)*
1	20	40
2	18	25
3	24	38
4	14	27
5	5	31
6	26	21
7	15	32
8	29	38
9	15	25
10	9	18
11	25	32
12	31	28
13	35	33
14	12	29

Find a 95% confidence interval for the true difference between number of bar presses on bars 1 and 2.

Solution. Here goes:

```
> x <- c(20,18,24,14,5,26,15,29,15,9,25,31,35,12)
> y <- c(40,25,38,27,31,21,32,38,25,18,32,28,33,29)
> conf.level <- 0.95
> d <- sort(x-y)
> n <- length(d)
> walsh <- outer(d, d, "+")/2
> walsh <- sort(walsh[lower.tri(walsh, diag = TRUE)])
> nW <- length(walsh)
> alpha <- 1 - conf.level
> k <- qsignrank(alpha/2, n)
> if (k == 0) k <- k+1
> k
[1] 22
```

```
> cat("achieved confidence level:", 1 - 2*psignrank(k-1, n), "\n")
achieved confidence level: 0.9505615

> c(walsh[k], walsh[nW+1-k])
[1] -15  -4
```

We can be 95% confident that the true difference in number of presses on bar 1 is between 4 to 15 presses *fewer* than on bar 2. ◄

Lecture 14 Practice Exercises

1. Consider the following sample of three matched pairs of observations. Calculate the Walsh averages for these data.

Subject	**1**	**2**	**3**
X	5	7	6
Y	1	4	4

2. A researcher randomly divides a class of 31 grade school students into two independent groups, and teaches one group using a traditional teaching method and the other group using a computer based tutoring tool. The primary goal is to test to see if the computer based teaching method produces higher scores in general. Find a 90% Wilcoxon-based confidence interval for the true population shift in test scores between the two methods.

   Here are semester scores for the two groups:

Traditional	78	18	34	80	87	78	72	89	78	94	90	62	69	98	22	
Computer	82	89	86	77	90	95	87	84	82	80	90	86	89	85	79	91

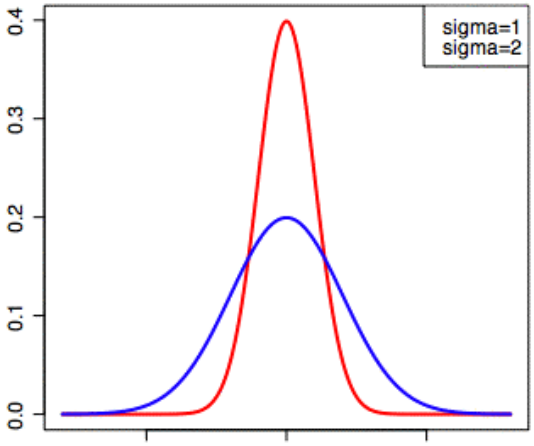
STA333 Lecture 15
Two nonparametric tests for equality of variances

15.1 Introduction to the problem

The tests we've recently covered (Lectures 11 through 14) are particularly designed to compare populations on the basis of their *location*. For example, we were interested in the difference in the effects of two treatments when the responses under one treatment tend to be higher or lower than in the other treatment.

In some situations, however, it may of interest to compare the **variability** in the responses between the two treatments. For example, suppose there are two methods for machining a certain mechanical component in a manufacturing process. There are of course target specs that should be met, but another consideration in comparing the methods might be to see which one produces the *most consistent* product, i.e. the one with the *least variability*.

To better visualize what we are talking about, look at the picture at right. These are two distributions (both normal, but that's irrelevant) with the *same* location parameter, but *difference* variances. Observations from the population with larger variance (more spread) tend to be nearer the extremes, while observations from the population with smaller variance (less spread) tend to be nearer the middle.



In this lecture, looking at nonparametric comparisons of variance (or dispersion) is the issue.

15.2 A permutation test based on deviances

Of course, we can resort to permutation testing to assess the question of variance comparison between the two populations. There are many choices for a test statistic, but in this section I introduce an efficient method for running the test using permutations.

It was stated earlier that observations from the population with larger variance tend to be nearer the extremes, while observations from the population with smaller variance tend to be nearer the middle. Our task is to build a test statistic using a scoring system that places different scores on observations based on how "far out" they are. This is crucial to developing a useful test.

Hypotheses. The parameters of interest are σ_1 and σ_2, known as the **dispersion parameters** for the two populations being compared. You may think of them as the standard deviations of the two populations, but since we will not be using the sample standard deviations when running the test, it is not technically a test that compares standard deviations. The higher σ is, the more variable the population values.

The hypotheses to be considered are

H_0: $\sigma_1 = \sigma_2$ vs. H_a: $\sigma_1 > \sigma_2$ (upper-tailed test)
H_0: $\sigma_1 = \sigma_2$ vs. H_a: $\sigma_1 < \sigma_2$ (lower-tailed test)
H_0: $\sigma_1 = \sigma_2$ vs. H_a: $\sigma_1 \neq \sigma_2$ (two-sided test)

Derivation of a test statistic. One efficient test statistic is based on **deviances**. Suppose we have two samples (as usual):

- Sample 1 consists of m observations $X_1, X_2, \ldots, X_m$
- Sample 2 consists of n observations $Y_1, Y_2, \ldots, Y_n$

We first compute the deviance for each observation (relative to its group median) as follows:

$$\text{deviance}_{Xi} = X_i - \text{median}(X) \qquad \text{deviance}_{Yi} = Y_i - \text{median}(Y)$$

where median(X) and median(Y) are the sample medians of the two groups, respectively. We then use as our test statistic the **ratio of mean deviances**:

$$RMD = \frac{\sum_{i=1}^{m} |deviance_{Xi}|/m}{\sum_{i=1}^{n} |deviance_{Yi}|/n}$$

In the permutation test, we permute the deviances of each observation and recalculate *RMD* for each permutation.

How does *RMD* behave?

- **If the null hypothesis H_0: $\sigma_1 = \sigma_2$ is true,** then the mean deviance in each sample should be about the same, and thus $RMD \approx 1$.
- **If (for instance) H_a: $\sigma_1 > \sigma_2$ is true,** then the mean deviances in the *X* sample would be larger than the mean deviance in the *Y* sample. In this case, $RMD > 1$.
- Similar arguments can be made for lower-tailed and two-tailed alternatives.

p-value. By now, it should be clear to you how to find the appropriate permutation *p*-value for *RMD*, since you know how it behaves under null or alternative conditions. We will let R compute this for us. We then can compare this to some pre-determined significance level α.

Assumptions.

1. The samples are random and independent.
2. The measurement scale for the items in the populations is interval.
3. The populations have similar shape (e.g. both symmetric bell, both skewed right, etc).

I now do an example, adapting our old permutation test program in R to the new situation.

► **Example: GRE scores.** Sixteen applications to a graduate program in psychology have been received from two universities, North Carolina and Duke. Nine came from UNC and seven from Duke. Each applicant included their score on the GRE (Graduate Records Exam), and those scores appear below:

UNC	1200	1170	1000	1010	980	1400	1430	1390	970
Duke	1220	1200	1300	1170	1080	1110	1190		

Run a test using α = 0.05 to see if there is significant evidence that GRE scores from one of these universities is more consistent that scores from the other.

Solution. The question is posed as a two tailed hypothesis, so the correct hypotheses to test here are as follows:

$$H_0: \sigma_{\text{UNC}} = \sigma_{\text{Duke}} \quad \text{vs.} \quad H_a: \sigma_{\text{UNC}} \neq \sigma_{\text{Duke}}$$

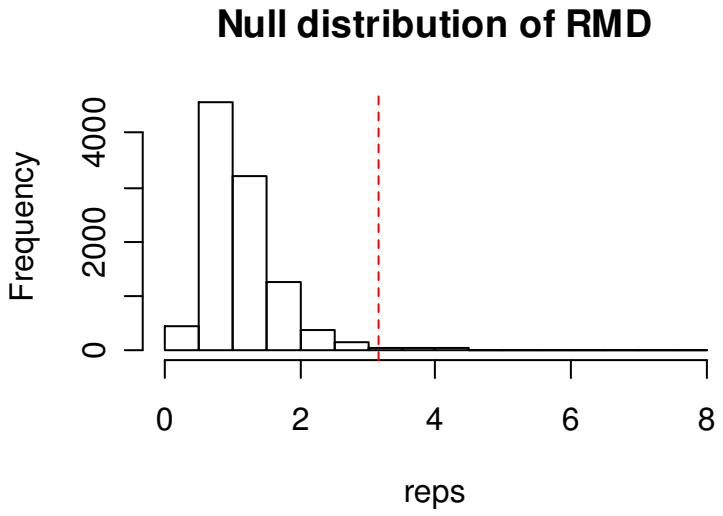
where σ_{UNC} is the true dispersion parameter for the UNC population of GRE scores, and σ_{Duke} is the true dispersion parameter for the Duke population of GRE scores.

We now adapt our R permutation program to the new test statistic and the two-sided *p*-value:

```
unc <- c(1200,1170,1000,1010,980,1400,1430,1390,970)
duke <- c(1220,1200,1300,1170,1080,1110,1190)

dev.unc <- unc - median(unc)
dev.duke <- duke - median(duke)

R <- 9999
all <- c(dev.unc,dev.duke)
k <- 1:length(all)
reps <- numeric(R)
RMD <- (sum(abs(dev.unc))/length(dev.unc))/(sum(abs(dev.duke))/length(dev.duke))
RMD

for (i in 1:R) {
  m <- sample(k, size=length(dev.unc), replace=FALSE)
  p.unc <- all[m]
  p.duke <- all[-m]
  reps[i] <- (sum(abs(p.unc))/length(p.unc))/(sum(abs(p.duke))/length(p.duke))
  }

# calculate two-tailed p-value
if (RMD > 1)    pvalue <- mean(c(RMD , reps) >= RMD) + mean(c(RMD , reps) <= 1/RMD)
if (RMD < 1)    pvalue <- mean(c(RMD , reps) <= RMD) + mean(c(RMD , reps) >= 1/RMD)
if (RMD  == 1)  pvalue <- 1
pvalue

hist(reps, main="Null distribution of RMD")
abline(v=RMD, lty=2 ,col="red")
```

Upon running the test, I obtained an observed test statistic value of $RMD = 3.154$ and a permutation p-value (based on 10,000 random permutations) of 0.0097. I can safely reject H_0. One university produces more consistent GRE scores than the other. Since the computed value of $RMD > 1$, Duke has the more consistent applicants.

Below is a picture of the null distribution of *RMD*, with our observed value indicated by a dashed line. ◀



15.3 A test based on ranks: the Ansari-Bradley test for equal variances

Occasionally we might want to resort to a ranks-based test for equality of variance between two populations. The chief reason for doing so usually comes from the situation where the data are by nature measured on an ordinal scale.

There are many ranks-based procedures for comparing variances, but they all have some drawbacks. Some require that the population medians be known (good luck with that one!), some require that they be the same (even if unknown). **A general suggestion is that you should only use a ranks-based procedure if your data are ordinal or very highly skewed.** If your data are interval scale (e.g. continuous) and somewhat well-behaved, the method in section 15.2 is superior.

The ranks-based test for comparing variances that I will consider here is referred to as the *Ansari-Bradley test*. It tests the same hypotheses as in section 15.2, and requires the following assumptions:

1. The samples are random and independent.
2. The measurement scale for the items in the populations is at least ordinal.
3. The two populations are identical in all respects (including equal medians) except for a possible difference in dispersion.

The last assumption may be viewed as a hindrance, but we can "tweak" an observed difference in medians by shifting one sample so that both have equal medians. I'll illustrate this a little later.

Derivation of a test statistic. Recall that we need to build a test statistic using a scoring system that places different scores on observations based on how "far out" they are. We can accomplish this with ranks as follows:

1. Combine the X and Y sample values into one sample, and arrange them from smallest to largest (but retain the identity of each measurement with respect to the sample of which it is a member).
2. Assign ranks as follows:
    a. Give both the smallest and largest measurements a rank of 1.
    b. Give the second smallest and second largest measurements a rank of 2.
    c. Continue in similar fashion until all values are ranked. Tied measurements should still receive average ranks.
3. The Ansari-Bradley test statistic is AB = sum of the ranks assigned to values in the X sample.

A simple example should help illustrate the process.

► **Simple example.** Consider the following mutually independent sample observations from two different populations:

X	14	22	31	16
Y	23	19	19	

Calculate the Ansari-Bradley test statistic *AB* for these data.

Solution. First, we check the medians for compatibility (see assumption 3). The median of the *X* sample is 19, and the median of the *Y* sample is 19. Since these are the same, we can proceed. We jointly sort the values, and then rank as described earlier:

Sample	***X***	***X***	***Y***	***Y***	***X***	***Y***	***X***
Sorted values	14	16	19	19	22	23	31
Rank	1	2	3.5	3.5	4	2	1

The Ansari-Bradley test statistic is the sum of the ranks associated with the *X* sample; thus, $AB = 1 + 2 + 4 = 7$. ◄

The rationale underlying the *AB* test statistic is that if the two sampled populations have equal medians, we would expect the population with the greater amount of spread to yield a sample with greater dispersion than that of the sample from the other population. *The sample with the greater amount of spread (variance) will receive the smaller ranks.*

Running the Ansari-Bradley test in R

The Ansari-Bradley test is available in R by using the `ansari.exact()` function in the R add-on package `exactRankTests`. This procedure correctly handles ties (if you've got 'em).

Here's an example:

► **Example: GRE scores.** Sixteen applications to a graduate program in psychology have been received from two universities, North Carolina and Duke. Nine came from UNC and seven from Duke. Each applicant included their score on the GRE (Graduate Records Exam), and those scores appear below:

UNC	1200	1170	1000	1010	980	1400	1430	1390	970
Duke	1220	1200	1300	1170	1080	1110	1130		

Run an Ansari-Bradley test using $\alpha = 0.05$ to see if there is significant evidence that GRE scores from one of these universities is more consistent that scores from the other.

Solution. First, check for compatibility of the medians (assumption 3):

```
> unc <- c(1200,1170,1000,1010,980,1400,1430,1390,970)
> duke <- c(1220,1200,1300,1170,1080,1110,1190)
```

```
> median(unc)
[1] 1170
> median(duke)
[1] 1190
```

We see that adding 20 to all the UNC scores will make the medians equal. We do this as follows:

```
> unc <- unc+20
```

Then run the test:

```
> library(exactRankTests)
> ansari.exact(unc, duke, alternative="two.sided")

        Ansari-Bradley test

data:  unc and duke
AB = 30, p-value = 0.02089
alternative hypothesis: true ratio of scales is not equal to 1
```

The Ansari-Bradley test statistic is $AB = 30$ and the p-value is 0.02089. We reject H_0 and conclude that one university truly produces more consistent GRE scores than the other.

How can we tell which University has more consistent scores? One quick check would be to just find some of the usual measures of variability for the two samples and "see which way the wind is blowing":

```
> sd(unc)
[1] 193.7639
> sd(duke)
[1] 72.44045
> IQR(unc)
[1] 390
> IQR(duke)
[1] 70
```

The IQR is probably better than the standard deviation to make the assessment. Either way, UNC had higher variability than Duke, and thus Duke has the more consistent GRE scores. ◄

Warning: At the time these notes were prepared, the `ansari.exact()` function in R was producing incorrect p-values for one-sided tests. In particular, it was erroneously calculating the lower-tailed p-value when an upper-tailed test was requested, and vice versa. It is hoped that this will be addressed in the near future, but for the time being, keep your eyes open if performing a one-tailed Ansari-Bradley test!

Lecture 15 Practice Exercises

Suppose you run an experiment where you measure the amount of a certain chemical produced in a chemical reaction. The experiment is run 10 times, where five of the runs are done using catalyst $\boldsymbol{C_1}$ and the remaining five using catalyst $\boldsymbol{C_2}$.

Here are the data:

$\boldsymbol{C_1}$	7.44	4.98	4.11	7.04	6.65
$\boldsymbol{C_2}$	4.26	4.55	5.23	5.29	5.31

Suppose you want to compare the variation in yields between the two catalysts.

1. Run a permutation test to compare variances between catalysts using *RMD* as your test statistic. Use 9,999 random permutations to run the test, and use $\alpha = 0.05$ to make your decision.

2. Can you think of another permutation test statistic that you could use instead of *RMD* to compare variances? Discuss.

3. Run an Ansari-Bradley test to compare variances between the two catalysts. Again, use $\alpha = 0.05$ to make your decision.

STA333 Lecture 16
An introduction to *k*-sample comparisons

16.1 Overview

We now consider the case of **comparing three or more populations** on the basis of location. In general, we will say that we are comparing k populations, so that what we develop here may be considered as the generalization of two-population comparison procedures we have already encountered. These include generalizing (to k populations) the ideas of

- Independent samples t-tests *(parametric comparison of two population means)*
- Permutation tests that compare two populations *(nonparametric resampling approaches)*
- Wilcoxon rank-sum test *(nonparametric rank-based approach)*

Later on, we'll generalize the ideas of the following two-population procedures to k populations:

- Paired samples t-tests *(parametric comparison of two means using paired samples)*
- Wilcoxon signed-ranks test *(nonparametric rank-based approach)*

First, we should review the usual parametric approach to making a k-population comparison based on means. This type of analysis is known as a **one-way analysis of variance** (or **one-way ANOVA**), and is covered in this lecture. This is a parametric analysis, but it is worth spending some time to understand it before we proceed to nonparametric approaches to the k-sample problem in Lecture 17.

16.2 One-way ANOVA (traditional parametric approach)

We observe random samples from k different populations. If this were a designed experiment, these populations may be defined on the basis of an administered treatment. For example, a certain factor may be under study by the researcher to see how it affects some measured response. The levels of the factor being manipulated by the researcher form the different **treatments**. Frequently, the data are obtained by collecting a random sample of individuals to participate in the study, and then randomly allocating a single treatment to each of the study participants. In this scenario, the individual subjects receiving treatment i may be thought of as a random sample from the specific population of all individuals who could be administered treatment i.

The one-way data structure looks like the following:

Treatment 1	**Treatment 2**	**…**	**Treatment *k***
Y_{11}	Y_{21}	…	Y_{k1}
Y_{12}	Y_{22}	…	Y_{k2}
Y_{13}	Y_{23}	…	Y_{k3}
↓	↓	↓	↓
$Y_{1,n1}$	$Y_{2,n2}$	…	$Y_{k,nk}$

There are n_1 observations sampled from treatment 1 (or population 1), n_2 from treatment 2, etc. Here is an example.

► **Example: Drug effects on alertness.** Three different medications are tested on a sample of 18 individuals to see if they impact alertness. Each subject is randomly assigned one drug, the drug is administered. After 60 minutes, a test is administered that scores the alertness level of each subject (a higher score means higher alertness).

Drug A	***Drug B***	***Drug C***
30	32	19
38	26	17
35	32	21
41	29	20
27	27	9
24	35	
	21	

The data appear in the R dataframe `onewaydrug` in our repository. The research questions are:

1. Do different medications have a different impact on alertness? *(an overall assessment)*
2. If so, which drugs differ and in what way do they differ? *(detailed pairwise assessments)*

The goal of an analysis is to answer these questions. ◄

An overall assessment: the ANOVA *F*-test

The test of interest in a one-way analysis of variance is to compare the population means. The null and alternative hypotheses are given by:

H_0: $\mu_1 = \mu_2 = \ldots = \mu_k$
H_a: at least two of the population means differ

We test these hypotheses by partitioning to total variability in the response into two components: (1) variation between treatments, and (2) variation within treatments. This is why, even though we are testing the equality of means, this method is called "analysis of variance": it's because we compare the sizes of these two variances to make the determination.

Test statistic. An *F*-statistic is used to run the test. *F*-statistics are always the ratio of two variances. The general form of an *F*-statistic is

$$F = \frac{variance\ between\ treatments}{variance\ within\ treatments}$$

Let the mean, standard deviation, and sample size for each treatment be denoted $\bar{Y}_i$, s_i, and n_i respectively. Moreover, let the overall (combined) sample size be denoted by N, and the overall (combined) sample mean be $\bar{Y}$. To construct F, we use the following measures from the sample data to construct these two variances.

Variance between treatments: $MST\ (mean\ square\ for\ treatments) = \frac{\sum_{i=1}^{k} n_i(\bar{Y}_i - \bar{Y})^2}{k-1}$

Variance within treatments: $MSE\ (mean\ square\ for\ error) = \frac{\sum_{i=1}^{k}(n_i-1)s_i^2}{N-k}$

The origin of these will be discussed in class, and even though you should be able to understand where these come from and why they work, it is not important for you to memorize these formulas. The *F*-statistic is then calculated by

$$F = \frac{MST}{MSE}$$

How does *F* behave?

- If the null hypothesis H_0 is true, then there should be very little variance between groups as compared to within groups. In this case, *F* should be small.
- If the alternative hypothesis H_a is true, then there should be a large degree of separation between groups as compared to within groups. In this case, *F* should be large.

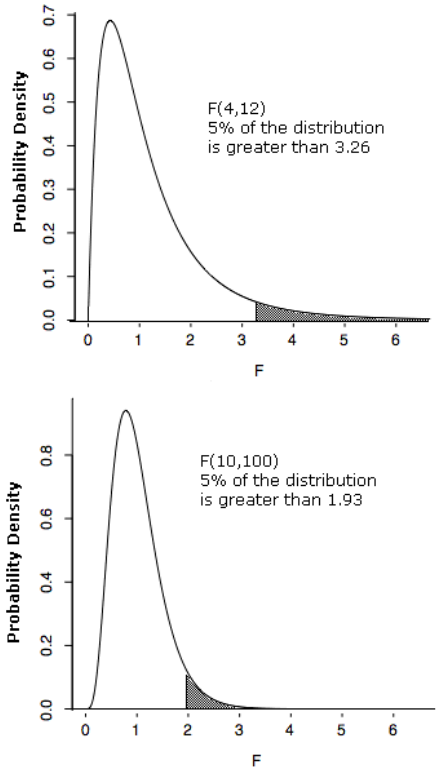
So, this is always an upper-tailed test!

Assumptions. The ANOVA *F*-test places three assumptions on the analysis:

1. The samples are random and independent
2. The populations we sample from are all normally distributed
3. All of the populations have the same variance (or standard deviation)

These assumptions should be checked to validate the test result. If the assumptions are met, then *F* follows a known parametric probability distribution known as (strangely enough) an *F*-distribution. *F* distributions are always skewed right and are determined by two parameters known as the **numerator and denominator degrees of freedom** (df_1 and df_2).

The pictures at right show two different *F*-distribution curves: the top one is for 4 numerator degrees of freedom and 12 denominator degrees of freedom. The bottom one is for 10 numerator degrees of freedom and 100 denominator degrees of freedom.



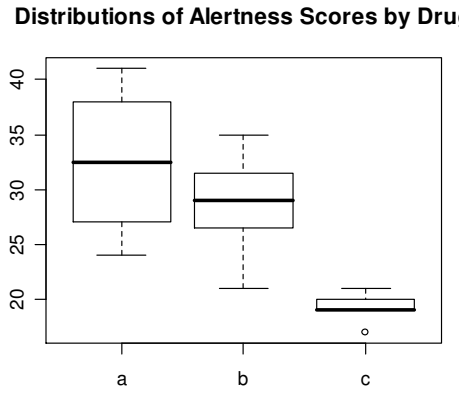
Below is the drug example in R.

► **Example: Drug effects on alertness.** Three different medications are tested on a sample of 18 individuals to see if they impact alertness. Each subject is randomly assigned one drug, the drug is administered. After 60 minutes, a test is administered that scores the alertness level of each subject (a higher score means higher alertness). The data appear in the R data frame `onewaydrug` in our repository.

Do different medications have a different impact on alertness? If so, which drugs differ and in what way do they differ?

Solution. We should first look at the data graphically. Since we are comparing multiple populations, side-by-side boxplots are a good choice:

```
> boxplot(alertness~drug, data=onewaydrug,
+          main="Distributions of Alertness Scores by Drug")
```



The scores under drug ***C*** seem to be noticeably lower on average than for the other two drugs. Also, we may suspect an equal variance problem with the data, as the distributions of scores for drugs ***A*** and ***B*** are more disperse than for drug ***C***. Here are the sample means for each drug:

```
> attach(onewaydrug)      # makes the dataframe's variable names visible to R
> tapply(alertness,drug,mean)
       a        b        c
32.50000 28.71429 19.20000
```

To run the analysis of variance, we may use the `aov()` function in R. Follow this with the `summary()` function to see the results.

```
> myFresults <- aov(alertness~drug, data=onewaydrug)
> summary(myFresults)

            Df Sum Sq Mean Sq F value   Pr(>F)
drug         2 504.27  252.14  10.753 0.001268 **
Residuals   15 351.73   23.45
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The test comparing the three drug population means is significant ($F = 10.753$, $df_1 = 2$, $df_2 = 15$, p-value = 0.0013). Thus, we could conclude that at least two of the drugs produce significantly different mean alertness responses. However, since we already suspect that assumption 3 has been violated, this result isn't necessarily trustworthy.

16.3 Multiple comparisons

When there is a significant difference detects, the ANOVA F-test does not inform you **where** the difference may lie. In such a situation, a follow-up procedure known as a *multiple comparison* is useful for seeking out pairwise mean differences where they exist.

There are numerous multiple comparison methods in existence, and they all have certain advantages and disadvantages, and some are designed for specific sorts of follow-up comparisons. One of the more widely used methods is attributable to statistician John Tukey. It

is sometimes referred to as **Tukey's Honestly Significant Difference** method (honestly!). It has the advantage of controlling the accumulation of potential false positive findings in testing when you conduct multiple tests on the same data.

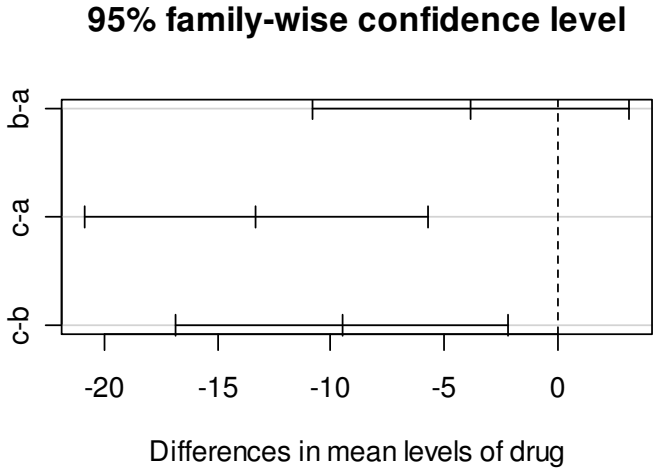
Below are Tukey multiple comparisons in R using our one-way ANOVA example, done solely for illustration:

```
> TukeyHSD(myFresults)

  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = alertness ~ drug, data = onewaydrug)

$drug
           diff       lwr       upr     p adj
b-a  -3.785714 -10.78342  3.211994 0.3632246
c-a -13.300000 -20.91631 -5.683692 0.0010797
c-b  -9.514286 -16.87916 -2.149411 0.0113636
```

The significant differences are what we expected earlier: drug ***C*** is significantly different in terms of mean alertness as compared to the other two drugs. The ***C*** versus ***A*** p-value is 0.00107, and the ***C*** versus ***B*** p-value is 0.01136. Since these differences are negative, we can confidently conclude that the true mean alertness level for ***C*** is significantly lower than for ***A*** or ***B***. ◄

We can also visualize the size of the differences via a plot:

```
> plot(TukeyHSD(myFresults))
```



Lecture 16 Practice Exercises

Suppose that the human resources department at a large company desires to know if occupational stress varies with age. They classify employees into four age classes:

A: below age 30
B: age 30-39
C: age 40-55
D over age 55

These four groups are the levels of factor **age** – there are four levels here. With this design, we shall have multiple observations in the form of scores on an Occupational Stress Assessment test from a number of employees belonging to the four levels of factor age. We are interested to know whether all the levels i.e. age groups have equal stress on the average.

The sample data appear in the R data frame `occupstress.Rdata`.

1. Build side-by-side boxplots as a visual comparison of the test scores by age classification.

2. Run a traditional one-way analysis of variance to see if there is a significant difference in mean stress level between at least two of the age classes. Show all elements of the test. If so, determine where differences lie using Tukey multiple comparisons.

STA333 Lecture 17
k-sample comparisons: nonparametric approaches

17.1 A permutation *F*-test

When we use the *F*-distribution curves to determine *p*-values (as was done in section 16.2), it is crucial that the three required assumptions be met. However, if they are not met, we can still use the computed *F*-statistic of *MST*/*MSE* as the test statistic in a permutation test.

A permutation test is perfectly valid here, since if the null hypothesis H_0: $\mu_1 = \mu_2 = \ldots = \mu_k$ is true, then random reshuffling of the samples (or equivalently, random re-allocation of subjects to different treatments) should produce outcomes consistent with H_0. This fact will be illustrated in class.

► **Example: Drug effects on alertness.** We revisit the drug comparison example from Lecture 16. Below is our permutation program adapted to run a permutation F-test for these data:

```
R <- 9999
reps <- numeric(R)

# calculate the observed F test statistic value
raw.results <- lm(alertness~drug, data=onewaydrug)
Fobs <- summary(raw.results)$f[1]
Fobs

for (i in 1:R) {
shuffle <- data.frame(drug=onewaydrug$drug,
                      alertness=sample(onewaydrug$alertness,
                      size=dim(onewaydrug)[1], replace=FALSE))
  shuffled.results <- lm(alertness~drug, data=shuffle)
  reps[i] <- summary(shuffled.results)$f[1]     # extract the ith permuted F statistic
  }
pvalue <- mean(c(Fobs, reps) >= Fobs)           # always an upper-tailed test
pvalue
```

A few notes about this permutation program:

1. An easy way to have R calculate the *F*-statistic is by using `aov()` or `lm()`. However, we need to extract the value of the *F*-statistic from permuted data sets, and this can be accomplished more easily by using the `lm()` function. Just so you know, the R commands `lm(alertness~drug)` and `aov(alertness~drug)` perform the same analysis.

2. The *F*-statistic is extracted by using the `summary()` function on the `lm()` output. `summary(raw.results)$f` extracts a vector containing the *F*-statistic, the numerator df, and the denominator df. By indexing it (`f[1]`), all we grab is the *F*-statistic.

3. Because the data are in the form of a data frame, we have to rethink how to ask R to do the permutations. In the program above, the object `shuffle` is a data frame created by keeping the `drug` values in their current places, but we randomly shuffle the values of `alertness`. Then the *F*-statistic is computed from the permuted data frame. This is repeated 9,999 times.

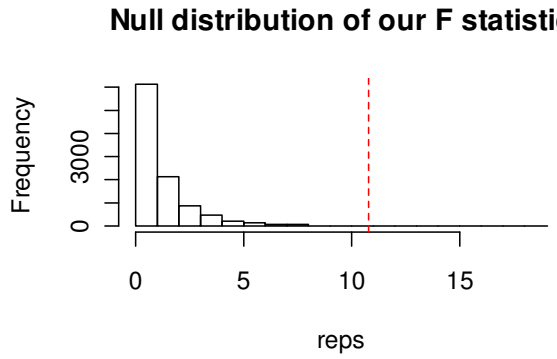
4. Since the *F*-test is always an upper tailed test, the *p*-value is **always** computed using the proportion of permuted test statistic values that are *at least as large* as the observed test statistic from the original sample, (denoted `Fobs` in the above program).

Upon running the program, I got the following results:

```
> pvalue
[1] 0.0021
```

Compare this to the parametric one-way ANOVA *p*-value obtained in the previous lecture (0.0012). The conclusion is the same, but keep in mind that **this result does not depend on the assumption of normality of the parent populations or the equal variance assumption**, since we used the observed data (not some theoretical distribution) to approximate the actual null distribution for *F*. Here is a picture of our null distribution, with the observed test statistic value (F_{obs} = 10.753) indicated. ◀

```
hist(reps, main="Null distribution of our F statistic")
abline(v=Fobs, lty=2 ,col="red")
```



17.2 A rank-based procedure for *k* samples: the Kruskal-Wallis test

The Kruskal-Wallis (KW) test is a ranks-based nonparametric alternative to a one-way ANOVA, and is a generalization of the Wilcoxon rank-sum test for two samples. It is basically performed by doing the following:

1. Jointly rank all the observations from all k samples (or treatments).
2. Perform a permutation F-test on the ranks instead of the original data values.

Simple, eh?

You may ask (yes, you may) why this is necessary if the permutation F-test is at our disposal. The answer is the same reason why we had the Wilcoxon rank-sum test at our disposal in addition to a permutation t-test: **if the data are truly ordinal by nature, then this approach may make more sense.**

Hypotheses. Bearing in mind that the KW test is a generalization of the Wilcoxon rank-sum test, the hypotheses should follow from that:

H_0: the k populations are all located in the same place (e.g. equal medians)
H_a: at least two of the populations have different locations (e.g. different medians)

The Kruskal-Wallis test statistic. Let R_{ij} denote the joint rank on observation Y_{ij}. Also assume that $N = n_1 + n_2 + \ldots + n_k$. The general layout of the ranks is as follows:

Treatments	Ranks	Sample Size	Means
1	$R_{11}, R_{12},\ldots, R_{1,n1}$	n_1	$\bar{R}_1$
2	$R_{21}, R_{22},\ldots, R_{2,n2}$	n_2	$\bar{R}_2$
↓	↓	↓	↓
k	$R_{k1}, R_{k2},\ldots, R_{k,nk}$	n_k	$\bar{R}_k$

The Kruskal-Wallis test statistic is defined as

$$KW = \sum_{i=1}^{k} n_i \left(\bar{R}_i - \frac{N+1}{2}\right)^2$$

In the absence of tied ranks, it can be shown that KW is equivalent to the F-statistic we used before (MST/MSE) when applied to the ranks rather than the original values.

Running the Kruskal-Wallis test in R

KW tests are available in R by using either of the built-in functions below:

- `kruskal.test()` function in the base R installation. Use only if no ties.
- `kruskal_test()` function in the R add-on package `coin`. Use if you have ties.

Both options use R data frames and the formula specification `y~group`. If the sample sizes are large enough, a large-sample chi-square approximation version of the test statistic is computed. (If you are wondering, there is no KW test available in the add-on package `exactRankTests`).

► **Example: Drug effects on alertness.** Run a Kruskal-Wallis test to see if a difference exists in resulting alertness due to drug. Use $\alpha = 0.05$.

Solution. We test the following:

H_0: the three drugs all produce the same general alertness
H_a: at least two of the three drugs produce different alertness levels

We can easily see in the original data that we have ties. Nevertheless, you can have R check for ties by doing a joint ranking of the `alertness` variable:

```
> rank(onewaydrug[,2])
 [1] 12.0 17.0 15.5 18.0  9.5  7.0 14.0  8.0 13.0 11.0  9.5 15.5  5.5  2.5
[15]  1.0  5.5  4.0  2.5
```

So, we use `kruskal_test()` from the `coin` package:

```
> library(coin)
> kruskal_test(alertness~drug, data=onewaydrug)

        Asymptotic Kruskal-Wallis Test

data:  alertness by drug (a, b, c)
chi-squared = 10.5612, df = 2, p-value = 0.005089
```

The large-sample KW test statistic is $\chi^2 = 10.5612$, with a p-value of 0.0051. We reject H_0. There is significant evidence to conclude that at least two of the three drugs produce different general alertness levels. ◄

Of course, we now ask: "OK, so where are the differences"? In a parametric setting, we would now perform a multiple comparison procedure (e.g. Tukey) to uncover the differences. In the next section, we see a way to do this nonparametrically.

17.3 A nonparametric multiple comparison procedure

If the null hypothesis is rejected in a Kruskal-Wallis test, we need a method to uncover where the differences are. In the usual one-way ANOVA setting, we performed a pairwise multiple comparison of **treatment means**. Now, in a nonparametric setting, this process is typically done by performing pairwise multiple comparisons of treatments on the basis of **the mean ranks** $\bar{R}_i$.

Logic. Let $\bar{R}_i$ denote the mean of the ranks in treatment *i* and let $\bar{R}_j$ denote the mean of the ranks in treatment *j*. Any nonparametric pairwise comparison of treatment *i* versus treatment *j* will be based upon investigating the quantity

$$|\bar{R}_i - \bar{R}_j|$$

which is just the **absolute distance between the mean ranks** of the corresponding samples. If $|\bar{R}_i - \bar{R}_j|$ is too large, we may confidently conclude that there is a significant difference between treatments *i* and *j*. The question then is: *how* large must $|\bar{R}_i - \bar{R}_j|$ get?

Tukey multiple comparison of all pairwise treatments. Using Tukey, we can declare treatments *i* and *j* to be different if

$$|\bar{R}_i - \bar{R}_j| \geq q_{1-\alpha,k,df(denom)} \sqrt{\frac{N(N+1)}{24}\left(\frac{1}{n_i}+\frac{1}{n_j}\right)}$$

$q_{1-\alpha,k,df(denom)}$ is the upper α area cutoff for the Studentized Range distribution used by the Tukey procedure. These comparisons can be accommodated by just using `TukeyHSD()` function on the ranks. The procedure works better if the sample sizes are large, but it is generally OK with modest sample sizes.

Here I will do the procedure for the drug alertness example:

► **Example: Drug effects on alertness.** We have already shown in the KW test that at least two of the three drugs produce different general alertness levels. Run a Tukey multiple comparison procedure on the mean ranks to determine where the differences are.

Solution. First, we must jointly rank the alertness values in the dataset (without regard to treatment group) and save the ranks into the dataset. Recall the data are in the R data frame called `onewaydrug`:

```
> onewaydrug

   drug alertness
1     a        30
2     a        38
3     a        35
4     a        41
5     a        27
```

```
6     a         24
7     b         32
8     b         26
9     b         31
10    b         29
11    b         27
12    b         35
13    b         21
14    c         19
15    c         17
16    c         21
17    c         20
18    c         19
```

We create a new variable called `ranks` in the `onewaydrug` data frame that contains the joint ranks:

```
> onewaydrug$ranks <- rank(onewaydrug$alertness)
> onewaydrug

   drug alertness ranks
1     a         30  12.0
2     a         38  17.0
3     a         35  15.5
4     a         41  18.0
5     a         27   9.5
6     a         24   7.0
7     b         32  14.0
8     b         26   8.0
9     b         31  13.0
10    b         29  11.0
11    b         27   9.5
12    b         35  15.5
13    b         21   5.5
14    c         19   2.5
15    c         17   1.0
16    c         21   5.5
17    c         20   4.0
18    c         19   2.5
```

Now, run a usual one-way analysis of variance on the ranks, and then perform a Tukey procedure on the resulting model object:

```
> results <- aov(ranks~drug, data=onewaydrug)
> TukeyHSD(results)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = ranks ~ drug, data = onewaydrug)

$drug
          diff        lwr       upr     p adj
b-a  -2.238095  -7.282127  2.805937 0.4981481
c-a -10.066667 -15.556593 -4.576740 0.0006927
c-b  -7.828571 -13.137262 -2.519881 0.0043898
```

The interpretation of where differences lie is the same as when we performed the parametric Tukey approach in Lecture 16. We can confidently conclude that the true general alertness level for ***C*** is significantly lower than for ***A*** or ***B***. The interpretation of CI endpoints is not entirely helpful since they only refer to rank positional differences in the samples. ◄

Here are both the parametric and nonparametric Tukey results side-by-side for you to compare:

Tukey multiple comparison results using sample means (parametric):

```
          diff       lwr       upr     p adj
b-a  -3.785714 -10.78342  3.211994 0.3632246
c-a -13.300000 -20.91631 -5.683692 0.0010797
c-b  -9.514286 -16.87916 -2.149411 0.0113636
```

Tukey multiple comparison results using mean ranks (nonparametric):

```
          diff        lwr       upr     p adj
b-a  -2.238095  -7.282127  2.805937 0.4981481
c-a -10.066667 -15.556593 -4.576740 0.0006927
c-b  -7.828571 -13.137262 -2.519881 0.0043898
```

In this example the same findings were arrived at in both cases, but you can see that the adjusted *p*-values are different, so there is the potential for the rank-based approach to arrive at different findings. You should expect this if the usual ANOVA assumptions are strongly violated.

Lecture 17 Practice Exercises

Suppose that the human resources department at a large company desires to know if occupational stress varies with age. They classify employees into four age classes:

A: below age 30
B: age 30-39
C: age 40-55
D over age 55

These four groups are the levels of factor **age** – there are four levels here. With this design, we shall have multiple observations in the form of scores on an Occupational Stress Assessment test from a number of employees belonging to the four levels of factor age. We are interested to know whether all the levels i.e. age groups have equal stress on the average.

The sample data appear in the R data frame `occupstress.Rdata`.

1. Run a one-way analysis of variance as a permutation *F*-test to see if there is a significant difference in mean stress level between at least two of the age classes. Show all elements of the test.

2. Run a Kruskal-Wallis one-way nonparametric analysis of variance to see if there is a significant difference in median stress level between at least two of the age classes. Show all elements of the test.

STA333 Lecture 18
Spearman rank correlation

18.1 Introduction

In this lecture, we will discuss ways to assess the association between two quantitative variables (which I will generically refer to as X and Y). By "assessing association", I mean **investigate any trends in the measurements between X and Y**.

For example, is amount of study related to your GPA? You might expect that more study may result in a higher GPA. If this is true, then in a sample of students you would expect to see students who study less having lower GPAs, and students who study more having higher GPAs. If this were the case, we would say that amount of study and GPA are *positively associated*.

Statisticians measure the degree of association between quantitative variables using statistical measures known as **correlation coefficients**. There are many different ways in which we could construct a correlation coefficient. As you might imagine, there are *parametric* correlations and *nonparametric* correlations.

All correlation measures have the following properties:

1. They always range between –1.0 and +1.0.
2. A correlation coefficient of –1.0 indicates that the two variables are *perfectly negatively associated*. (Sometimes people call this "inversely related".)
3. A correlation coefficient of +1.0 indicates that the two variables are *perfectly positively associated*.
4. A correlation coefficient of 0 indicates that the two variables are *not associated*. (Sometimes people say that the two variables are independent.)
5. **Correlation does not imply causation**!

18.2 Old stuff: Pearson correlation and testing for linear association

The **Pearson correlation coefficient** (usually denoted r) is the most common parametric method of measuring the degree of association between two quantitative variables. However, it has a restriction: the type of association it investigates is linear association between X and Y: that is, the type of trend it looks for a straight-line relationship.

Before I present an example, I'll first say that for any correlation, the data you collect must be in the form of (X_i, Y_i) pairs, where X_i is the value of X for the i^{th} sample member, and Y_i is the value of Y for the i^{th} sample member. In other words, you must have **paired data**.

► **Example: TV ad yields.** These data appeared in the Wall Street Journal in 1984. TV ads were selected by an annual survey conducted by Video Board Tests, Inc., a New York ad-testing company, based on interviews with 20,000 adults who were asked to name the most outstanding TV commercial they had seen, noticed, and liked. The retained impressions were based on a survey of 4,000 adults, in which regular product users were asked to cite a commercial they had seen for that product category in the past week. The variable `spend` is the 1983 TV advertising budget (in $ millions), and `milimp` is the number of retained impressions per week (in millions). The data appear in the R data frame `tvadyields`, and are given below:

firm	spend	milimp
MILLER LITE	50.1	32.1
PEPSI	74.1	99.6
STROHS	19.3	11.7
FEDEX	22.9	21.9
BURGER KING	82.6	60.8
COCA COLA	40.1	78.6
MCDONALDS	185.9	92.4
MCI	26.9	50.7
DIET COLA	20.4	21.4
FORD	166.2	40.1
LEVI STRAUSS	27.0	40.8
BUD LITE	45.6	10.4
ATT	154.9	88.9
CALVIN KLEIN	5.0	12.0
WENDYS	49.7	29.2
POLAROID	26.9	38.0
SHASTA	5.7	10.0
MEOW MIX	7.6	12.3
OSCAR MEYER	9.2	23.4
CREST	32.4	71.1
KIBBLES N BITS	6.1	4.4

The goal is to assess the strength of the relationship between spending on advertising and effectiveness (measured by retained impressions).

The Pearson correlation is given by the formula

$$r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{X_i - \bar{X}}{s_X}\right)\left(\frac{Y_i - \bar{Y}}{s_Y}\right)$$

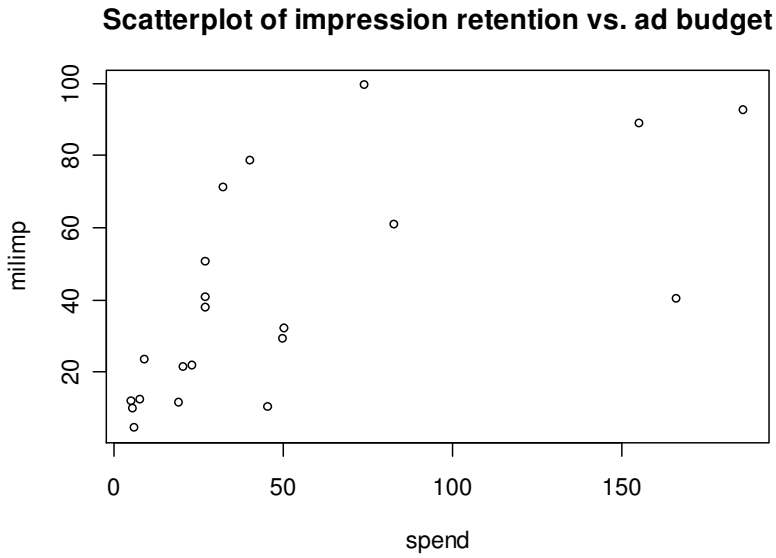
.

We want to do the following:

- Plot the data to visualize the nature of the relationship (e.g. is it linear?).
- Calculate *r* (using software).
- Conduct a hypothesis test to see if the population correlation is significant.

Solution. In R, these steps are easy. First, a plot:

```
> plot(milimp~spend, data=tvadyields,
+      main="Scatterplot of impression retention vs. ad budget")
```



We can see what appears to be a weak positive trend (i.e., more ad money seems associated with higher retained impressions), but the association does not seem particularly linear. Even so, we calculate *r* as follows:

```
> attach(tvadyields)        # makes variables in the dataframe visible to R
> cor(milimp, spend, method="pearson")
[1] 0.6511683
```

We get $r = 0.651$. This is a moderately strong positive linear association. (A frequently used rule of thumb is that $|r| > 0.80$ or so qualifies as a strong linear relationship.)

We may also run a hypothesis test for the Pearson correlation. To do so, we realize that we observed a random sample of $n = 21$ ads. We may use these as a random representation of the population of all ads, and hence r is a sample representation of the true linear association between advertising expenditure and ad impression retention.

Let ρ ("rho") represent the true unknown Pearson population correlation. The null hypotheses we want to test is

$$H_0: \rho = 0 \quad \text{(i.e. the two variables are not linearly related)}$$

versus one of the three following possible alternatives:

$H_a: \rho \neq 0$ (two-sided test, stating that the two variables are linearly related)
$H_a: \rho > 0$ (upper-tail test, stating that the two variables are positively linearly related)
$H_a: \rho < 0$ (lower-tail test, stating that the two variables are negatively linearly related)

Under the assumption of normal populations for both X and Y, this can be run using a t-test. In R, the test can be done as follows:

```
> cor.test(milimp, spend, alternative="greater", method="pearson")

        Pearson's product-moment correlation

data:  milimp and spend
t = 3.74, df = 19, p-value = 0.0006937
alternative hypothesis: true correlation is greater than 0
```

The test statistic is $t = 3.74$ on 19 degrees of freedom, and the p-value is 0.0007, so we can reject H_0 and confidently state that there is a significant linear association between ad expenditure and impression retention. ◄

Of course, this method is fine *unless* you encounter at least one of the following issues:

1. What if X and/or Y are not normally distributed?
2. What if your data are ordinal?
3. What if X and Y are related, but not necessarily in a linear way?

Nonparametrics to the rescue!

18.3 Spearman's rank correlation

► **In-class example.** Consider the following data:

X	0	2	3	4	6
Y	0	16	81	256	1296

Let's use R to calculate the Pearson correlation:

```
> x <- c(0,2,3,4,6)
> y <- c(0,16,81,256,1296)
> cor(x,y,method="pearson")
[1] 0.839785
```

1. How would you interpret this result?
2. Now, plot the data. What do you see?

So what is the moral of the story? It is that the usual Pearson correlation doesn't necessarily tell the whole story. The association between X and Y in the above example is in some sense perfect … it's just not linear. ◄

Spearman's rank correlation is a nonparametric correlation in which only the ordinal aspect of the relationship between X and Y is investigated. This actually has an advantage: since we are "discarding" the notion of distance between values, we are not locked into investigating a specific parametric relationship (e.g. a linear one). Thus, Spearman can investigate the strength of other types of associations. By using only the ranks, all we are investigating is the monotonicity of the relationship between X and Y.

To calculate Spearman's rank correlation, this is all we do:

1. Rank the X values.
2. Rank the Y values.
3. Find the Pearson correlation using the ranks instead of the original data values. The resulting correlation coefficient is the Spearman rank correlation.

► **In-class example.** As an exercise, do this using the above data. What do you get for the Spearman correlation? Does your answer make sense?

► **Example: TV ad yields.** Recall from the scatterplot that the relationship between spending on advertising and ad effectiveness did not look particularly linear. So, let's calculate Spearman's rank correlation for these data instead of the Pearson correlation. In R, this is accommodated easily using the `cor()` function, but changing to the option `method="spearman"`.

Just to be more explicit, here is what you get when ranking the data:

firm	spend	*Rank*(spend)	milimp	*Rank*(milimp)
MILLER LITE	50.1	**16**	32.1	**11**
PEPSI	74.1	**17**	99.6	**21**
STROHS	19.3	**6**	11.7	**4**
FEDEX	22.9	**8**	21.9	**8**
BURGER KING	82.6	**18**	60.8	**16**
COCA COLA	40.1	**13**	78.6	**18**
MCDONALDS	185.9	**21**	92.4	**20**
MCI	26.9	**9**	50.7	**15**
DIET COLA	20.4	**7**	21.4	**7**
FORD	166.2	**20**	40.1	**13**
LEVI STRAUSS	27.0	**11**	40.8	**14**
BUD LITE	45.6	**14**	10.4	**3**
ATT	154.9	**19**	88.9	**19**
CALVIN KLEIN	5.0	**1**	12.0	**5**
WENDYS	49.7	**15**	29.2	**10**
POLAROID	26.9	**9**	38.0	**12**
SHASTA	5.7	**2**	10.0	**2**
MEOW MIX	7.6	**4**	12.3	**6**
OSCAR MEYER	9.2	**5**	23.4	**9**
CREST	32.4	**12**	71.1	**17**
KIBBLES N BITS	6.1	**3**	4.4	**1**

Here's the result:

```
> cor(milimp, spend, method="spearman")
[1] 0.7528419
```

Note that the rank correlation is higher than the linear correlation in this case ($r = 0.651$).

Can we run a hypothesis test for the Spearman correlation? Sure. One quick way is to use the `cor.test()` function in R with the Spearman option:

```
> cor.test(milimp, spend, alternative="greater", method="spearman")

        Spearman's rank correlation rho

data:  milimp and spend
S = 380.6235, p-value = 4.104e-05
alternative hypothesis: true rho is greater than 0
sample estimates:
      rho
0.7528419
```

```
Warning message:
In cor.test.default(milimp, spend, alternative = "greater", method =
"spearman") :
  Cannot compute exact p-values with ties
```

The *p*-value is 0.000041, so there is significant evidence of an association between ad expenditure and ad effectiveness. (Notice that I *didn't* say there was significant evidence of a *linear* association between ad expenditure and ad effectiveness.)

However, we got the dreaded message that there were ties, so the *p*-value is approximate. If there are a lot of ties, this can be a real problem. We can get around this by instead running the test using a permutation test approach.

A permutation test for the Spearman correlation

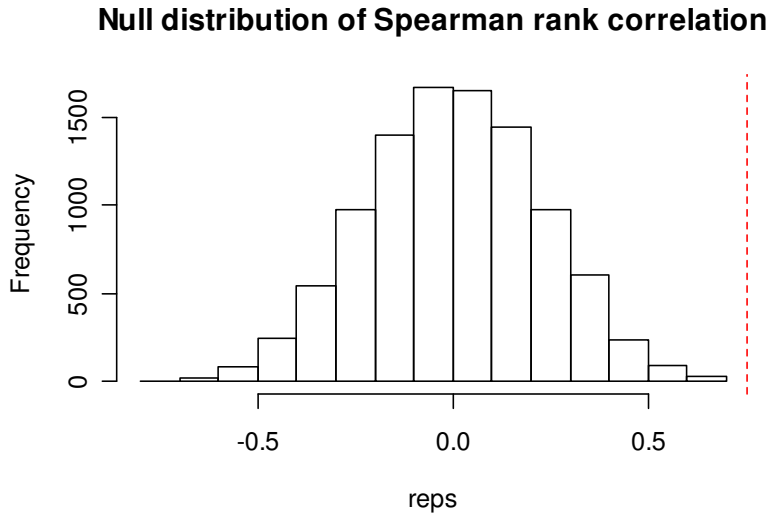
Let ρ_S represent the true unknown Spearman population correlation. Our null hypothesis states that H_0: $\rho_S = 0$ (i.e., X and Y are unrelated). Recall that a permutation test is possible as long as we can reshuffle ("permute") our data in a manner that is consistent with our null hypothesis.

In this case, here's how this works: if X and Y are truly unrelated, then X does not affect the values of Y. Thus, any observed Y_i value is just as likely to appear with any X_i value. So, we can create a permuted version of our sample that is consistent with H_0 simply by leaving the X's alone and reshuffling the Y's. Here's a program in R to do it for the TV ad example:

```
attach(tvadyields)

R <- 9999
reps <- numeric(R)

# calculate the observed Spearman rank correlation
spearmancorr <- cor(milimp, spend, method="spearman")
spearmancorr

for (i in 1:R) {
  x <- tvadyields$spend
  y <- sample(tvadyields$milimp, size=dim(tvadyields)[1], replace=FALSE)
  reps[i] <- cor(y,x, method="spearman")
  }

pvalue <- mean(c(spearmancorr, reps) >= spearmancorr)
pvalue


hist(reps, main="Null distribution of Spearman rank correlation")
abline(v=spearmancorr, lty=2 ,col="red")
```

The *p*-value above is set up for an upper-tailed test. Upon running this, we get the following:

```
> pvalue
[1] 1e-04
```



Our permutation p-value is 0.0001. The conclusion is the same as before. ◄

Two notes:

1. We can just as easily run permutation tests for a Pearson correlation, or even a regression slope. While these add the assumption of a linear relationship into the mix, we can get around the usual normality assumption that Pearson correlation and regression tests require by using permutation tests to calculate p-values. This will be illustrated in class.

2. **Caution.** One of the implicit assumptions about Pearson and Spearman correlations is that *the observations are collected independently*, that is, the information in one observation *cannot* be used to predict what another observation will be. The most common situation where this assumption is violated is when you are dealing with time-dependent data. For instance, suppose you are looking at long-term trends in sales over time. You might be interested in investigating the relationship between time and amount of sales, but short-term seasonal effects might mean that what happens next quarter could be predicted from what happened last quarter. This sort of interdependency among observations can invalidate the results of a test for a correlation (causing too many false positive findings, or increased Type I error rates). So, keep your eyes open for this sort of situation!

Lecture 18 Practice Exercises

1. Consider the following data:

X	0	2	3	4	6
Y	0	16	81	256	1296

   a. Use R to calculate the Pearson correlation. How would you interpret this result?

   b. Now, plot the data. What do you see?

   c. Find the Spearman rank correlation by ranking the *X* and *Y* values and calculating the Pearson correlation from the ranks. What do you get for the Spearman correlation? Does your answer make sense?

2. **Respiratory rates for children**. A high respiratory rate is a potential diagnostic indicator of respiratory infection in children. To judge whether a respiratory rate is truly "high," however, a physician must have a clear picture of the distribution of normal respiratory rates. To this end, Italian researchers measured the respiratory rates of 618 children between the ages of 15 days and 3 years. The data appear in the R workspace `respiratory`.

   a. Plot the data and describe the nature of the relationship between respiratory rate and age (e.g. strength, direction, pattern, etc).

   b. Which correlation measure do you feel is most appropriate to use here? Why?

   c. Test for a significant correlation (positive or negative) between age and respiratory rate for normal children in this age range. Use a permutation test, and report all aspects of the test and the findings.

STA333 Lecture 19
Fisher's Exact Test for a 2 × 2 contingency table

19.1 Introduction

In the last lecture, we discussed ways to assess the association between two quantitative variables. This was done by computing and testing a correlation coefficient. In this section, the goal is the same: assess and test to see if two variables are associated, but the difference now is that both variables are qualitative and dichotomous.

► **Example: Opinion on same-sex marriage.** There is diversity of opinion on whether marriage is more of a social institution or a religious institution. To that end, we will conduct a small in-class survey whereby everyone will be asked to answer the following questions:

Question 1: *Do you consider yourself a religious person, or person of faith?*
Question 2: *Do you believe that same-sex marriage should be legalized?*

We will fill in the table below, and analyze later:

		Legalize same-sex marriage?		
		Yes	*No*	**Total**
Are you a person of faith?	*Yes*			
	No			
	Total			

◄

Fisher's exact test is attributable to R.A. Fisher, one of the founders of experimental statistics, and is another example of a permutation test, here applied to a 2×2 table under the presumption of some null hypothesis. So, at this point it would be helpful to know what the hypotheses are!

Hypotheses. The null hypotheses we want to test is

H_0: the two variables are not associated

versus one of the three following possible alternatives:

H_a: the two variables are positively associated (upper-tailed test)
H_a: the two variables are negatively associated (lower-tailed test)
H_a: the two variables are associated (two-tailed test)

The way we assess the degree of association between two qualitative variables in a table is to **determine whether the proportions of those falling into each category differ by group.** For example, consider this example:

► **Example: Drug treatment of pulmonary embolism.** A surgeon is interested in determining whether the administration of a certain drug can reduce the incidence of pulmonary embolism (PE) in patients undergoing high-risk surgery. Nineteen patients were selected for the study, with 11 getting the drug and 8 receiving the standard treatment. Here are the data:

		Experienced pulmonary embolism?		
		Yes	*No*	**Total**
Treatment	*Drug*	3	8	**11**
	No drug	4	4	**8**
	Total	**7**	**12**	**19**

Let p_{drug} = true probability of experiencing PE when taking the drug, and $p_{standard}$ = true probability of experiencing PE without taking the drug. Our point estimates of these parameters are:

$$\hat{p}_{drug} = \frac{3}{11} = 0.272 \qquad \hat{p}_{standard} = \frac{4}{8} = 0.5$$

It appears that the drug has lowered the incidence of PE as compared to the standard treatment, but remember that this is just sample evidence. The real question is if this is a statistically significant drop. To address this exactly, we can resort to the notion of permutations: that is, rearranging the data and seeing how likely it would be to see results at least as extreme as these if in fact the null hypothesis was true.

By the way, this is an example of a lower-tailed test. The hypotheses here are

H_0: the treatments do not produce any change in the likelihood of PE
H_a: the drug treatment lowers the likelihood of PE

19.2 *p*-values in Fisher's exact test: an illustration

To illustrate, let's consider a generic table of counts:

	Column 1	Column 2	**Total**
Row 1	*a*	*b*	***a* + *b***
Row 2	*c*	*d*	***c* + *d***
Total	***a* + *c***	***b* + *d***	***n* = *a* + *b* + *c* + *d***

The key notion in calculating the *p*-value here is to permute the cell counts *a, b, c, d* **within the restriction of fixing the row and column marginal totals and sample size *n***. Without getting into a theoretical derivation (in other words, believe me when I tell you), it can be shown that the probability of this particular table arrangement is given by

$$probability = \frac{\binom{a+c}{a}\binom{b+d}{b}}{\binom{n}{a+b}} = \frac{(a+b)!\,(b+c)!\,(a+c)!\,(b+d)!}{a!\,b!\,c!\,d!\,n!}$$

The *p*-value for Fisher's exact test is the probability, given the observed marginal counts, of obtaining *exactly the counts observed and any configuration more extreme*. By "more extreme," I mean any configuration (given the observed marginal totals) with a smaller probability of occurrence in the same direction (one-tailed) or in both directions (two-tailed).

For example, if your 2 × 2 table is

2	3
6	4

then all the possible permutations of the cell counts that retain the same fixed marginal totals are:

0	5
8	2

1	4
7	3

2	3
6	4

3	2
5	5

4	1
4	6

5	0
3	7

with corresponding probabilities **0.007**, **0.093**, **0.326**, **0.392**, **0.163**, and **0.019** (check these).

The two shaded tables on the left constitute the permutations that are *more extreme* than the observed configuration in the *same* direction. The two shaded tables on the right are the permutations more extreme in the *opposite* direction. (Extremity is defined in terms of probability, so the probability of any configuration to the right of the table of observed counts with probability less than or equal to that of the observed configuration are added to the total probability of more extreme configurations.)

Thus, the one-tailed *p*-value for this table would be **0.007 + 0.093 + 0.326 = 0.426.**
The two-tailed *p*-value for this table would be **0.007 + 0.093 + 0.326 + 0.163 + 0.019 = 0.608.**

19.3 Fisher's exact test in R

Fisher's exact test is very easy to implement in R, but you must be careful about correctly specifying the alternative hypothesis. Also, you may enter your table counts directly into a data matrix. This is illustrated with the following example:

► **Example.** Let's enter the data and run the one-tailed test for the simple illustration in section 19.2. The following code creates the table as an R matrix:

```
> mytable <- matrix(c(2,6,3,4),
+                   nrow = 2,
+                   dimnames = list(rows = c("row1", "row2"),
+                                   columns = c("col1", "col2")))
```

I print the matrix to confirm that it is what I want it to be:

```
> mytable
      columns
rows   col1 col2
  row1    2    3
  row2    6    4
```

Here's the analysis. The built-in R function is `fisher.test()`:

```
> fisher.test(mytable, alternative = "less")

        Fisher's Exact Test for Count Data

data:  mytable
p-value = 0.4266
alternative hypothesis: true odds ratio is less than 1
95 percent confidence interval:
 0.000000 4.446144
sample estimates:
odds ratio
 0.4698172
```

The lower-tailed p-value is 0.426 as determined earlier. We fail to reject H_0. ◄

► **Example: Drug treatment of pulmonary embolism.** Run Fisher's exact test to determine whether the administration of the drug significantly reduces the incidence of pulmonary embolism (PE) in patients undergoing high-risk surgery. Use $\alpha = 0.05$. Again, here are the data:

		Experienced pulmonary embolism?		
		Yes	*No*	**Total**
Treatment	*Drug*	3	8	**11**
	No drug	4	4	**8**
	Total	**7**	**12**	**19**

This is another example of a lower-tailed test: the important cell count to look at is $a = 3$ (which indicates that 3 people who took the drug experienced a pulmonary embolism). Since the hypotheses are

H_0: the treatments do not produce any change in the likelihood of PE
H_a: the drug treatment lowers the likelihood of PE,

we would expect this count to *lower* if the drug were effective. **This kind of assessment is essential for properly specifying the correct test in R.**

```
> PE <- matrix(c(3,4,8,4),
+              nrow = 2,
+              dimnames = list(drug = c("yes", "no"),
+                              exp.pe = c("yes", "no")))
> PE
     exp.pe
drug  yes no
  yes   3  8
   no   4  4
```

Now, run the test:

```
> fisher.test(PE, alternative = "less")

        Fisher's Exact Test for Count Data

data:  PE
p-value = 0.2966
alternative hypothesis: true odds ratio is less than 1
95 percent confidence interval:
 0.000000 2.736281
sample estimates:
odds ratio
 0.3959561
```

The p-value for the test is 0.2966. Thus, there is not sufficient evidence in this study to conclude that the drug is effective in reducing the incidence of pulmonary embolism. ◄

Notes:

1. There is a version of Fisher's exact test for tables with more than two rows and/or columns ($r \times c$ in general), but the p-value calculations can be very complicated.

2. For larger sample sizes, Fisher's exact test can be approximated by using a **chi-square test of independence**, which is covered in virtually every introductory statistics book ever published. However, Fisher's always provides an exact p-value.

3. *Question:* How does Fisher's exact test differ from McNemar's Test? (See Lecture 10).

Lecture 19 Practice Exercises

Opinion on same-sex marriage. There is diversity of opinion on whether marriage is more of a social institution or a religious institution. To that end, we will conduct a small in-class survey whereby everyone will be asked to answer the following questions:

Question 1: *Do you consider yourself a religious person, or person of faith?*
Question 2: *Do you believe that same-sex marriage should be legalized?*

We will fill in the table below, and analyze later:

		Legalize same-sex marriage?		
		Yes	*No*	**Total**
Are you a person of faith?	*Yes*			
	No			
	Total			

1. After taking the class survey, fill in the observed counts into the contingency table.

2. Create a set of table counts that exhibits even stronger evidence of an association between faith status and opinion on legalizing same-sex marriage than the observed table of counts that resulted from the class survey. What is the probability of this particular table arrangement?

3. Run Fisher's exact test to determine if there is significant evidence to conclude that, in general, there is a relationship between faith status and opinion on legalizing same-sex marriage.

STA333 Lecture 20
Introduction to bootstrapping

20.1 Introduction

Statistics has changed drastically in the past decade or so as a result of modern computing and software that enable us to look at data graphically and numerically in ways that were previously inconceivable. These advances have also opened up doors to methods of data analysis that would have been completely impractical (if not impossible) before. Permutation tests are an example of this: the theory behind them was developed long ago and is so straightforward and elegant, but they were impossible to perform in a practical sense with real world problems before resampling software came on the scene.

The "**bootstrap**" is another method of resampling your data, but it has a different goal in mind as opposed to resampling as we did in permutation tests. But it is a nonparametric resampling method, so it has a lot of the same benefits that permutation tests had:

1. **Fewer assumptions.** Resampling methods such as permutation tests or bootstrapping do not require normally distributed populations or large samples to work.

2. **Better accuracy.** Resampling methods can be more accurate that traditional methods.

3. **Generalizability.** Permutation tests and bootstrap methods are remarkably similar for a wide array of different statistics, so they do not require new formulas or "starting from scratch" for every new statistic you encounter. If you understand the general process taking place, it is easy to adapt the process to new problems.

So now, on to this bootstrap thing. What is it, anyway?

20.2 Bootstrapping 101

Bootstrapping is a method of resampling your observed data so as to estimate the standard error (SE) and sampling distribution of some statistic, like the sample mean.

To introduce the notion of bootstrapping, let's see how it works with a particular example. I'll start by showing you how to bootstrap, and then relate the results to ideas of standard errors and sampling distributions.

► **"Baby steps" example.** Suppose you had the following data sampled from some population:

23.3 26.1 19.0 28.8 29.0

We want to compute a 95% confidence interval for the mean of the population using these $n = 5$ measurements. Tradition would be to just apply the usual *t*-based confidence interval formula:

$$\bar{x} \pm t_{0.025} \frac{s}{\sqrt{n}}$$

The quantity $s/\sqrt{n}$ is called the **standard error** of the sample mean, and may be written as $SE_{\bar{x}}$. However, *what if your population were not normal*? Then this CI formula isn't valid. Alright, then check your sample for normality using something like a normal quantile-quantile plot. That's difficult, though, because the plot won't show much if *n* is small. So what can we do?

Statistical inference is based on the sampling distribution of sample statistics. **The bootstrap, first and foremost, is a way of finding the sampling distribution (at least approximately) from just one sample.** Here is how you do it:

1. **Resample.** Create hundreds, if not thousands, of new samples called **bootstrap samples**, by sampling *with replacement* from the original sample. Each bootstrap sample is the same size as the original sample.
2. **Calculate the bootstrap distribution.** Calculate the statistic of interest from each resampling. The distribution of these resample statistics is called the *bootstrap distribution*.
3. **Use the bootstrap distribution.** The bootstrap distribution gives us information about the shape, center and spread of the sampling distribution of the statistic.

For the example above, here is an illustration of the process using R to do the resampling. The code below creates three different bootstrap samples:

```
> x <- c(23.3,26.1,19.0,28.8,29.0)
> x
[1] 23.3 26.1 19.0 28.8 29.0

> bootsample1 <- sample(x, size=length(x), replace=TRUE)
> bootsample1
[1] 26.1 23.3 23.3 19.0 19.0
```

```
> bootsample2 <- sample(x, size=length(x), replace=TRUE)
> bootsample2
[1] 26.1 26.1 19.0 19.0 29.0

> bootsample3 <- sample(x, size=length(x), replace=TRUE)
> bootsample3
[1] 29.0 26.1 26.1 29.0 26.1
```

Since bootstrapping resamples the original data with replacement, any value may be drawn once, more than once, or not at all.

This example illustrates the notion on a very small scale. **In practice, we would draw literally hundreds or thousands of bootstrap samples, not just three!** To facilitate this in software, using loops much like we did with permutation tests will automate the process. Here is a simple R program that collects 1000 bootstrap samples and finds the means of each:

```
R <- 1000
bootmean <- numeric(R)

x <- c(23.3,26.1,19.0,28.8,29.0)
mean(x)     # mean of the original sample

for (i in 1:R) {
  bootsample <- sample(x, size=length(x), replace=TRUE)
  bootmean[i] <- mean(bootsample)
  }
```

We'll try this out in class and see what we get. ◄

The bootstrap idea

The original sample represents the population from which it was drawn. So, resamples from *this* sample represent what we would get if we took many resamples from the population.
The bootstrap distribution of the statistic, based on many resamples, represents the sampling distribution of the statistic.

Why does bootstrapping work? It might seem that bootstrapping creates data out of nothing. This seems a little suspicious. But we are not using the resampled observations as if they were real data—the bootstrap is *not* a substitute for gathering more data to improve accuracy. Instead, the bootstrap idea is to use the resample means to estimate how the sample mean from a sample of size $n = 5$ from this population varies because of random sampling.

Using the data twice—once to estimate the population mean $\bar{x}$, and again to estimate the variation in the sample mean—is perfectly legal. In fact, you've done this many times before when you used the same sample to calculate both $\bar{x}$ and $s/\sqrt{n}$. What is different now is that:

1. We compute the standard error $SE_{\bar{x}}$ via resampling instead of the formula $s/\sqrt{n}$.
2. We use the bootstrap distribution to see if the sampling distribution is approximately normal, rather than just hoping that *n* is large enough for the CLT to apply.

Moreover, the beauty of this method is that it applies to statistics other than the sample mean. We can use bootstrapping in situations where traditional statistical approaches completely break down. For example, we can use bootstrapping to estimate the standard error and sampling distribution of

- the sample median
- a ratio of sample means
- a difference in sample means
- a ratio of sample medians
- a correlation (Pearson or Spearman, you pick!)
- a regression slope
- a ratio of regression slopes
- etc, etc, etc…

Bootstrapping is highly flexible and adaptable to many scenarios. I will show you some applications in the next couple of lectures.

20.2 Standard errors and sampling distributions via bootstrapping

The **standard error (SE) of a statistic** is a measure of the variability that the statistic has due to random sampling. In short, it is the standard deviation of the statistic. Knowing the standard error is crucial for quantifying the level of certainty (or uncertainty) we may place in our sample estimate.

The **bootstrap standard error (SE) of a statistic** is just the standard deviation of the bootstrap distribution of that statistic. I illustrate this using the previous example:

```
x <- c(23.3,26.1,19.0,28.8,29.0)
se.trad <- sd(x)/sqrt(length(x))
se.trad          # SE of the mean by traditional formula
[1] 1.874193

for (i in 1:R) {
   bootsample <- sample(x, size=length(x), replace=TRUE)
   bootmean[i] <- mean(bootsample)
   }

sd(bootmean)     # SE of the mean estimated by bootstrapping
[1] 1.672297
```

The standard error of $\bar{x}$ via the traditional formula is SE = 1.874. However, a bootstrap estimate of the standard error based on 1000 bootstrap samples is SE = 1.672.

Sampling distributions

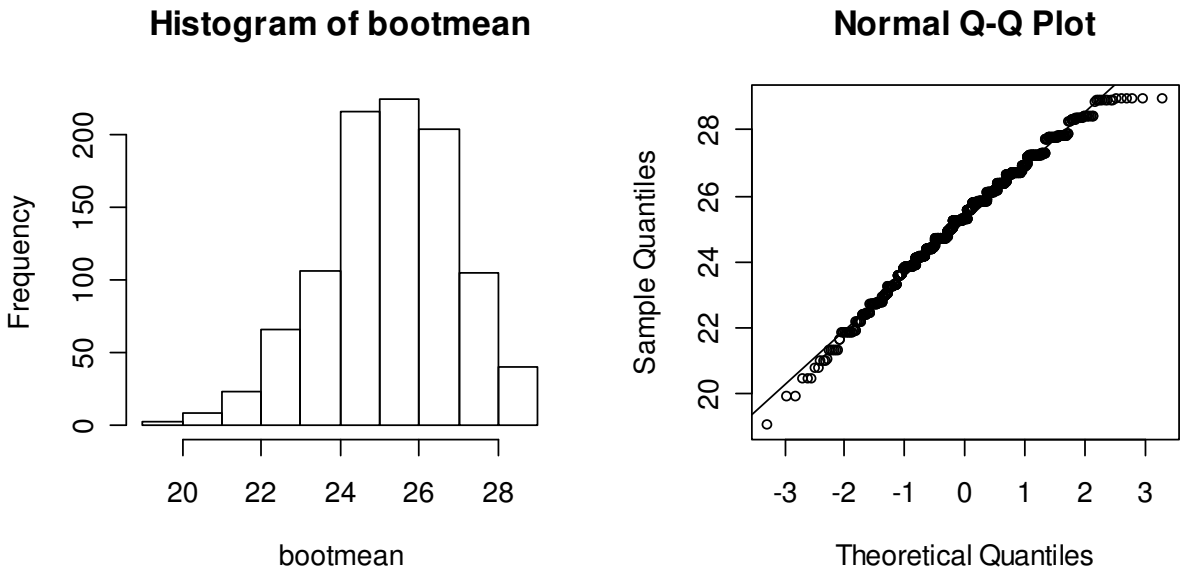
We should also inspect the *shape* of the sampling distribution of our statistic.

Why is this important? Recall from introductory statistics that the **sampling distribution of a statistic** is the distribution of *all possible values* that the statistic could take on from *all possible samples* of a given fixed size *n*. It is an important concept to wrap your brain around, because every confidence interval and hypothesis test ever conceived is based upon the idea of the sampling distribution of a statistic. In traditional statistics, we rely on a preconceived model for specifying the shape of the population, probability theory and certain assumptions to ensure that the sampling distribution takes on a particular shape (e.g. normal).

But in many settings, we have no model for the population. We then can't appeal to probability theory, so traditional approaches get stuck. This is where bootstrapping can save you.

► **"Baby steps" example.** We may check the overall shape of the sampling distribution of $\bar{x}$ (as approximated by bootstrapping) by drawing a histogram and a normal quantile-quantile plot of the bootstrap distribution. Here it is for the "baby steps" example (I've already generated the vector of bootstrapped means, called `bootmean`):

```
> par(mfrow=c(1,2))    # sets up a plot window for two side-by-side plots
> hist(bootmean)       # make a histogram of the bootstrapped means
> qqnorm(bootmean)     # make a normal Q-Q plot of the bootstrapped means
> qqline(bootmean)     # add a reference line for assessing normality
```



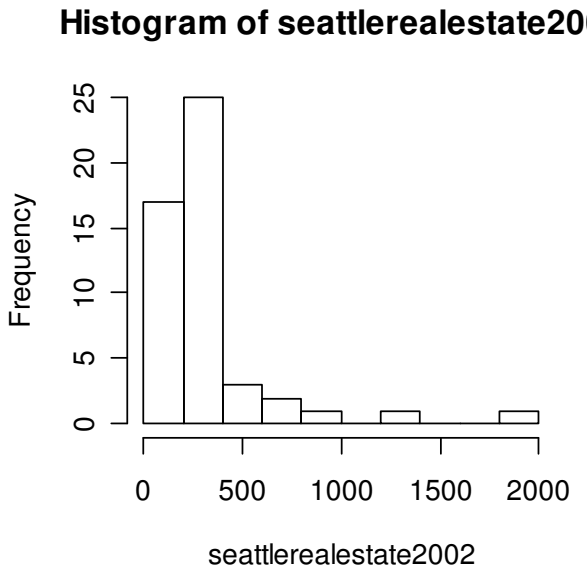
The sampling distribution appears to be left skewed. ◄

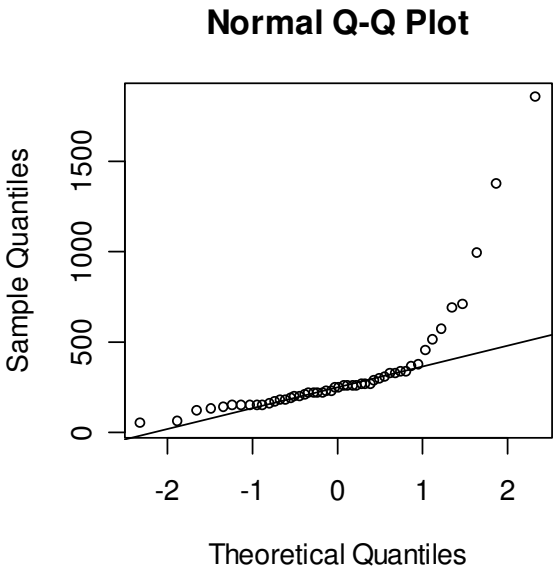
So you see that bootstrapping can be used to estimate the variability *and* the overall distribution of a statistic. This will eventually enable us to form **nonparametric confidence intervals** for population parameters. I conclude with a different example.

► **Example: Seattle real estate prices.** We are interested in the sales prices of residential properties in Seattle. Unfortunately, the data available from the county assessor's office do not distinguish residential properties from commercial properties. While most of the sales were residential, the fewer large commercial sales in the sample can greatly increase the mean selling price. The data are for 2002, and appear in the R workspace `seattlerealestate2002`. Investigate both the sample mean and the sample median as ways of characterizing "typical" sales prices.

Solution. Let's begin by looking at the data using a histogram and a normal quantitle-quantile plot. First open the workspace.

```
> par(mfrow=c(1,2))
> hist(seattlerealestate2002)
> qqnorm(seattlerealestate2002)
> qqline(seattlerealestate2002)
```





It's pretty easy to see the impact of the commercial properties: they make the sample (representative of all property sales in Seattle) highly right skewed. The normal Q-Q plot indicates strong non-normality in the data.

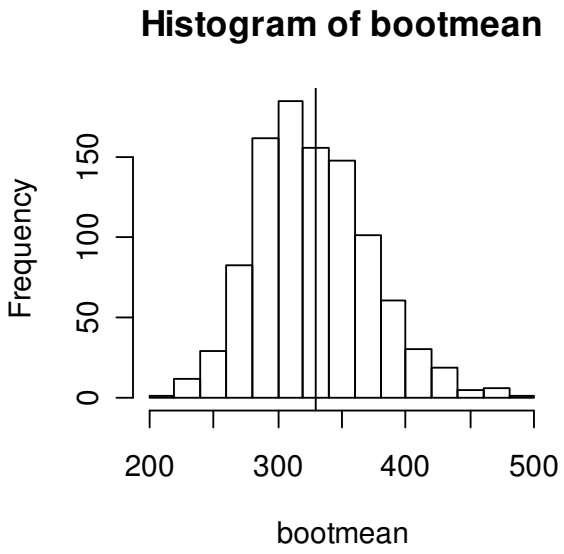
Let's collect 1000 bootstrap samples and generate the bootstrap distributions of both the sample mean and sample median. Here's an R program to do it:

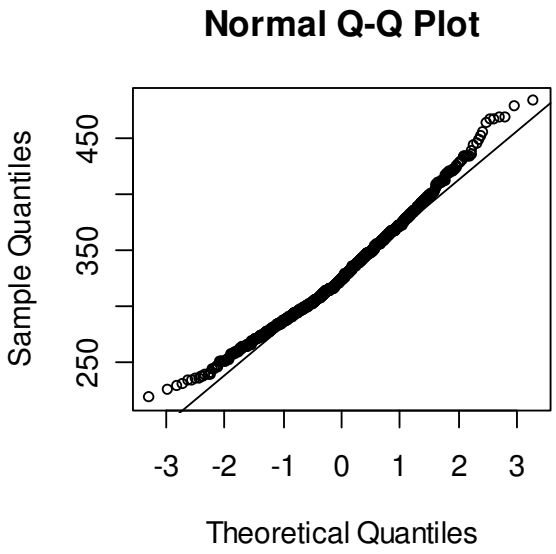
```
d <- seattlerealestate2002    # shorten the dataframe name!

R <- 1000
bootmean <- numeric(R)
bootmedian <- numeric(R)

for (i in 1:R) {
  bootsample <- sample(d, size=length(d), replace=TRUE)
  bootmean[i] <- mean(bootsample)
  bootmedian[i] <- median(bootsample)
  }

sd(bootmean)      # SE of the mean estimated by bootstrapping
sd(bootmedian)    # SE of the median estimated by bootstrapping
```

Run this and see what you get. Which statistic (the mean or median) might provide a more stable estimate of "typical" sales price?

Here is a visualization of the sampling distribution of the sample **mean** via bootstrapping:

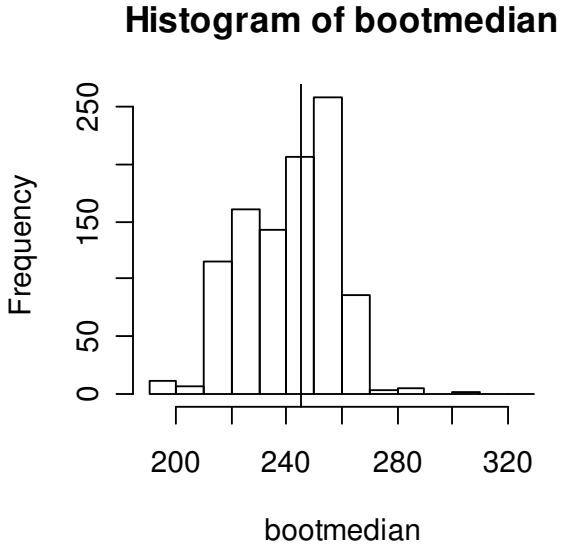
```
par(mfrow=c(1,2))     # set up for two plots side by side
hist(bootmean)        # histogram
abline(v=mean(d))     # lay in a vertical line at the actual sample mean
qqnorm(bootmean)      # create a normal Q-Q plot
qqline(bootmean)      # lay in a normal reference line
```

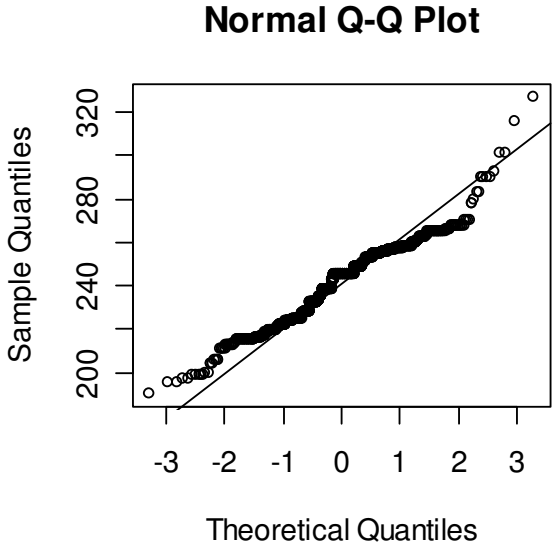




And here is the sampling distribution of the sample **median** via bootstrapping:

```
par(mfrow=c(1,2))
hist(bootmedian)
abline(v=median(d))
qqnorm(bootmedian)
qqline(bootmedian)
```





Does either statistic appear to behave normally? What about the precision of each statistic? Make some assessments about which statistic you would prefer to use to form a confidence interval for residential sales. ◄

Lecture 20 Practice Exercises

A random sample of 300 customer weekday noon-hour waiting times at a downtown Columbus Chipotle restaurant were recorded (in minutes). The data are in the R data frame `waittime.Rdata`.

1. Make a histogram of the sample and use it to describe the distributional pattern of the population of waiting times. Also, calculate and interpret the sample mean wait time $\bar{x}$.

2. Create a single bootstrap sample from the waiting times.

3. Estimate the SE and sampling distribution of the sample mean $\bar{x}$ using bootstrapping. Use 1000 bootstraps. Describe the shape of the sampling distribution of the sample mean $\bar{x}$. Is it approximately normal? Are you surprised? Why or why not?

4. With the observed sample, the traditional estimate for the SE of $\bar{x}$ is 0.05438. How does the bootstrap estimate of the SE of $\bar{x}$ compare? What might you attribute the difference to?

5. Estimate the SE and sampling distribution of the sample variance s^2 (the square of the standard deviation) using bootstrapping. Use 1000 bootstraps. Describe the shape of the sampling distribution of the sample variance. Is it approximately normal?

STA333 Lecture 21
Bootstrap confidence intervals (part 1)

21.1 Preliminaries: an introduction to the notion of bias

One of the fundamental uses of the bootstrap is to find confidence intervals for population parameters. In traditional statistics when required assumptions are met, the approach to doing this is formulaic, but if we are dealing with a situation where usual assumptions are violated and/or we are trying to estimate some atypical parameter, the bootstrap can provide us with a mechanism for constructing confidence intervals for such parameters *nonparametrically*.

We said in the last lecture that the shape of the bootstrap distribution approximates the true shape of the sampling distribution of the statistic we are using. So, for example, we can use the bootstrap distribution to check for normality. If the sampling distribution appears to be normal and centered at the true parameter, we can use the bootstrap standard error to calculate a *t*-based confidence interval. So, it turns out that we need to use the bootstrap to check the **center** of the sampling distribution as well as the shape and spread. As it turns out, the bootstrap does not reveal the center directly, but rather reveals the **bias**.

Bias. A statistic used to estimate a parameter is biased if its sampling distribution is not centered at the true value of the parameter being estimated. The bias of a statistic is found thus:

$$bias = mean\ of\ sampling\ distribution - true\ parameter\ value$$

Of course, we do not know the true parameter value, so we cannot directly calculate the bias. However, the bootstrap method allows us to check for bias by seeing whether the bootstrap distribution of a statistic is centered at the statistic's value from the original random sample. The bootstrap estimate of bias is given by

bootstrap estimate of bias = mean of bootstrap distribution – statistic for original data

► **A normal example using the sample mean.** It is a known statistical fact that the sample mean $\bar{x}$ is an unbiased estimator for the population mean μ, i.e. the true bias is 0. Let's run a little example using some randomly generated normal data to check this using bootstrapping.

The steps are:

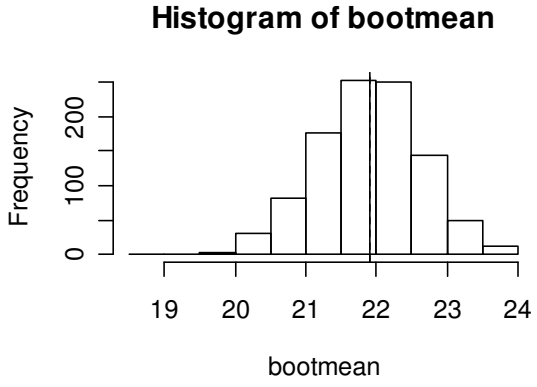
1. Randomly generate a sample of size $n = 50$ from the normal distribution with mean $\mu = 22$ and standard deviation $\sigma = 5$.
2. Calculate the sample mean $\bar{x}$ for our random sample.
3. Generate 1000 bootstrap samples from this sample, calculating the bootstrap sample mean for each.
4. From the bootstrap distribution for $\bar{x}$, calculate the estimated bias using the formula in the box on the previous page.

Here goes:

```
x <- rnorm(50,22,5)                          # step 1
xbar <- mean(x)                              # step 2

bootmean <- numeric(1000)                    # step 3
for (i in 1:1000) {
  bootsample <- sample(x, size=length(x), replace=TRUE)
  bootmean[i] <- mean(bootsample)
  }
est.bias <- mean(bootmean) - xbar            # step 4
est.bias
```

Upon running this R program, I found my estimate of bias to be 0.00497. I encourage all of you to do the same to see what results you get. Because you are resampling, you will all get different answers, but they should all be very close to 0. Here is a picture of what is happening:

```
hist(bootmean)                    # picture of the bootstrap distribution
abline(v=xbar)                    # add vertical line at the actual sample mean
abline(v=mean(bootmean), lty=2) # add dashed line mean of the bootstrap dist
```

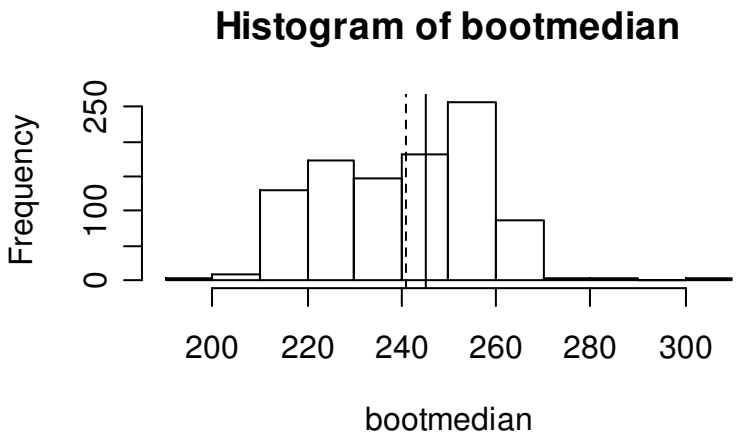


What about other kinds of statistics? Let's check for bias in the median estimate for the Seattle real estate problem from last lecture.

► **Example: Seattle real estate prices.** We are interested in the sales prices of residential properties in Seattle. Unfortunately, the data available from the county assessor's office do not distinguish residential properties from commercial properties. Because of this, we are considering the median selling price. (Remember that the data appear in the R workspace `seattlerealestate2002`.) Let's estimate the bias in using the sample median to estimate the true median selling price.

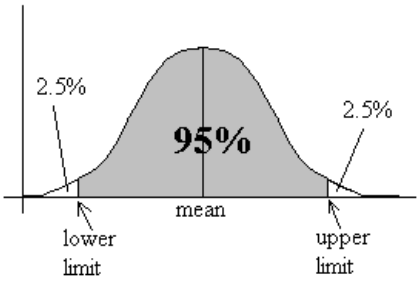
```
d <- seattlerealestate2002        # shorten the vector name!
median <- median(d)

bootmedian <- numeric(1000)
for (i in 1:1000) {
  bootsample <- sample(d, size=length(d), replace=TRUE)
  bootmedian[i] <- median(bootsample)
  }

est.bias <- mean(bootmedian) - median
est.bias                          # print the bias estimate

hist(bootmedian)                  # picture of the bootstrap distribution
abline(v=median)                  # add line at the actual sample median
abline(v=mean(bootmedian),lty=2) # add line at the mean of the bootstrap dist
```

My bias estimate was –3.9995. Upon running the program a few times, you will see that this is clearly a systematic bias in using the median. The bias is that the sampling distribution tends to underestimate the true median of the population. This is clearly shown in the picture below. Thus, a good confidence interval procedure using the bootstrap should try to compensate for the bias that we have estimated to be there. We'll see this soon. ◄



21.2 Basic bootstrap confidence intervals

In traditional (parametric) statistics, finding a CI for a mean is a snap once you have identified the sampling distribution of the statistic. Once you do that, you determine the lower and upper limits of (say) a 95% confidence interval by finding the appropriate 2.5% and 97.5% quantiles on the appropriate sampling distribution (e.g. a *t*-curve): see picture at right for an illustration.



We then calculated the CI using a formula, such as $\bar{x} \pm t_{0.025} \frac{s}{\sqrt{n}}$.

How do we find a nonparametric confidence interval? Well, the notion is essentially the same, except we use the bootstrap distribution as our estimate of the actual sampling distribution of the statistic. Once we find the 2.5% and 97.5% quantiles of the bootstrap distribution, we have essentially found a 95% confidence interval for the parameter. This type of interval is known as a **bootstrap percentile confidence interval**. It is easy to do in R using the `quantile()` function.

A little later, we will see that there are built-in routines in R for bootstrapping automatically and then calculating bootstrap CIs. We'll start, however, by building one ourselves using R code.

► **A normal example using the mean.** Let's find a 95% CI for the mean of the population using our simulated data. Here are the things we know:

1. We randomly sampled $n = 50$ observations from a normal population with mean $\mu = 22$ and standard deviation $\sigma = 5$. Thus, we know that the true mean is 22. Hopefully our CI captured this!
2. Since $\bar{x}$ is an unbiased estimator for the population mean μ, we do not need to worry about bias messing up the CI results.
3. Since the normality assumption is met and the estimator we use is unbiased, we can see how consistent the bootstrap results are with the usual parametric results you'd get using a t-based CI.

Here's the program to do it:

```
> x <- rnorm(50,22,5)
>
> bootmean <- numeric(1000)
> for (i in 1:1000) {
+   bootsample <- sample(x, size=length(x), replace=TRUE)
+   bootmean[i] <- mean(bootsample)
+   }
> quantile(bootmean, c(0.025, 0.975))

    2.5%    97.5%
19.95437 22.44248
```

I got (19.95, 22.44) as my 95% nonparametric bootstrap CI for μ. Let's compare this to the usual t-based CI from the formula $\bar{x} \pm t_{0.025} \frac{s}{\sqrt{n}}$. This is easy to get using `t.test()`:

```
> t.test(x)

        One Sample t-test

data:  x
t = 32.2378, df = 49, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 19.91887 22.56728
sample estimates:
mean of x
 21.24308
```

Buried in this R output is the 95% parametric CI result: (19.92, 22.57). We see that the parametric and nonparametric results are very consistent. ◄

► **Example: Seattle real estate prices.** We now find a 95% *bootstrap percentile confidence interval* for the median of Seattle real estate selling prices in 2002.

```
> d <- seattlerealestate2002        # shorten the vector name!
>
> bootmedian <- numeric(1000)
> for (i in 1:1000) {
+   bootsample <- sample(d, size=length(d), replace=TRUE)
+   bootmedian[i] <- median(bootsample)
+   }
> quantile(bootmedian, c(0.025, 0.975))
   2.5%   97.5%
213.975 266.000
```

Using this method, we can be 95% confident that the true median selling price of residential properties in Seattle in 2002 was between $213,975 and $266,000. ◄

A few notes:

1. Try finding a 95% CI for the mean using both a *t*-based approach and the bootstrap approach for the last example. Compare the results and comment on what you see.
2. We can even improve on the interval result above. How? Remember that the median was biased. Our method (percentile-based bootstrap) does not account for this bias. Next time we will see an improved method using bootstrapping that does account for the estimated bias
3. The R add-on package `boot` handles many kinds of bootstrap problems and calculates bootstrap CIs automatically. On the next page I will re-do the Seattle real estate problem using the `boot` package.

► **Example: Seattle real estate prices.** I re-do the problems in this section but using the R add-on package `boot`. The code below will be discussed in class.

```
> d <- seattlerealestate2002        # shorten the vector name!
>
> library(boot)
> mymedian <- function(d,i)  median(d[i])
> myboot <- boot(d, mymedian, R=1000)
> myboot

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = d, statistic = mymedian, R = 1000)

Bootstrap Statistics :
    original    bias    std. error
t1*  244.925 -4.304875    16.26544
```

The original sample median is 244.925, the estimated bias is –4.305, and the SE of the bootstrap distribution is 16.27.

We now ask for the 95% percentile-based bootstrap confidence interval for the median. This is done via the `boot.ci()` function in the `boot` package:

```
> boot.ci(myboot, type="perc")

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = myboot, type = "perc")

Intervals :
Level     Percentile
95%   (213.2, 266.0 )
Calculations and Intervals on Original Scale
```

The 95% confidence interval is found to be (213.2, 266.0). ◄

We will see more of this next time, along with several different problems in varying contexts and by using several different statistics. In the meantime, do the following:

1. Check the help page by typing `?boot` in R.
2. Visit http://www.statmethods.net/advstats/bootstrapping.html on the Quick-R website.
3. Visit http://www.mayin.org/ajayshah/KB/R/documents/boot.html for another writeup on the basics of the `boot` package.

Lecture 21 Practice Exercises

A random sample of 300 customer weekday noon-hour waiting times at a downtown Columbus Chipotle restaurant were recorded (in minutes). The data are in the R data frame `waittime.Rdata`.

1. Find and interpret a 95% bootstrap confidence interval for the true mean weekday noon-hour waiting times at the downtown Columbus Chipotle restaurant.

2. Repeat problem 1, but using the R add-on package `boot`.

3. Recall from the Lecture 20 practice exercises that the sampling distribution of the sample mean (via bootstrapping) appeared fairly normal, except for a bit of discrepancy in the tails of the distribution. Due to this, go ahead and find the usual 95% *t*-based CI for the true mean weekday noon-hour waiting times at the downtown Columbus Chipotle restaurant. How does the traditional approach result vary from the bootstrap CI from problem 1?

4. Find and interpret a 95% bootstrap confidence interval for the true median weekday noon-hour waiting times at a downtown Columbus Chipotle restaurant. Estimate the bias and comment. How does this CI differ from the CI you got for the mean in problem 1? Can you explain why the difference exists?

STA333 Lecture 22
Bootstrap confidence intervals (part 2)

22.1 Better bootstrap CIs: the bias-corrected accelerated (BCa) interval

No method for obtaining CIs exactly the intended confidence level in practice. When you compute what is supposed to be, say, a 90% CI, the method you use may in fact give intervals that capture the true parameter value less often, say 87% of the time. Or instead of "missing" 5% of the time on each side, a method may in some settings miss 3% of the time on one side and 7% of the time on the other, giving a biased picture of where the true parameter is.

Accuracy. We say that a method for obtaining a 90% CI is **accurate** in a particular setting when 90% of the time it produces an interval that covers the true parameter value, and produces intervals that "miss" the true parameter value 5% of the time on the high side and 5% of the time on the low side. No CIs are perfectly accurate in practice because the conditions under which they work are never perfectly satisfied. The common culprits resulting in inaccurate CIs are:

1. sampling from a highly skewed population; or,
2. using a statistic that is a biased estimator of its corresponding population parameter.

One advantage of bootstrapping is that it allows you to check for skewness in the sampling distribution. When this skewness is present, it can produce a **CI bias** in the direction of the skew. So, a percentile-based bootstrap CI may not be accurate enough if:

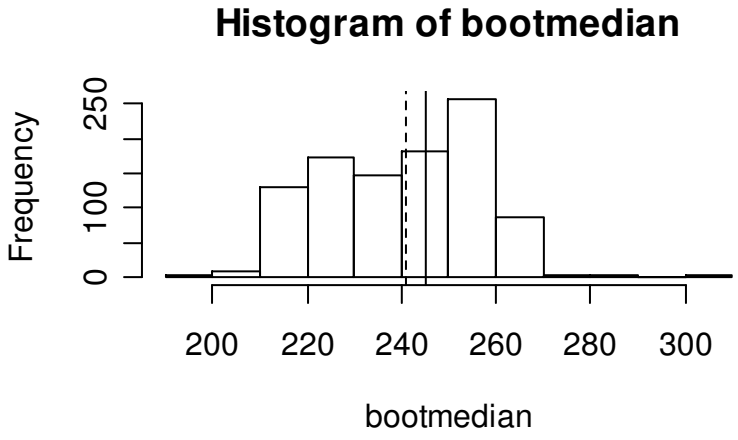
- The statistic itself is strongly biased, as indicated by the bias estimate from the bootstrap,
- The sampling distribution of the statistic is clearly skewed, as indicated by the bootstrap distribution itself, or
- High accuracy is needed because the stakes are high (e.g. large amounts of money, public welfare, etc.)

The BCa interval

The BCa, or *bias-corrected and accelerated bootstrap* confidence interval, is a modification of the percentile-based bootstrap confidence interval. **BCa CI endpoints are percentiles of the bootstrap distribution that are adjusted to correct for bias and skewness in the distribution.** For example, if the statistic is biased upward (that is, it tends to be too large), the BCa bias correction shifts the endpoints to the left. If the bootstrap distribution is skewed to the right, the BCa incorporates a correction to shift the endpoints even farther to the right (this may seem counterintuitive, but it is the correct action).

Details of these computations are pretty advanced stuff, so we will rely on software to calculate BCa intervals. In R, BCa intervals may be found using the `boot.ci()` function in the `boot` package. Ask for `method="bca"` rather than `method="perc"`.

► **Example: Seattle real estate prices.** We earlier saw (and estimated) the downward bias in the median for these data. Remember this? –




```
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = d, statistic = mymedian, R = 1000)

Bootstrap Statistics :
    original    bias    std. error
t1*  244.925 -4.304875    16.26544
```

The estimated bias was –4.305. Because of this, a 95% percentile-based bootstrap confidence interval for the median may not be accurate. So, we instead ask for the BCa interval. Just for kicks, I'll ask for both types from R:

```
> boot.ci(myboot, type=c("perc", "bca"))


BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates
```

```
CALL :
boot.ci(boot.out = myboot, type = c("perc", "bca"))

Intervals :
Level     Percentile            BCa
95%   (213.2, 266.0 )   (213.0, 265.5 )
Calculations and Intervals on Original Scale
```

The 95% BCa bootstrap interval is (213.0, 265.5). The lack of a noticeable difference between the percentile-based CI and the BCa CI actually suggests that the bias does not have an impact on the results. ◄

22.2 Parting comments on the bootstrap

Some comments are in order before we look at several examples using bootstrapping in the next section. Here goes:

1. Bootstrapping and conclusions based on them include *two* sources of random variation:
   - The original sample was chosen at random from the population.
   - Bootstrap samples are chosen at random from the original sample.

2. For most statistics, **almost all of the variation in bootstrap distributions comes from the selection of the original sample**, *not* from what bootstrap samples you happen to select. A bootstrap resampling process using 1000 or more resamples introduces very little additional variation to the outcome.

3. While bootstrapping allows us to relax assumptions like normality, **you must still respect the structure of your data**; any form of dependence in the data must be taken into account. For instance, how you draw bootstrap samples for a comparison of means using independent samples must be done differently than if you had paired samples.

4. **Bootstrapping is *not* a cure for tiny sample sizes!** If your sample size n is small, there is no getting around the fact that you do not have much information from the population from which to generate an accurate or usable inference. *This is always true, whether you are bootstrapping or not!* As always, bigger samples yield more trustworthy results.

5. **Bootstrapping does not work on <u>all</u> kinds of statistics.** In particular, I recommend against using the bootstrap for statistics that are largely functions of the sample extremes: for example, the minimum value, the maximum value, the range, etc. Bootstrapping works best on statistics that are functions of the *entire* sample: this includes means, standard deviations, correlations, regression-based quantities such as model coefficients and predictions, etc. It also works very well for medians and quartiles, though usually larger samples are desirable for bootstrapping these statistics.

22.3 Examples

Now, several examples using bootstrapping in different context.

► **Example: Diet comparisons.** These data were first presented in Lecture 11. In a comparison of the effect on growth of two diets ***A*** and ***B***, a number of growing rats were placed on these two diets, and the following growth figures were observed after 7 weeks:

A	156	183	120	113	138	145	142			
B	109	107	119	162	121	123	76	111	130	115

Earlier, we did hypothesis tests of H_0: $\mu_A = \mu_B$ versus H_a: $\mu_A \neq \mu_B$ using both a parametric independent samples *t*-test and a nonparametric permutation test. Now, let's use bootstrapping to estimate, with 90% confidence, the *true mean difference in growth* between diets ***A*** and ***B***.

Solution. To do this, we can enter the samples into two separate R vectors, bootstrap each one, and then calculate the difference between the means of the bootstrapped samples. **We bootstrap each sample independently because the samples are independent** (see Comment 3 in the previous section).

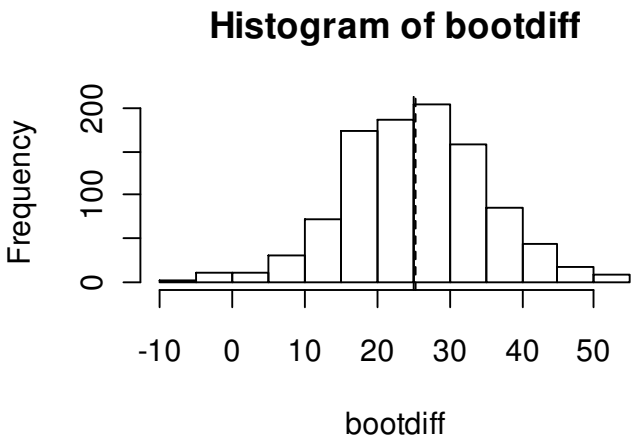
The true mean difference in growth is $\mu_A - \mu_B$. So, we use $\bar{x}_A - \bar{x}_B$ as our statistic. The steps are:

1. Collect a bootstrap sample from the diet ***A*** sample. Calculate $\bar{x}_A$ from this.
2. Collect a bootstrap sample from the diet ***B*** sample. Calculate $\bar{x}_B$ from this.
3. Calculate $\bar{x}_A - \bar{x}_B$. This is a bootstrapped mean difference.
4. Repeat steps 1-3 a very large number of times (say 1000). This produces our bootstrap distribution for $\bar{x}_A - \bar{x}_B$.
5. Form a CI for $\mu_A - \mu_B$ from the bootstrap distribution.

Here is an R program to do the bootstrapping, including a check of bias:

```
A <- c(156,183,120,113,138,145,142)
B <- c(109,107,119,162,121,123,76,111,130,115)
obs.diff <- mean(A) - mean(B)

bootdiff <- numeric(1000)

for (i in 1:1000) {
  bootsampleA <- sample(A, size=length(A), replace=TRUE)
  bootsampleB <- sample(B, size=length(B), replace=TRUE)
  bootdiff[i] <- mean(bootsampleA) - mean(bootsampleB)
  }
est.bias <- mean(bootdiff) - obs.diff
est.bias

hist(bootdiff)                  # picture of the bootstrap distribution
abline(v=obs.diff)              # add vertical line at the actual sample mean diff
abline(v=mean(bootdiff),lty=2)  # add dashed line mean of the bootstrap dist
```

Upon running, I got a bias estimate very close to zero. The plot of the bootstrap distribution reveals that bias is not an important issue here:

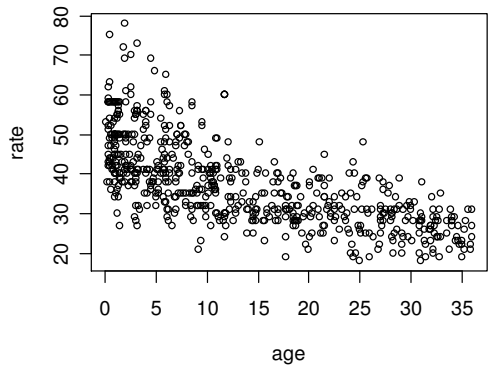


Now, I calculate a 90% percentile-based bootstrap CI for $\mu_A - \mu_B$:

```
> quantile(bootdiff, c(0.05, 0.95))
       5%       95%
 9.826429 41.687857
```

We can be 90% confident that the true mean growth due to diet ***A*** is between 9.82 to 41.68 higher than for diet ***B***. ◄

► **Example: Respiratory rates in children.** A high respiratory rate is a potential diagnostic indicator of respiratory infection in children. To judge whether a respiratory rate is truly "high," however, a physician must have a clear picture of the distribution of normal respiratory rates. To this end, Italian researchers measured the respiratory rates of $n = 618$ children between the ages of 15 days and 3 years (given in months). The data appear in the R workspace `respiratory`. Here is a plot of the data:



Find a 95% bootstrap confidence interval for the true rank (Spearman) correlation relating age to respiratory rate.

Solution. To do this, we must perform the bootstrapping by resampling the children. In this context, that means **we must bootstrap entire rows of the R dataframe**. This will retain the connection between each child's age and their respiratory rate.

For this example, I will demonstrate by using the `boot` package in R. After opening the dataframe in R, run the following program. The code will be explained in class, but brief comments are included in-line below.

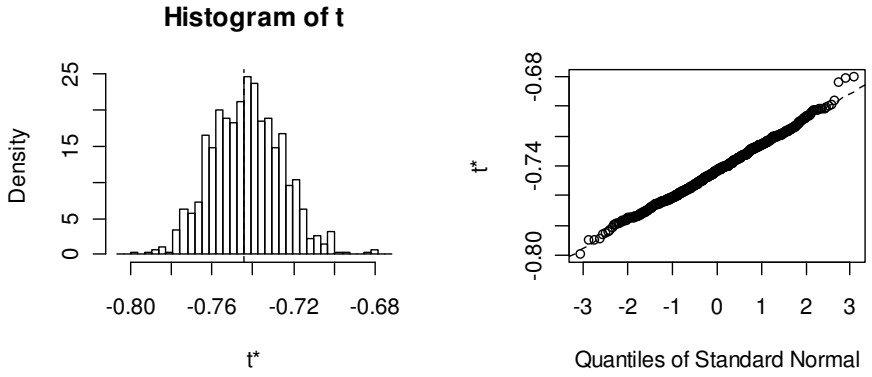
```
library(boot)

# First, create a function to obtain Spearman correlations from the data:
mycorr <- function(data, indices) {
               d <- data[indices,]
               return(cor(d$age, d$rate, method="spearman"))
             }

# Next, run the bootstrapping using 1000 replications:
results <- boot(data=respiratory, statistic=mycorr, R=1000)

# Finally, view results of bootstrapping:
results
```

Here's the output for the code above:

```
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = respiratory, statistic = mycorr, R = 1000)

Bootstrap Statistics :
      original      bias    std. error
t1* -0.7445018 0.001416301  0.01725391
```

The Spearman correlation between rate and age is -0.7445. The standard error of the correlation estimate is 0.0173, and the bias seems negligible. We can plot the bootstrap distribution using the `plot()` function applied to the `boot` object that we named `results`:

```
plot(results)
```



*t** is a generic name that `boot` gives to the bootstrapped statistic. In this problem, the bootstrap distribution actually appears reasonably normal! Now, the CI:

```
> boot.ci(results, type=c("perc","bca"))

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = results, type = c("perc", "bca"))

Intervals :
Level     Percentile            BCa
95%   (-0.7745, -0.7079 )   (-0.7766, -0.7115 )
Calculations and Intervals on Original Scale
```

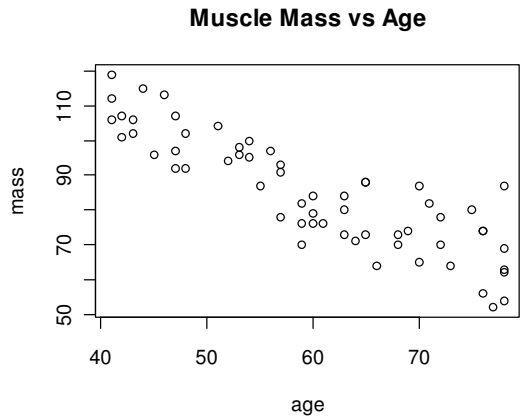
We can be 95% confident that the true rank correlation between age and respiratory rate for children aged 15 days and 3 years is between –0.77 and –0.71. Thus, there is statistical evidence of a quite strong negative association between age and respiratory rate. Due to the symmetry and lack of bias, either bootstrap CI is fine to use here. ◄

► **Example: Muscle mass.** A person's muscle mass is expected to decrease with age. To explore this relationship in women, a nutritionist randomly selected 15 women from each 10-year age group beginning with age 40 and ending with age 79. The data reside in the R workspace `musclemass`. The variables in the dataset are `mass` and `age`. Let's do the following:

1. Investigate the relationship visually using a scatterplot.
2. Fit a simple linear regression model relating muscle mass to age.
3. Find a 95% bootstrap CI for the true mean rate of change in muscle mass for each additional year of age. (In other words, find a 95% bootstrap CI for the regression slope.)

Solution. First, here's the scatterplot:

```
> plot(mass~age, data=musclemass, main="Muscle Mass vs Age")
```



We can clearly see the negative trend one would expect: as age increases, muscle mass tends to decrease. You should also notice that it decreases in roughly a linear fashion, so it is reasonable to fit a simple linear regression model to this data. (However, this point is arguable … we may investigate this in the next lecture!)

We now fit the model in R. To do so, the `lm()` function may be used, followed by a `summary()` function call to see results of the regression fit:

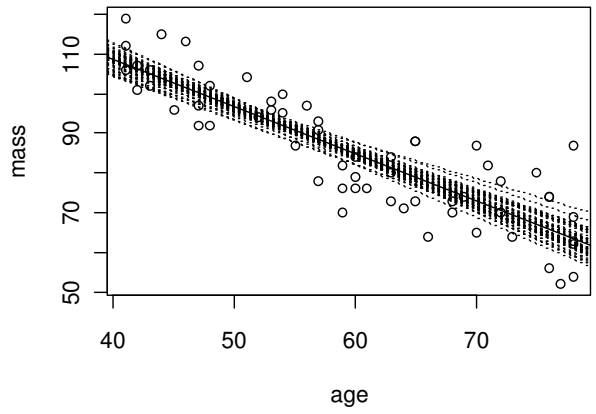
```
> fit <- lm(mass~age, data=musclemass)
> summary(fit)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 156.3466     5.5123   28.36   <2e-16 ***
age          -1.1900     0.0902  -13.19   <2e-16 ***
```

The fitted regression model is $\widehat{mass} = 156.345 - 1.19(age)$. The slope estimate is $b_1 = -1.19$. This means that we estimate a drop of 1.19 in mean muscle mass per year of age in women aged 40-79.

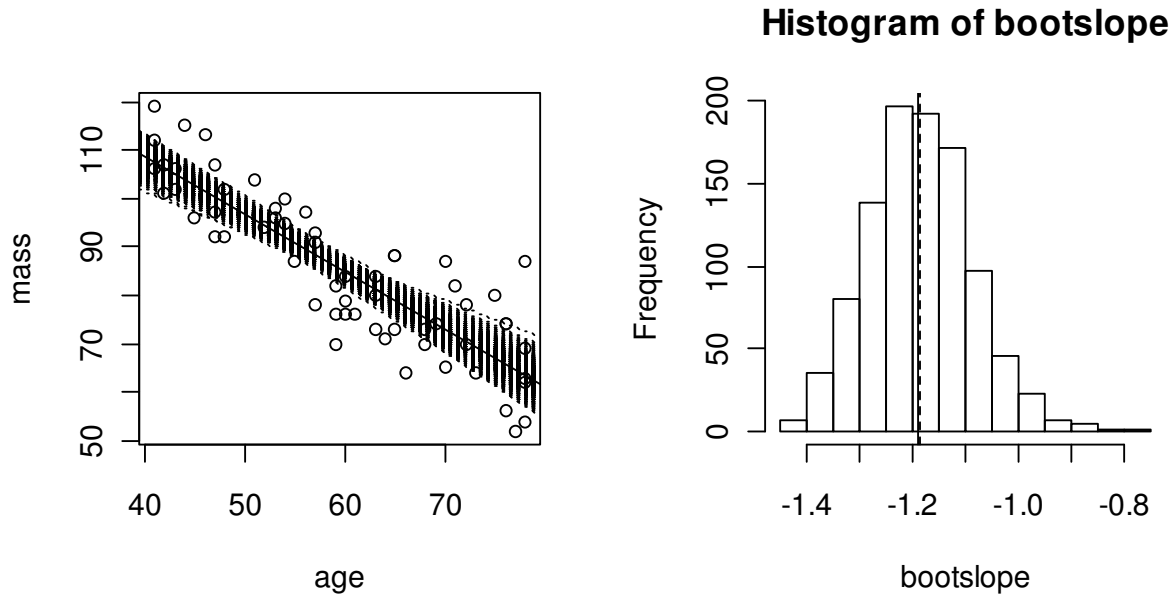
What we want now is a 95% confidence interval for the true mean rate of change in muscle mass for each additional year of age. There is a parametric way of doing this using a *t*-based CI, but if the usual regression assumptions aren't met, a bootstrap CI may be used instead. This approach is detailed below. The following R program was built by yours truly from the ground up. I will explain it in class in detail, but one of the nice things it does is show you the bootstrap distribution of the slopes visually by plotting all the bootstrapped slopes over the scatterplot. (Kinda cool, if you ask me.)

```
## A sexy nonparametric bootstrap program for the slope in a simple linear regression.
## This example uses the R dataframe "musclemass".

par(mfrow=c(1,2))                # set up two panels for our plots
bootslope <- numeric(1000)       # create vector to hold 1000 bootstrapped slopes

plot(mass~age, data=musclemass)        # make scatterplot of the data points
fit <- lm(mass~age, data=musclemass)   # fit the simple linear regression to orig data
summary(fit)                           # print a summary of fitted model results
slope <- summary(fit)$coeff[2]         # extract the slope estimate
abline(fit)                            # draw in the fitted line

for (i in 1:1000) {
  n <- nrow(musclemass)
  index <- sample(1:n, size=n, replace=TRUE)
  bootdata <- data.frame(mass=musclemass$mass[index],age=musclemass$age[index])
  bootfit <- lm(mass~age, data=bootdata)
  bootslope[i] <- summary(bootfit)$coeff[2]     # extract bootstrapped slope
  abline(bootfit, lty=2)                        # draw in bootstrapped slope on plot
   }

hist(bootslope)                          # picture of the bootstrap dist of slopes
abline(v=slope)                          # add vertical line at the actual sample slope
abline(v=mean(bootslope),lty=2)          # add dashed line at mean of the bootstrap dist
quantile(bootslope, c(0.025, 0.975))     # percentile-based 95% bootstrap CI for slope
```

We'll run this in class (and do so yourself, too). For the sake of illustration, here is the scatterplot with just 50 bootstrapped slopes superimposed. This provides you with a sense of the variation you might expect in the slope if you had collected a different random sample (of the same size) of women:



A full run of the program using 1000 bootstraps gives this:



There appears to be a very minor right skew in the distribution of slope estimates (can you look at the scatterplot and guess why?). The percentile-based bootstrap CI is given below:

```
> quantile(bootslope, c(0.025, 0.975))

      2.5%      97.5%
-1.3698876 -0.9774157
```

The 95% CI is (–1.37, –0.97). We can be 95% confident that each additional year of age will result in a mean decrease in muscle mass of between 0.97 to 1.37. This CI is entirely below 0, so we are confident that muscle mass and age are negatively linearly related in women aged 40-79.

Side note. The usual parametric *t*-based CI for the regression slope can be found by applying the `confint()` function in R to the original fitted model from `lm()`:

```
> confint(fit)
                 2.5 %     97.5 %
(Intercept) 145.312572 167.380556
age          -1.370545  -1.009446
```

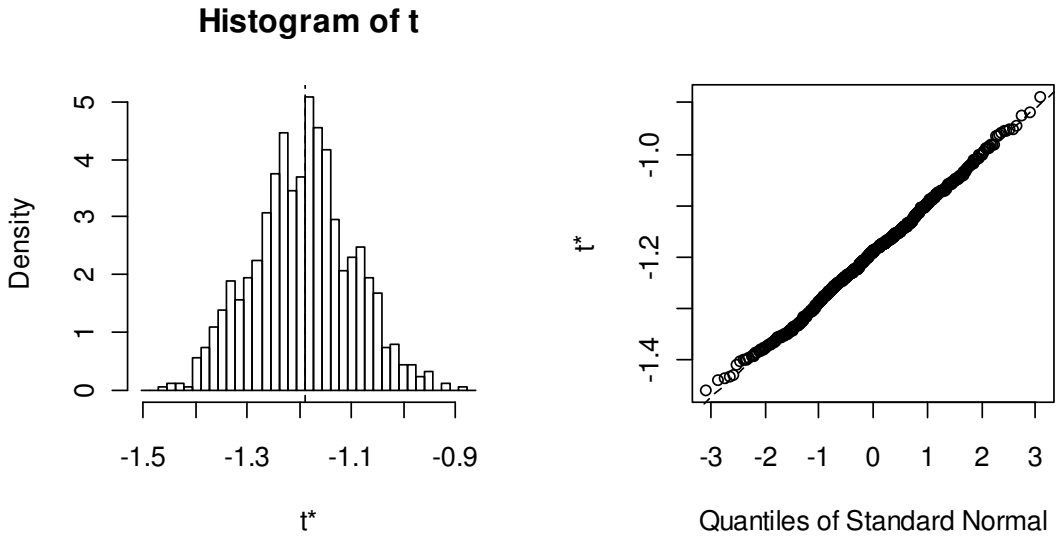
Note how close the parametric CI is to our bootstrap CI. This is because the standard regression assumptions were reasonably met in this case. ◄

► **Example: Muscle mass (again).** Here's the same problem I just did, but now done using the `boot` package in R:

```
library(boot)                                   # load the 'boot' package

# First, create a function to obtain the slope from the data:
myslopes <- function(data, indices) {
               d <- data[indices,]              # tells 'boot' to resample ROWS of data
               fit <- lm(mass~age, data=d)      # fit regression using bootstrap dataset
               return(summary(fit)$coeff[2])    # returns a bootstrapped slope estimate
}

# Next, run the bootstrapping using 1000 replications:
results <- boot(data=musclemass, statistic=myslopes, R=1000)

# Finally, view results of bootstrapping:
results
plot(results)

# ... and obtain 95% bootstrap CIs (both percentile and BCa):
boot.ci(results, type=c("perc","bca"))
```

Here are the results. Compare these to what I got above.

```
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = musclemass, statistic = myslopes, R = 1000)

Bootstrap Statistics :
     original       bias    std. error
t1* -1.189996 -0.002642710  0.09428257
```




```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = results, type = c("perc", "bca"))

Intervals :
Level     Percentile            BCa
95%   (-1.374, -1.001 )   (-1.370, -0.996 )
Calculations and Intervals on Original Scale
```

All methods (parametric *t*-based CI, percentile-based bootstrap CI, BCa bootstrap CI) produce highly consistent results here and lead to the same conclusions. ◄

Lecture 22 Practice Exercises

These practice problems use data sets from the examples in Lecture Notes section 22.3.

1. Use the diet comparisons data to calculate and interpret a 90% bootstrap confidence interval for σ_1/σ_2, the ratio of the population standard deviations between diets *A* and *B*. Use the ratio of the sample standard deviations s_1/s_2 as your bootstrapping statistic.

2. Use the muscle mass data to calculate and interpret a 99% bootstrap CI for the Pearson correlation. Use the R add-on package `boot` to do the problem. What is your bias estimate? Should you use a percentile-based bootstrap CI or a BCa bootstrap interval?

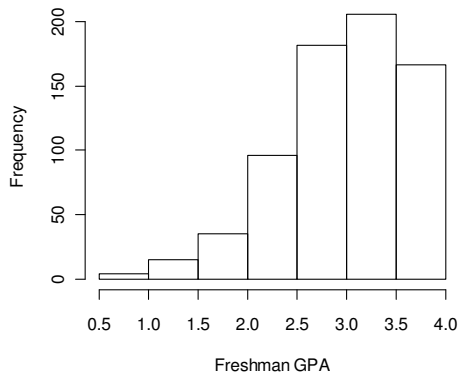
3. *Challenge problem:* Use the muscle mass data to calculate and interpret a 95% bootstrap CI for the true mean muscle mass for a 60-year old woman. (*Hint:* this prediction is found via the fitted regression equation $\hat{y} = b_0 + b_1(60)$. Look into the `predict()` function in R.)

STA333 Lecture 23
Kernel density estimation

23.1 The problem

In introductory statistics, the tool you typically use to picture the distribution of a set of values is a histogram. For example, in section 3.2 you saw how to get a histogram of the freshman GPAs in the `uadata` dataset:

```
> site <- "http://www.users.muohio.edu/hughesmr/sta333/univadmissions.txt"
> uadata <- read.table(site, header=TRUE)
> hist(uadata$gpa.endyr1, main=" ", xlab="Freshman GPA")
```



While this may suffice in many instances, histograms are actually fraught with potential problems. The goal of this lecture is to start from the basic idea of what a histogram is, seek what it attempts to do, and then employ "modifications" to try to improve it.

23.2 Estimating a density

Histograms are a way of estimating the **probability density**, or probability distribution, of a population of variable values by using a selected sample.

Remember from basic statistics that the **total area under a probability distribution equals 1**, since *areas* represents *probabilities* of outcomes for the variable in question.

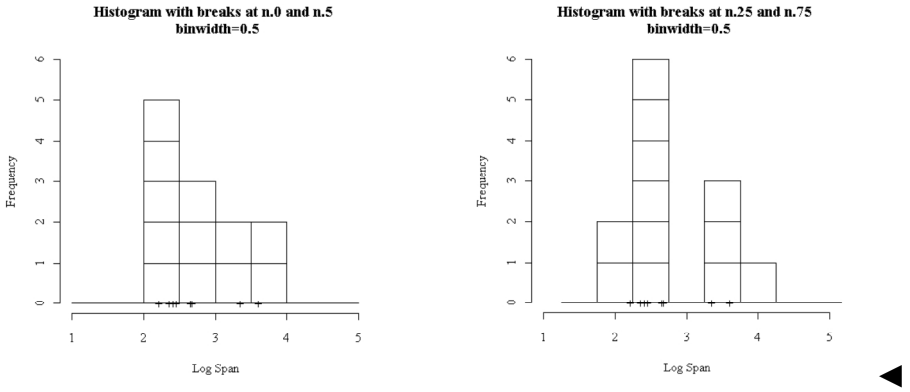
To construct a histogram, we divide the interval covered by the data values into equal sub-intervals, known as **bins**. Every time, a data value falls into a particular sub-interval, then a block, of size equal 1 by the bin width, is placed on top of it. When we construct a histogram, we need to consider these two main points: the size of the bins (the binwidth) and the end points of the bins.

► **Example: Wingspans.** These data are logs of wing spans of aircraft built from 1956-1984. (The complete dataset can be found in Bowman & Azzalini (1997) *Applied Smoothing Techniques for Data Analysis*. We use a subset of this, namely these observations:

2 22 42 62 82 102 122 142 162 182 202 222

I only use a subset for illustration; otherwise the plots become too crowded. The data points are represented by crosses on the *x*-axes below.

Below are two representations of the *same* density using the *same* sample. If we choose the starting break point at 0 and use a binwidth of 0.5, our histogram looks like the one on the left. It appears that this distribution is unimodal and skewed to the right. But if we use the same binwidth but with the starting break point shifted up to 0.25, then our histogram looks like the one on the right. We now have a completely different estimate of the density: it now appears to be bimodal (two peaks)!



◄

This example illustrates three issues that one encounters with traditional histograms. Namely,

- **Histograms are not smooth.** In the first example, GPAs are measured on a continuous scale, but the histogram doesn't necessarily indicate that. A histogram is more of a

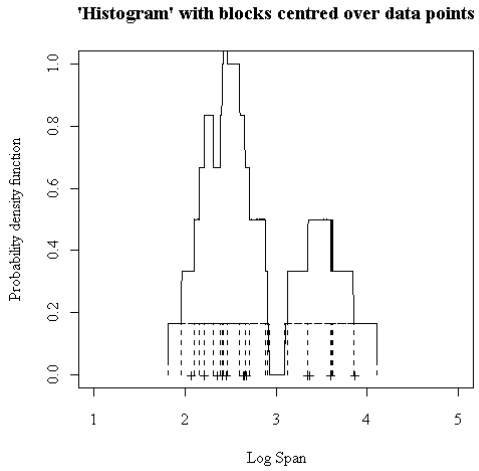
discontinuous "categorical" representation of a continuous distribution of values. So, it gives something up.

- **Histograms depend on the boundary points of the bins.** You saw this in the wingspan example immediately above.
- **Histograms depend on the width of the bins.** The level of detail (be it desired or not) revealed in a histogram depends on how widely spaced the bin boundaries are.

So histograms are subjective beasts, prone to subjective influence and possible misinterpretation of the actual shape of a population distribution which your sample is attempting to represent.

23.3 Kernel density estimation (KDE)

We can alleviate the first two problems by using *kernel density estimators*. To remove the dependence on the end points of the bins, we center each of the blocks at each data point rather than fixing the end points of the blocks. Here is an illustration of the difference using the wingspan sample:



In the "histogram" at left, we placed a block of width 1 and height 1/12 (the dotted boxes), as there are 12 data points, and then we "add them up". This density estimate (the solid line) is less blocky than either of the histograms we saw earlier, as we are now starting to extract some of the finer structure in the data. It suggests that the density is bimodal.

This is known as **box kernel density estimate**. It is still discontinuous as we have used a discontinuous kernel as our building block.

Question: What is the total area under this kernel density estimate?

Smooth kernels

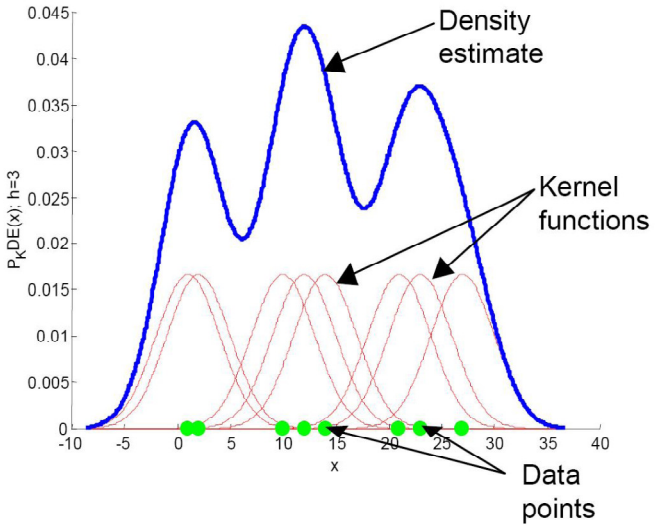
If we instead use some sort of "smooth" kernel for our building block, then we will have a smooth kernel density estimate. Thus we can eliminate the first problem with histograms as well.

A smooth kernel function is some function $K(x)$ that satisfies the mathematical property

$$\int K(x)\,dx = 1$$

If you don't know calculus, all that means is that the area under a kernel is 1. (I'm trying to keep the complicated math to a minimum here, I promise.).

Usually, $K(x)$ is chosen to be a symmetric, unimodal probability density function. The most popular choice is to actually use the **normal distribution** as a kernel. If you use this, the kernel density estimation process can be pictured as follows:

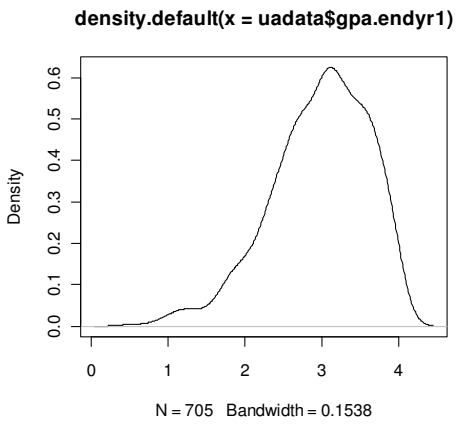


Thanks to Ricardo Gutierrez-Osuna of Wright State University for this picture.

The little normal kernels are "added up" to create the density estimate, just like with the box kernel density estimate. But see how the use of a smooth kernel creates a smoothed representation of the sample distribution? This better reflects the fact that the actual population distribution is continuous.

► **Example: University admissions.** Here is how to have R create a plot of a kernel density estimate of the GPA data from the `uadata` data frame. Compare this to the histogram on the first page of this lecture.

```
> plot(density(uadata$gpa.endyr1))
```



◄

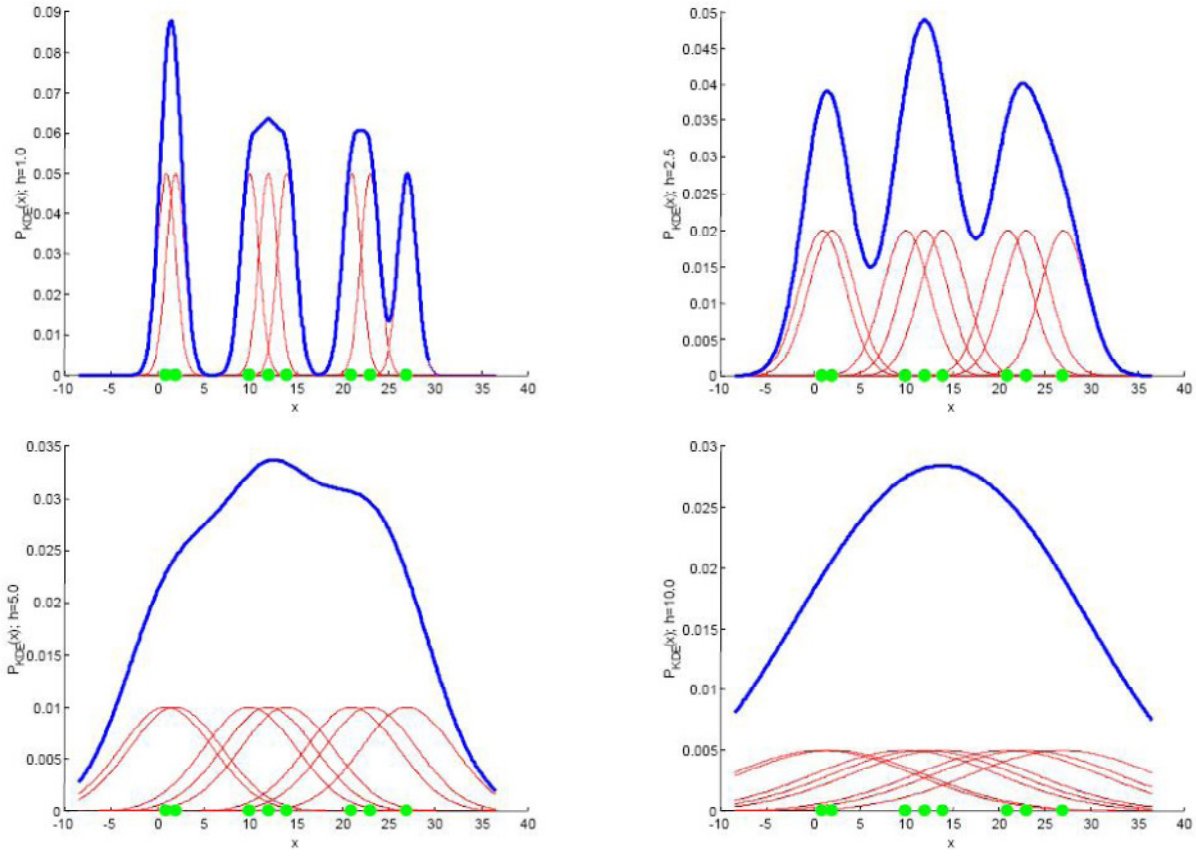
23.4 Choice of bandwidth (AKA: "it's always something")

So we've knocked out two of those three problems traditional histograms have. But what about that third one, the "bin width" issue?

This issue of bandwidth in kernel density estimation is akin to choosing the binwidth in a traditional histogram. Actually, there is *much* potential subjectivity left in KDE problems based on bandwidth choice. Choice of a kernel function $K(x)$ actually has very little affect on the final result (that's why normals are often used as kernels) … **but how wide a normal should you use as the kernel?** That's a crucial question, because:

- **Using too wide a bandwidth** will *oversmooth* the KDE, resulting in obscuring essential features of the distribution of values.
- **Using too narrow a bandwidth** will *undersmooth* the KDE, resulting in the capture of too much detail that overemphasizes negligible or random artifacts in the distribution.

This quartet of plots sums up the problem beautifully. These are four different kernel density estimates of the same data, all using normal kernels, but only with different bandwidths:



The KDE in the upper right corner appears to be the best of the bunch here.

So how does one choose an "optimal" bandwidth? As with most things, it depends on how you define "optimal". Clearly, the best bandwidth to choose is the one that produces a KDE that is as close to the true distributional shape as is possible. However, that requires you to know the true distribution in the first place … so this could be a problem!

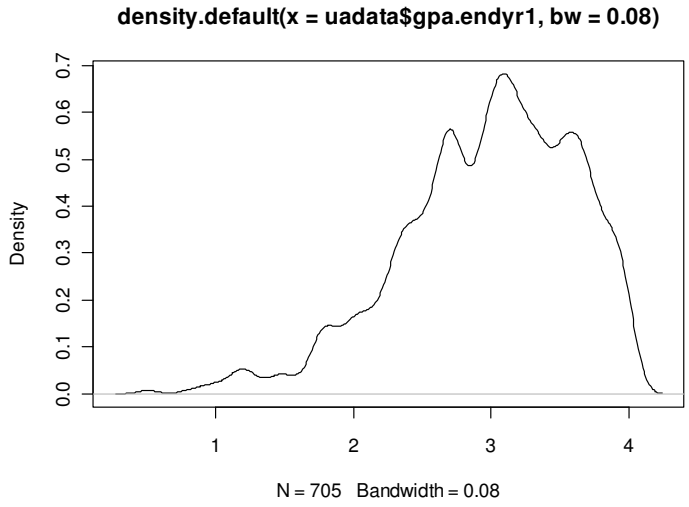
However, there are empirical rules of thumb that work well. One of them in wide use is to choose as the bandwidth the value

$$\frac{1.06s}{\sqrt[5]{n}}$$

where s is the sample standard deviation of the data values. It is sometimes suggested that rather than using s, one should use the smaller of s and $IQR/1.34$. Doing so is better if there are outliers or really strange stuff happening in your data.

► **Example: University admissions.** The default density plot in R (see earlier example) used a bandwidth of 0.1538. You can change this by using the `bw=` option:

```
> plot(density(uadata$gpa.endyr1, bw=0.08))
```



This bandwidth (arbitrarily chosen by me, nothing scientific) obviously "undersmooths" the data a bit. ◄

Lecture 23 Practice Exercises

A random sample of 300 customer weekday noon-hour waiting times at a downtown Columbus Chipotle restaurant were recorded (in minutes). The data are in the R data frame `waittime.Rdata`.

1. Use the data and the *frequentist interpretation of probability* to estimate $P(X > 1)$, i.e. estimate the probability that a randomly selected weekday noon-hour customer will wait longer than one minute to be served.

2. Find a kernel density estimate for the probability distribution of weekday noon-hour waiting times at this downtown Columbus Chipotle restaurant. Use the default bandwidth.

3. Repeat problem 1, but now the KDE from problem 2 to estimate $P(X > 1)$.

STA333 Lecture 24
LOESS nonparametric regression

24.1 Generalizing the simple regression idea

Simple linear regression is a widely used tool for modeling the relationship between a predictor variable X and a response variable Y. It can be used to

1. Assess the significance and strength of the linear relationship between X and Y
2. Predict future values of the mean response of Y at a given hypothetical setting of X.

To get started (and as a bit of review), here is a simple example of simple linear regression in R.

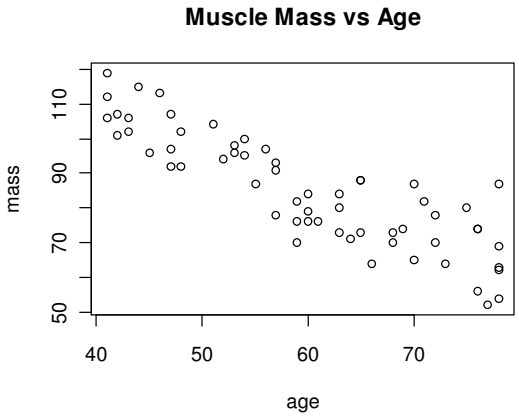
► **Example: Muscle mass.** A person's muscle mass is expected to decrease with age. To explore this relationship in women, a nutritionist randomly selected 15 women from each 10-year age group beginning with age 40 and ending with age 79. The data reside in the R workspace `musclemass`. The variables in the dataset are `mass` and `age`. Let's do the following:

1. Investigate the relationship visually using a scatterplot.
2. Fit a regression model relating muscle mass to age.
3. Predict the mean muscle mass for 65 year old women, and then find a 95% confidence interval for the prediction.

Solution. First, here's the scatterplot:

```
> plot(mass~age, data=musclemass, main="Muscle Mass vs Age")
```

The plot is at right. (We saw this plot in the last lecture.) We can see that as age increases, muscle mass tends to decrease. You should also notice that it decreases in a roughly linear fashion, so a simple linear regression seems viable here.



We now fit the model in R. To do so, the `lm()` function may be used, followed by a `summary()` function call to see results of the regression fit:

```
> fit <- lm(mass~age, data=musclemass)
> summary(fit)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 156.3466     5.5123   28.36   <2e-16 ***
age          -1.1900     0.0902  -13.19   <2e-16 ***
```

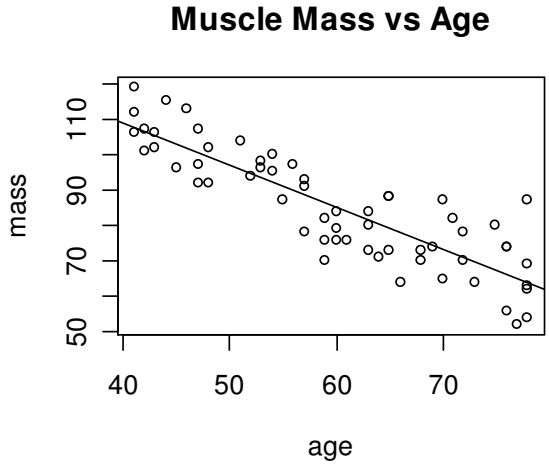
The fitted regression model is $\widehat{mass} = 156.345 - 1.19(age)$. The slope estimate is $b_1 = -1.19$. This means that we estimate a drop of 1.19 in mean muscle mass per year of age in women aged 40-79. We can find the 95% CI for mean muscle mass in 65 year old women using the following command on the fitted model object fit:

```
> predict(fit, newdata=data.frame(age=65), int="conf")
          fit      lwr    upr
[1,] 78.99686 76.69872 81.295
```

We can be 95% confident that the true mean muscle mass for women aged 65 is between 76.7 and 81.3. ◀

The linear regression line may also be overlaid on the plot by issuing the following after fitting the model:

```
> abline(fit)
```



Simple linear regression (or, if you have several predictor variables, multiple regression) is an incredibly useful tool for data analysis: it provides a compact, simple representation of the relationship that enables us to extract useful information about how *X* and *Y* associate. So what's the problem? Everything seems fine here.

Oh wait … remember those regression assumptions? In case you forgot (or never knew about them), here they are:

Error assumptions

We assume that the errors have three properties: (1) they are normally distributed, (2) they have constant variance, and (3) they are independent. Notationally, we write these as $\varepsilon_i \sim$ iid $N(0, \sigma^2)$.

Linearity assumption

We assume that the structural part of the linear regression model is correctly specified.

Unusual observations

Occasionally, a few observations do not fit the model. These isolated observations have the potential to dramatically alter the fit, or even the choice of model used.

Seems like a lot of requirements. Any good statistician would check these to make sure they are reasonable before trusting the results of a standard regression analysis. While there are techniques one can use when assumptions are violated (e.g. data transformations), occasionally you may find that things are "too far gone" to even consider using simple linear regression. In other words, simple linear regression may be "too simple" for a given data set.

The main issues we deal with in this lecture are:

- *Violation of the linearity assumption.* When you fit a simple linear regression, you impose a linear structure onto the relationship that forms the basis of your *X/Y* association. What if the association is not linear?

- *Violation of normality and/or constant variance assumptions.* If the errors (residuals) are not normally distributed or we see a pattern in the residuals suggesting non-constant variance, then the CI we found on the previous page may be inaccurate, or just plain useless.

Enter a nonparametric approach to simple linear regression. Sometimes this is referred to as **scatterplot smoothing**. The particular method we will look at is known as LOWESS, which is short for **LOcally WEighted Scatterplot Smoothing**.

Sometimes is it also referred to as LOESS (without the "W"). Don't ask me why. R has two routines (called `lowess` and `loess`) that both fit such models. `loess` is newer and is an improved version of the older function `lowess`, so we will use `loess`.

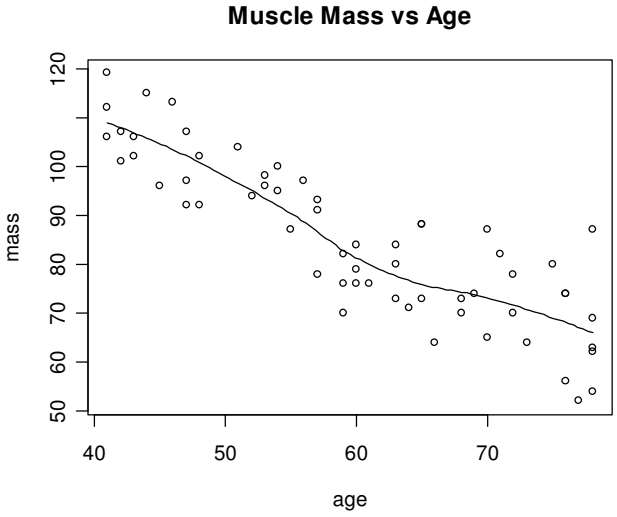
24.2 LOESS basics

LOESS was first developed in the late 1970s and is one of many modern modeling methods that build on traditional linear regression. Modern regression methods are designed to address situations in which the traditional parametric procedures do not perform well or cannot be effectively applied without herculean effort.

LOESS combines much of the simplicity of linear regression with the flexibility of nonlinear regression. **It does this by fitting simple models to localized subsets of the data to build up a function that describes the relationship in the data point-by-point.** In fact, one of the chief attractions of this method is that you are not required to specify a global function of any form to fit a model to the data, only to fit segments of the data.

For example, here is a quick-and-dirty LOESS curve fit to the `musclemass` data, using the R function `loess`. The plot appears below.

```
> plot(mass~age, data=musclemass, main="Muscle Mass vs Age")
> out <- loess(mass~age, data=musclemass)
> curve(predict(out, newdata=data.frame(age = x)), add=TRUE)
```



Because it is so computationally intensive, LOESS would have been practically impossible to use until fairly recently. These methods have been consciously designed to use current computational capabilities to their fullest potential to achieve modeling goals not easily achieved by traditional approaches.

LOESS specifically denotes a method that is more descriptively known as **locally weighted regression.** At each point in the data set, a low-degree polynomial (either linear or quadratic) is fit to a subset of the data, with predictor variable values (X) near the point whose response is being estimated. The fit is done using a technique known as *weighted least squares*: this method gives more weight to points near the point whose response is being estimated and less weight to points further away. The value of the regression function for the point is then obtained by evaluating the local polynomial using the predictor variable values for that data point. The LOESS fit is complete after regression function values have been computed for each of the n data points.

Many details of this method are flexible. In particular, you can control these three attributes:

- **The degree of the local polynomial model**. Usually this is linear (straight-line) or quadratic (second degree polynomial).

- **A smoothing parameter**. This defines the size of the "neighborhood" of a particular X value. It is called a smoothing parameter because it controls the flexibility of the LOESS regression function. Large values produce the smoothest functions that wiggle the least in response to fluctuations in the data. The smaller the smoothing parameter is, the closer the regression function will conform to the data. *Using too small a value of the smoothing parameter is not desirable, however, since the regression function will eventually start to capture the random error in the data.*

  In the R function `loess`, the smoothing parameter is given by the argument `span`.

- **A weight function** that determines how much each data value in a local neighborhood plays a role in fitting the LOESS curve at a particular point. *The weight function gives the most weight to the data points nearest the point of estimation and the least weight to the data points that are furthest away.* The use of the weights is based on the idea that points near each other in the predictor variable space are more likely to be related to each other in a simple way than points that are further apart. Following this logic, points that are likely to follow the local model best influence the local model parameter estimates the most. Points that are less likely to actually conform to the local model have less influence on the local model parameter estimates.

  For you math geeks out there, the traditional weight function used for LOESS is the tri-cube weight function:

$$w(X) = \begin{cases} (1 - |X|^3)^3 & |X| < 1 \\ 0 & |X| \geq 1 \end{cases}$$

The gory details of the involved computations for a LOESS modeling example will be available via a separate class handout.

LOESS advantages:

- LOESS does not require the specification of a function describing the relationship.
- LOESS is very flexible, making it ideal for modeling complex processes for which no theoretical models exist.
- It can be used to help verify if a simpler model (such as one that imposes a linear relationship on the data) is reasonable.

LOESS disadvantages:

- LOESS requires fairly large, densely sampled data sets in order to produce good models. This is not really surprising, since LOESS needs good empirical information on the local structure of the process in order perform the local fitting.
- LOESS does not produce a regression function that is represented by a mathematical formula. This can make it difficult to communicate the results of an analysis to others.
- LOESS is somewhat prone to the effects of outliers in the data set, like other methods.

24.3 Examples

► **Example: Respiratory rates in children.** A high respiratory rate is a potential diagnostic indicator of respiratory infection in children. To judge whether a respiratory rate is truly "high," however, a physician must have a clear picture of the distribution of normal respiratory rates. To this end, Italian researchers measured the respiratory rates of $n = 618$ children between the ages of 15 days and 3 years (given in months). The data appear in the R workspace `respiratory`. Use the data to do the following:

1. Fit a nonparametric LOESS regression curve to the data, and
2. Use it to calculate a 95% bootstrap CI for the true mean respiratory rate for children aged 12 months.

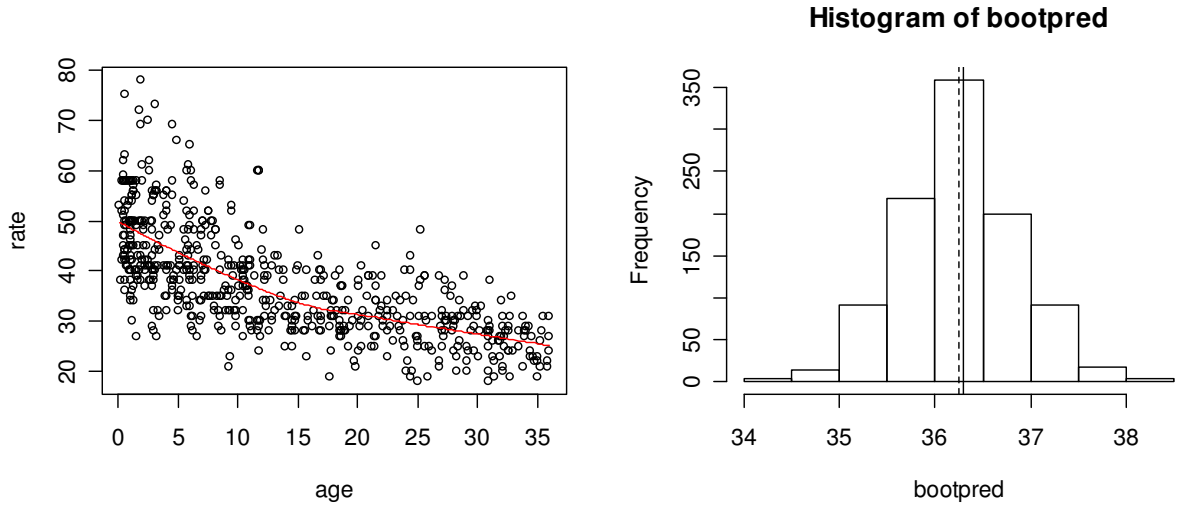
Solution. Here is a program to do the work:

```
par(mfrow=c(1,2))              # set up two panels for our plots

attach(respiratory)
bootpred <- numeric(1000)      # create vector to hold bootstrapped predictions
hyp.age <- 12                  # set the hypothesized x value for prediction


rate.lo <- loess(rate~age,respiratory)
pred <- predict(rate.lo, data.frame(age = hyp.age))
plot(rate~age, data=respiratory)         # make scatterplot of the data points
lines(age,rate.lo$fit, col=2)            # draw in the fitted loess curve

for (i in 1:1000) {
  n <- nrow(respiratory)
  index <- sample(1:n, size=n, replace=TRUE)
  bootdata <- data.frame(rate=respiratory$rate[index],age=respiratory$age[index])
  bootfit <- loess(rate~age, bootdata)
  bootpred[i] <- predict(bootfit, data.frame(age = hyp.age))    # extract predicted
                                                                # rate when age=12
   }

hist(bootpred)                  # picture of the bootstrap dist of predictions
abline(v=pred)                  # add vertical line at the actual sample prediction
abline(v=mean(bootpred),lty=2   # add dashed line at mean of the bootstrap dist

quantile(bootpred, c(0.025, 0.975))    # bootstrap CI for prediction
```

We will run this in class and discuss the program and results. For the record, here are the resulting findings:

```
    2.5%    97.5%
35.06006 37.46490
```

We can be 95% confident that the true mean respiratory rate for children aged 12 months is between 35.06 to 37.46. The plots on the next page show the fitted LOESS curve and the bootstrap distribution for the predicted respiratory rate. ◄



► **Example: Respiratory rates in children.** Find the same prediction (i.e. a 95% CI for the true mean respiratory rate in 12 month old children), only now using a conventional simple linear regression. How does the result compare to your bootstrap CI found from a LOESS model?

Solution. Here's the R code and output:

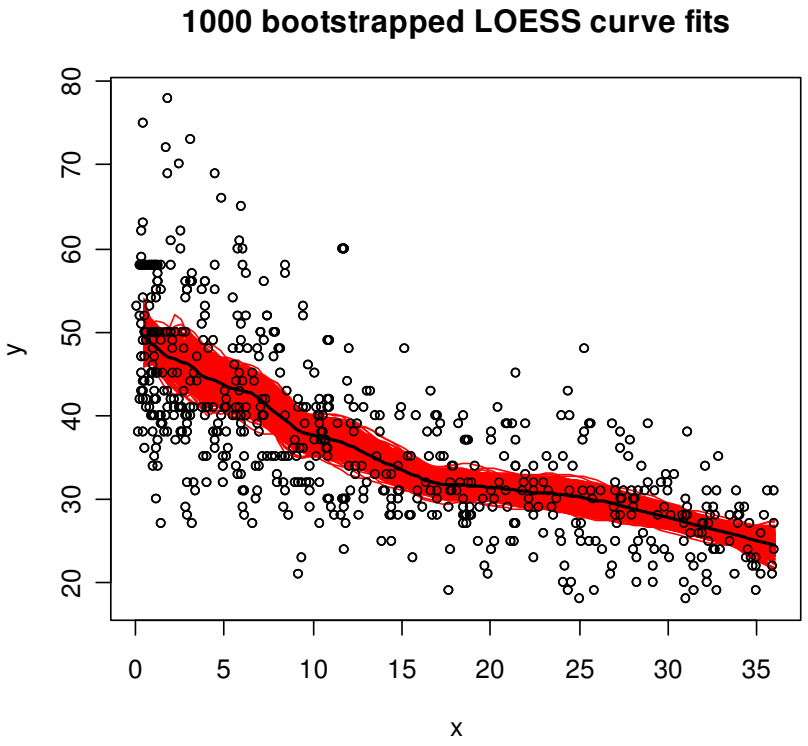
```
> slrfit <- lm(rate~age, data=respiratory)
> predict(slrfit, newdata=data.frame(age=12), int="conf")

         fit      lwr      upr
[1,] 38.7036 38.07893 39.32828
```

Simple linear regression yields a 95% CI for the true mean respiratory rate for 12-month olds of (38.08, 39.32). The LOESS model with bootstrapping yielded (35.06, 37.46). Clearly, the simple linear regression "misses" the shallowing out of the rate of decrease in respiratory rate as age increases. You can imagine the straight line passing *above* the LOESS line in the plot above. That translates into simple linear regression *overestimating* the response at this point. ◄

You can (just for fun and enlightenment) generate a picture of the bootstrapped LOESS curves. Below is a program that illustrates this for the `respiratory` data. I also chose to lower the `span` value of 0.3 just for kicks: this make the fit a bit more "wobbly" in comparison to the default span of 0.75. Feel free to try different values and see the effects on the fitted curve.

```
x <- respiratory$age
y <- respiratory$rate

out <- loess(y~x, span = 0.3)
plot(y~x, main="1000 bootstrapped LOESS curve fits")
curve(predict(out, newdata=data.frame(x = x)), add=TRUE)

n <- length(x)
nboot <- 1000
for (i in 1:nboot) {
    index <- sample(n, replace = TRUE)
    boot.x <- x[index]
    boot.y <- y[index]
    boot.out <- loess(boot.y~boot.x, span=0.3)
    curve(predict(boot.out, data.frame(boot.x = x)), add=TRUE, col="red")
}

points(x, y)
curve(predict(out, newdata=data.frame(x = x)), add=TRUE, lwd=2)
```

Here's the resulting plot:



Lecture 24 Practice Exercises

Respiratory rates in children. A high respiratory rate is a potential diagnostic indicator of respiratory infection in children. To judge whether a respiratory rate is truly "high," however, a physician must have a clear picture of the distribution of normal respiratory rates. To this end, Italian researchers measured the respiratory rates of $n = 618$ children between the ages of 15 days and 3 years (given in months). The data appear in the R workspace `respiratory.RData`.

Use the data to do the following:

1. Find 95% bootstrap CIs for the true mean respiratory rate for children aged 6 months, 24 months, and 36 months.

2. Redraw the LOESS curve using different values of the smoothing parameter to see the effect on the fitted curve. Use the R function `loess` and specify the span parameter using `span=` option. (*Note:* `span=0.75` is the default.)

STA333 Lecture 25
Regression and classification trees (part 1)

25.1 Introduction

When we use regression analysis, one of the main goals is to determine how a response variable Y is related to (or determined by) a predictor variable X. In practice, however, we are not limited to just looking at one predictor variable: the concept of using several predictors (X_1, X_2, …, X_k) to determine the value of a response variable Y is very widely used in data analysis.

For example, consider these situations:

- Your risk of having heart disease may be related to several potential factors simultaneously: your weight, if you have a family history of heart disease, whether or not you have diabetes, how much you exercise, your blood pressure, etc.
- The gas mileage of a given car may be impacted by several factors: the weight of the car, how many cylinders the engine has, how aerodynamic the body style is, etc.
- Whether or not an email filter identifies a piece of incoming mail as spam may be related to several potential predictors: e.g. the total length of words that are all in capital letters, the frequency of the word "money" in the message, the occurrence of the characters "XXX" in the message, etc.

The standard data analysis tool used in practice for such problems is **multiple regression**. In multiple regression, we attempt to relate the response variable Y to the predictor variables using some specified mathematical function, such as

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k + \varepsilon$$

The β parameters are estimated and used to generate predictions of *Y* and to determine which variables have the most influence on *Y*. Obviously, this is a parametric method since inferences about *Y* depend on the estimation of β parameters and assumptions about the regression errors.

Another way of describing how potential predictor variables influence a response variable *Y* is to think about where we should "draw the line" with regard to a predictor variable's value in order to maximally distinguish the value of *Y*.

For example, suppose that "drawing the line" at a systolic blood pressure level of 144 does a better job of distinguishing between people with or without heart disease than we would get by drawing the line at any other systolic blood pressure level. In essence, doing this **partitions the sample** into two groups:

- Those with a systolic blood pressure ≤ 144.
- Those with a systolic blood pressure > 144.

Within each group, there is **maximal similarity** in terms of heart disease prevalence, but *between* groups there is **maximal dissimilarity**. In other words, we choose the splitting point for the predictor *X* so as to make the values of the response variable *Y* as homogeneous as possible within each group, and as heterogeneous as possible between groups.

There is no reason to just split once. We could split on a predictor variable in several places if that's what maximally determines *Y*. For example, we could end up with more than two groups based on blood pressure levels. *More generally though, we could perform the splitting simultaneous on several different predictor variables.* This process is known as **recursive partitioning**, and it is a major statistical tool used in **data mining**.

25.2 What is data mining?

Perhaps at this point, a brief introduction to "data mining" is in order.

Data mining is the process of analyzing data from different perspectives and summarizing it into useful information that can be used to increase revenue, cuts costs, or both. Technically speaking, *data mining is the process of finding patterns or structure among dozens of variables in large relational databases.*

► **Example: Your Kroger Plus card.** The Midwest grocery chain Kroger uses data mining to analyze local buying patterns. They collect data every time you swipe your Kroger Plus card. How does it work?

Well, suppose through data mining customer purchasing patterns, they discover that when men buy diapers on Thursdays and Saturdays, they also tend to buy beer. Suppose a further analysis shows that these shoppers typically did their weekly grocery shopping on Saturdays. On Thursdays, however, they only bought a few items. Kroger then determines that they purchased the beer to have it available for the upcoming weekend. The grocery chain could then use this information in various ways to increase revenue. For example, they could move the beer display closer to the diaper display. Or, they could make sure beer and diapers are sold at full price on Thursdays. You get the picture. ◄

Data mining consists of five steps:

- Extracting and loading data (e.g. purchase transactions) into a data "warehouse".
- Storing and managing the data in a multidimensional database system.
- Providing data access to business analysts and IT professionals.
- Analyzing the data using software.
- Presenting the analysis in a useful format, such as a graph or table.

Many types of analysis are possible in data mining, but the one we'll talk about (briefly!) is known as a **decision tree**. Tree-shaped structures represent sets of decisions, and these decisions generate rules for the classification of a dataset. A nonparametric statistical method of developing decision trees is known as **classification and regression trees** (or CART). These provide a set of rules that you can apply to a new (unclassified) dataset to predict which records will have a given outcome. The "rules" are developed by finding where to optimally "split" the data, and that's where the process of recursive partitioning comes into play.

25.3 Recursive partitioning 101

The idea behind recursive partitioning is to repeatedly select the *most relevant* predictor variable and use it to stratify the data. Within each stratum, we select the most relevant predictor on that data and sub-stratify until the strata are so small that we run out of data. The result of all this is a **decision tree**. If the response variable Y is a classification variable (e.g. do you have heart disease or not?), the decision tree is also known as a **classification tree**. If Y is a continuous numeric variable (e.g. what is the gas mileage of a given automobile?), the tree is known as a **regression tree**.

By design, the responses within each stratum are as similar as possible. Recursive partitioning is not particularly good at providing a model for the data or making detailed predictions – rather, *the usefulness of this procedure for data mining is that it finds the main subdivisions of the data as well as the most relevant predictors.*

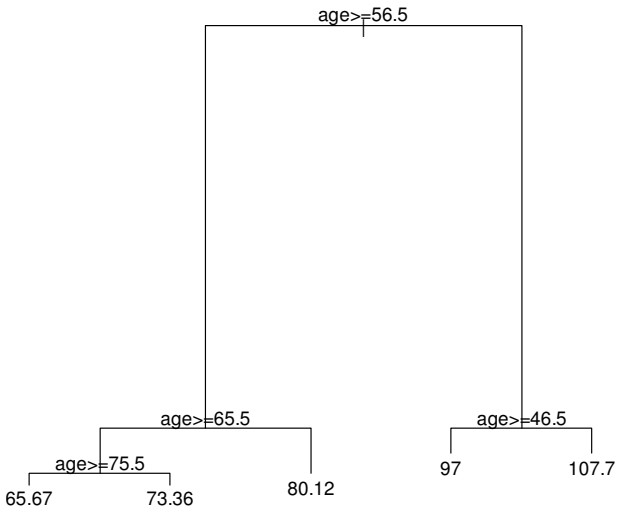
To get started, let's revisit (one last time) the `musclemass` data. This is a small dataset with only one predictor, but we can use it as a basic example to illustrate what happens.

► **Example: Muscle mass.** To explore the relationship between muscle mass and age in women, a nutritionist randomly selected 15 women from each 10-year age group beginning with age 40 and ending with age 79. The variables in the dataset are `mass` and `age`. Let's build a regression tree for determining `mass` from `age`.

Solution. We can perform recursive partitioning in R using the add-on package `rpart`. Once installed and loaded, we run the following to get a regression tree:

```
> library(rpart)
> tree1 <- rpart(mass~age, data=musclemass)
> plot(tree1)
> text(tree1)
```

The resulting regression tree for the single predictor `age` is given below:



Reading the tree is done as follows: If the condition that is specified at a node (i.e. a decision point) is satisfied, then we take the branch to the left. Thus, if `age` is greater than or equal to 56.5, we branch left. Then comes another decision: is `age` greater than or equal to 65.5? If not, we take the branch to the right. The combination of these two branches means that a person is in the [56.5, 65.5) age group. For this relatively homogenous age group, the predicted muscle mass is $\overline{Y} = 80.12$.

Another way of viewing the results can be accommodated as follows:

```
> print(tree1)

n= 60

node), split, n, deviance, yval
      * denotes terminal node

1) root 60 15501.9300  84.96667
  2) age>=56.5 37  3674.9190  74.59459
    4) age>=65.5 20  1929.8000  69.90000
      8) age>=75.5 9  1042.0000  65.66667 *
      9) age< 75.5 11   594.5455  73.36364 *
    5) age< 65.5 17   785.7647  80.11765 *
  3) age< 56.5 23  1443.2170 101.65220
    6) age>=46.5 13   348.0000  97.00000 *
    7) age< 46.5 10   448.1000 107.70000 *
```

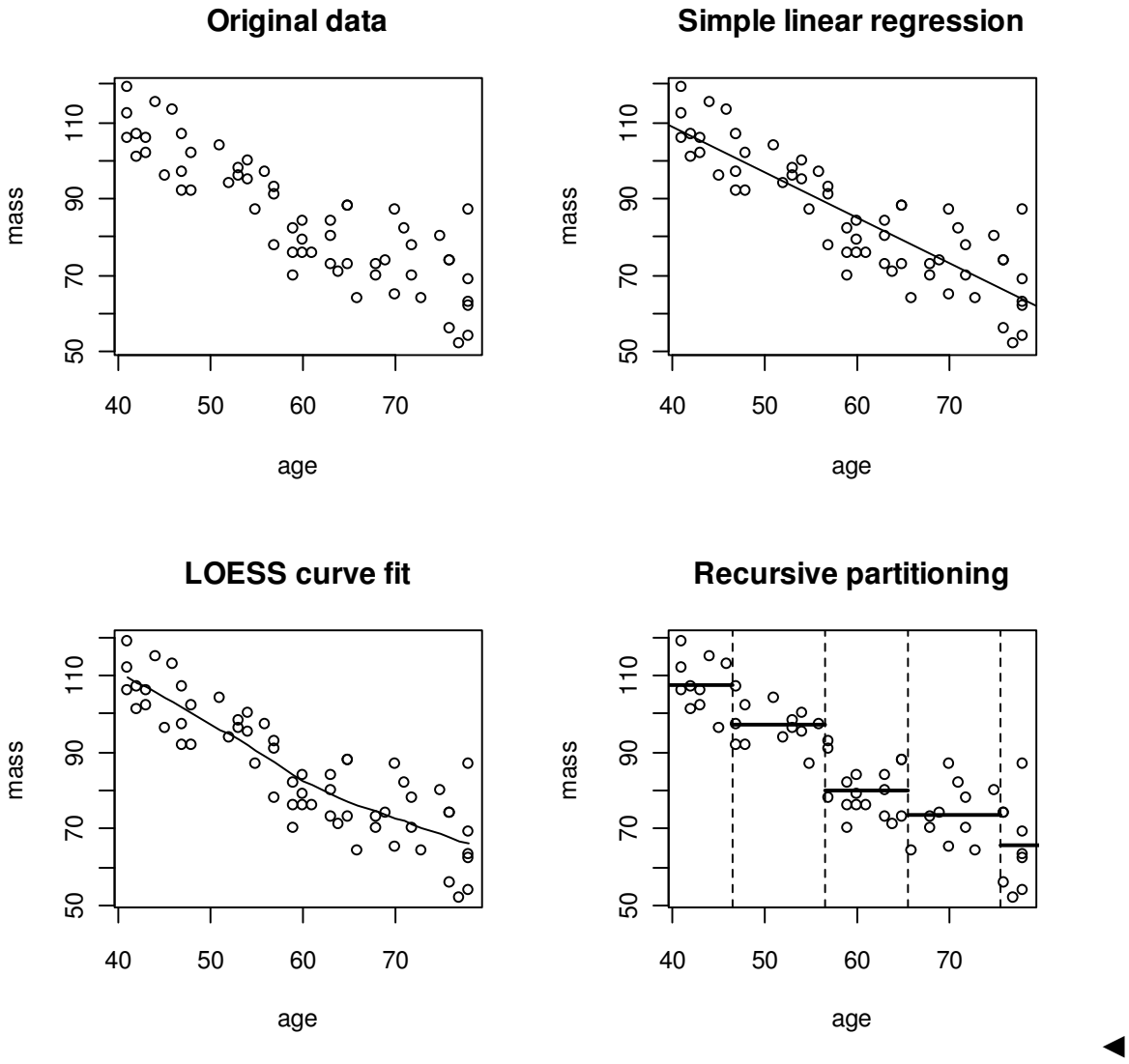
This tree fitting produced five partitions of `age`, as indicated above by the terminal nodes:

- Age less than 46.5 — predicted mean muscle mass = 107.7
- Age in the interval [46.5, 56.5) — predicted mean muscle mass = 97.0
- Age in the interval [56.5, 65.5) — predicted mean muscle mass = 80.12
- Age in the interval [65.5, 75.5) — predicted mean muscle mass = 73.36
- Age greater than or equal to 75.5 — predicted mean muscle mass = 65.66

This is certainly another way to think about predicting the response. So far, we have seen two different ways of "modeling" the response variable mass using information about a woman's age:

1. Simple linear regression (imposes several assumptions about error distributions and structure of the relationship)
2. LOESS model fitting (a nonparametric way to describe the trend evident in the relationship between a response and predictor(s))

Just for illustration, the plots on the next page show the resulting models from these two methods, alongside the results from fitting a regression tree using recursive partitioning:



◄

The previous example was a simple regression tree, that is, a decision tree for a continuous numerically-measured outcome. We can also build decision trees when the outcome is qualitative or categorical. In that case, the tree is known as a classification tree.

Here is another simple example with only one predictor variable, only now the response variable is categorical. Our goal is to predict a classification based on the values of the predictor variable.

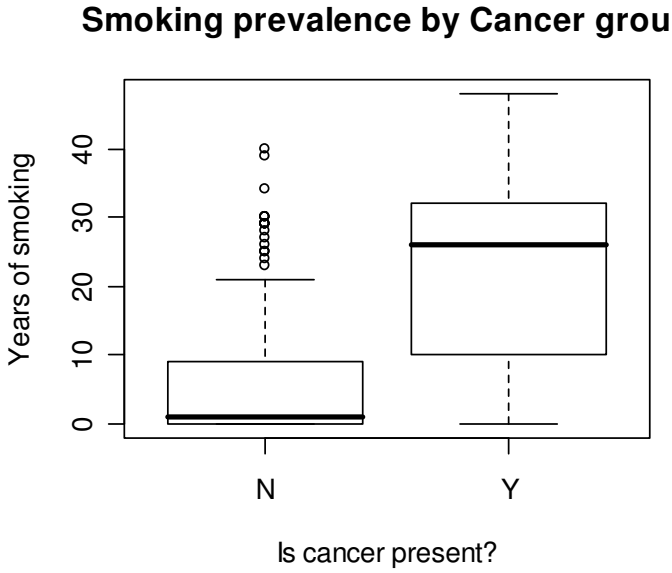
► **Example: Cancer prevalence.** A random sample of patient records of 200 men aged 60 was collected to investigate the association between number of years smoking and presence of lung cancer. The data appear in the R data frame `cancerprev` in our repository.

`yrssmoke`	Number of years the subject has been a smoker
`cancer`	Variable indicating whether the subject has cancer (`Y` or `N`)

Build a classification tree for predicting presence of cancer in 60-year old men.

Solution. First, it might be instructive to look at the data. Here are boxplots to compare the sample groups visually:

```
> boxplot(yrssmoke~cancer, data=cancerprev,
                           main="Smoking prevalence by Cancer group",
                           xlab="Is cancer present?",
                           ylab="Years of smoking")
```

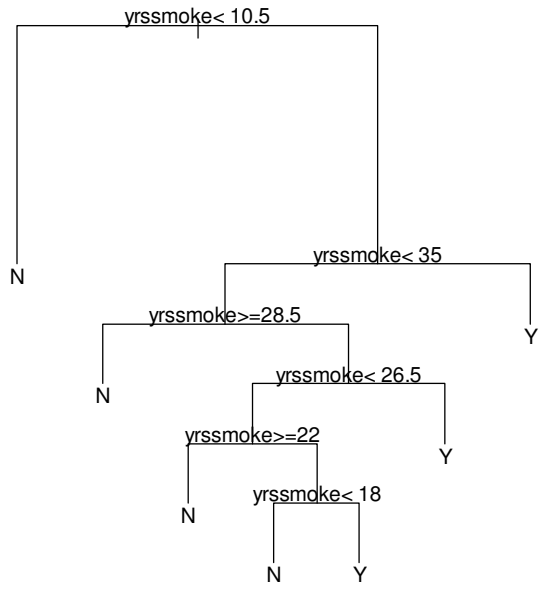


We will discuss this plot in class. Now, here is the fitting of a classification tree using `rpart`:

```
> library(rpart)
> tree2 <- rpart(cancer~yrssmoke, data=cancerprev, method="class")
> print(tree2)

n= 200

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root                     200 51 N (0.74500000 0.25500000)
    2) yrssmoke< 10.5         132 13 N (0.90151515 0.09848485) *
    3) yrssmoke>=10.5          68 30 Y (0.44117647 0.55882353)
      6) yrssmoke< 35          54 26 N (0.51851852 0.48148148)
       12) yrssmoke>=28.5      18  7 N (0.61111111 0.38888889) *
       13) yrssmoke< 28.5      36 17 Y (0.47222222 0.52777778)
         26) yrssmoke< 26.5    28 13 N (0.53571429 0.46428571)
           52) yrssmoke>=22     8  2 N (0.75000000 0.25000000) *
           53) yrssmoke< 22    20  9 Y (0.45000000 0.55000000)
            106) yrssmoke< 18  10  4 N (0.60000000 0.40000000) *
            107) yrssmoke>=18  10  3 Y (0.30000000 0.70000000) *
         27) yrssmoke>=26.5     8  2 Y (0.25000000 0.75000000) *
      7) yrssmoke>=35           14  2 Y (0.14285714 0.85714286) *
```

The tree has seven terminal nodes, which we can picture this way:

```
> plot(tree2)
> text(tree2)
```



Here are some observations:

- At the starting node (the **root node**), there are n = 200 men in the group (i.e. the whole sample). 74.5% of them do not have cancer (`cancer=N`) and the remaining 25.5% do have cancer (`cancer=Y`).
- Suppose a man has smoked for less than 10.5 years. Then they would take the first branch to the left. There are 132 such men. Of them, 90.15% do not have cancer and the remaining 9.85% do. **The predicted classification for this group is "N"**: that is, we predict that men who have smoked less than 10.5 years will not have cancer by age 60. This is our prediction because the majority of men in this group do not have it (90.15% vs. 9.85%). You will note, however, that the tree model isn't perfect: in this segment alone, we have misclassified $132 \times 0.0985 = 13$ men. *News flash:* no model is perfect!
- See if you can make similar determinations for other branches of the tree.

In any modeling exercise (linear regression, LOESS, trees, etc), we want a model that explains the essential features and structure in the data well, **but we also want a model that is not overly complicated**. The splits as chosen in the above example do a good job of distinguishing cancer presence from cancer absence, but there may be more splits than are really necessary to make effective predictions. So, just like regression, we may want to eliminate "unnecessary" splits.

In effect, we may eventually want to **prune the tree**! We'll see this later, but in the next lecture, I will extend tree building to situations when you have many predictor variables.

Lecture 25 Practice Exercises

1. **Respiratory rates for children**. A high respiratory rate is a potential diagnostic indicator of respiratory infection in children. To judge whether a respiratory rate is truly "high," however, a physician must have a clear picture of the distribution of normal respiratory rates. To this end, Italian researchers measured the respiratory rates of 618 children between the ages of 15 days and 3 years. The data appear in the R workspace `respiratory.RData`. Fit and interpret a basic regression tree for these data.

2. **Bird habitation.** A field study in ornithology is conducted to determine how two different characteristics influence the likelihood of habitation of a particular species of bird on a given island. In this example, the response variable is called `occupied`: a value of 1 means that a given island was occupied by the species in question, and a 0 means it was not. The two predictor variables are the area of the island (`area`, in km^2) and the isolation of the island (`distmain`, distance from the mainland, in km). The data appear in the R workspace `isolation.RData`. Fit and interpret a basic classification tree for these data.

STA333 Lecture 26
Regression and classification trees (part 2)

26.1 A classification tree example using more than one predictor

In the last lecture, we saw two examples of tree models using only one predictor. In more useful contexts such as data mining, recursive partitioning and tree models are valuable tools for investigating the effects of many predictors simultaneously. In particular, we are interested in the following tasks:

1. **Optimality.** We may be interested in finding a combination of predictor values that produces an optimal outcome, e.g. maximizes sales.

2. **Identification of important predictors.** The exploration of a new data set with tree-based regression or classification can help us gain a quick handle on which variables have major effects on the outcome. In addition, tree-based methods can be used in tandem with their traditional counterparts (multiple or stepwise regression) in ways that combine the strengths of both approaches.

3. **Identification of structure.** Traditional regression relies of the specification of the form of the response/predictor relationship. Tree-based methods have no such pre-specified structure, so the process of recursive partitioning may reveal interesting structure in how the predictors affect the response (e.g. interactions) that may be difficult for traditional approaches to detect.

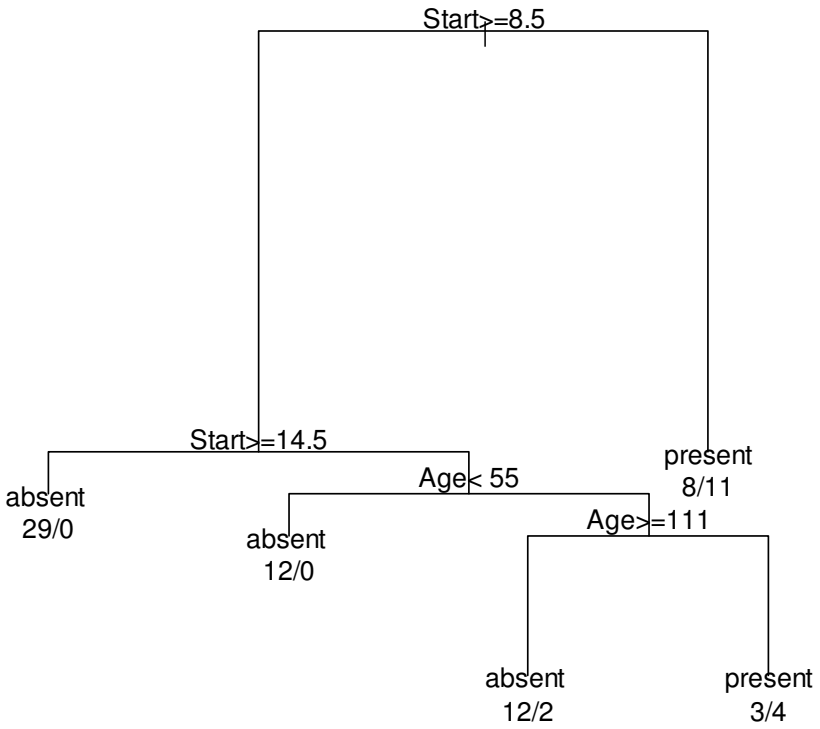
► **Example: Kyphosis in children.** The R dataframe `kyphosis` resides within the `rpart` add-on package as a demonstrator data set. It consists of records of n = 81 children who had corrective spinal surgery. The dataset contains the following variables:

`Kyphosis`	indicator of the presence or absence of kyphosis (a type of deformation) following spinal surgery
`Age`	patient age in months
`Number`	the number of vertebrae involved
`Start`	the number of the first (topmost) vertebrae operated on

The researcher would like to study how these factors are related to the presence or absence of kyphosis following surgery, and which factor is most important. (*Question*: what sort of valuable information might this study produce?)

Solution. We fit a default classification tree to predict `Kyphosis` from `Age`, `Number` and `Start`. The details of the following code will be discussed in class. Here is the program:

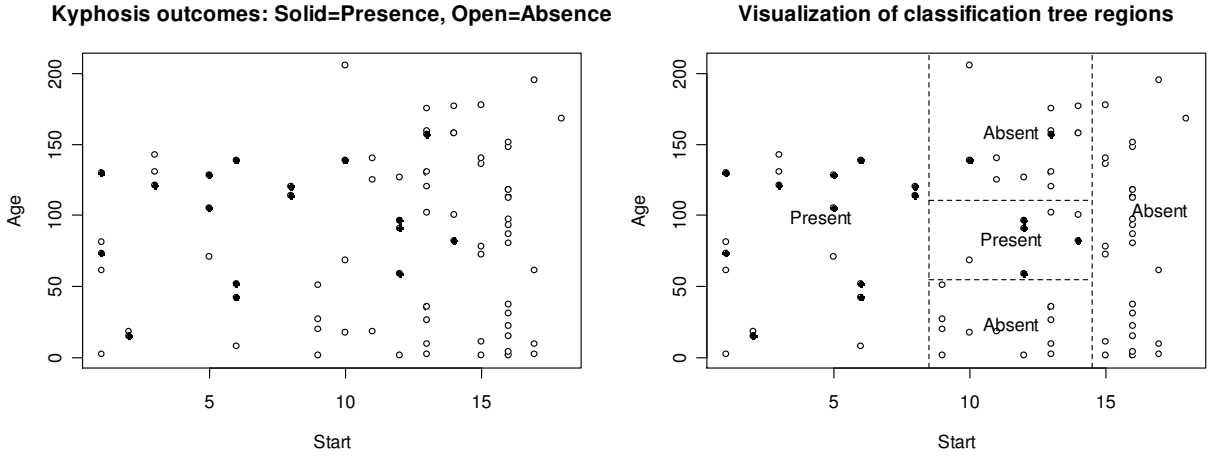
```
> library(rpart)
> kfit <- rpart(Kyphosis~.,data=kyphosis,method="class")
> plot(kfit,compress=TRUE,margin=0.1)
> text(kfit, use.n=TRUE)
```



You'll note that the number of vertebrae involved (`Number`) is not a predictor of kyphosis. The first split (node) in the tree deals with the number of the topmost vertebrae involved, so **this is the single most important distinguisher** of the presence or absence of the deformation. It is fairly clear at this point by looking at the tree that kyphosis is more likely when the topmost vertebrae involved is at or above the eighth vertebrae. In this case, 11 out of 8 + 11 = 19 children (58%) had kyphosis.

The more interesting result comes from looking at what happens when the topmost vertebrae involved is at or below the ninth vertebrae: it is at this point that the tree reveals an **interaction** between patient age and the topmost involved vertebrae. *Two variables are said to interact to determine an outcome if the effect that one variable has on the outcome depends on what the other variable is.* In this case, generally a patient will be predicted to not acquire kyphosis, unless their topmost involved vertebrae is between the 9th and 14th and the patient's age is between 55 and 110 months.

This last result is a bit curious, and it may help to visualize the predictor space another way. This is possible here since there are really only two predictors at work, and thus we may picture what is happening in two dimensions:



Here are the classification tree results as a table. See if you can fully interpret the information contained in this display:

```
> print(kfit)

n= 81

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 81 17 absent (0.7901235 0.2098765)
   2) Start>=8.5 62  6 absent (0.9032258 0.0967742)
     4) Start>=14.5 29  0 absent (1.0000000 0.0000000) *
     5) Start< 14.5 33  6 absent (0.8181818 0.1818182)
      10) Age< 55 12  0 absent (1.0000000 0.0000000) *
      11) Age>=55 21  6 absent (0.7142857 0.2857143)
        22) Age>=111 14  2 absent (0.8571429 0.1428571) *
        23) Age< 111 7  3 present (0.4285714 0.5714286) *
   3) Start< 8.5 19  8 present (0.4210526 0.5789474) *
```

We will discuss this in class. ◄

26.2 Pruning the tree: predictive accuracy and cost-complexity tradeoff

In fitting a tree model, it is true that each local split is optimal in terms of making the within-split responses as homogenous as possible and the across-split responses as heterogeneous as possible. *However, doing so can result in an overall tree that is far from optimal.* Many decisions are made along the way to growing a tree: what variable to split on, how to split it, etc. Each of these choices allows the tree to actually overfit the data. We got a sense of this from the cancer prevalence example in Lecture 24: the number of splits we ended up with seemed like it could be overkill. Well, if the tree has more branches than necessary, why not "prune" the tree?

We need a valid way of assessing the performance of a tree. Once we have grown a tree from a dataset, how do we know that the tree represents properties of the entire population and not just the sample? In general, how much should we trust the results of data mining? One way to quantify whether a tree represents the right things is by its **predictive ability**, e.g. error rate, on future data. If it can predict the class of new data points accurately, then it must capture something important about the population. *How can we estimate the population error rate?*

Cross-validation

Realistic estimates of the predictive power of a tree model must take into account the extent to which the model has been selected from a wide range of candidate models. The *cross-validation error rate* is relevant to predictions for the population from which the data were sampled. It works as follows:

1. Randomly split the entire data set into k equal sized subsets. Let's refer to these subsets as subsets 1, 2, 3, …, k.
2. Leave out subset 1. Fit a tree model using all the combined data from subsets 2, 3, …, k.
3. Leave out subset 2. Fit a tree model using all the combined data from subsets 1, 3, …, k.
4. Repeat this process for all subsets. At the last step, leave out subset k. Fit a tree model using all the combined data from subsets 1, 2, …, $k - 1$.
5. Aggregate some measure of the predictive performance of the k fitted trees. Choose as optimal a tree size that maximizes this predictive performance.

In a regression tree, prediction error is usually taken as the sum of differences between observed and predicted outcomes. In a classification tree, prediction error is usually determined by counting up the numbers of correct and incorrect classifications. The crucial feature of cross-validation is that each prediction is independent of the data to which it is applied. As a result of this, cross-validation gives an unbiased estimate of predictive power of a tree model, albeit for a model that uses $100(k - 1)/k\%$ of the data.

An estimate of the "average" error is found by summing up the measure of error over all observations and dividing by the number of observations. Once predictions are available in this way for each of the subsets, the average error is taken as

total error / total number of observations.

Cross-validation is built into the `rpart` package, giving an assessment of the change in prediction error with changing tree size. **The usual approach is to build a tree that has more splits than is optimal (that is, it is over-fitted), and then to prune back to a tree that has close to the minimum cross-validated prediction error.**

`rpart` uses $k = 10$ subsets by default. It also goes without saying that cross-validation works better on bigger data sets, since you are always analyzing a subset of the original data.

The cost-complexity value: c_p

Rather than controlling the number of splits directly, we control it indirectly through a cost-complexity quantity denoted by c_p that puts a cost on each additional split. c_p appears as `cp` in the output from `rpart`. The increase in cost as the tree becomes more complex is traded off against the reduction in the prediction error criterion. Further splitting should stop when the increase in cost outweighs the improvement in model prediction. **A high value of c_p leads to a smaller tree, while a low value of c_p leads to a more complex tree.** Thus, *the choice of c_p is a surrogate for the number of splits.*

Here is a strategy for optimizing the tree model:

1. Fit a tree that is more complex than is optimal (the default tree may not do this).
2. Plot the cross-validated prediction error vs c_p. This helps us determine the value of c_p for which the tree seems optimal.
3. Having identified the optimal tree size (with minimum cross-validated error rate), we then prune off unnecessary splits.

I'd say it's time for an example!

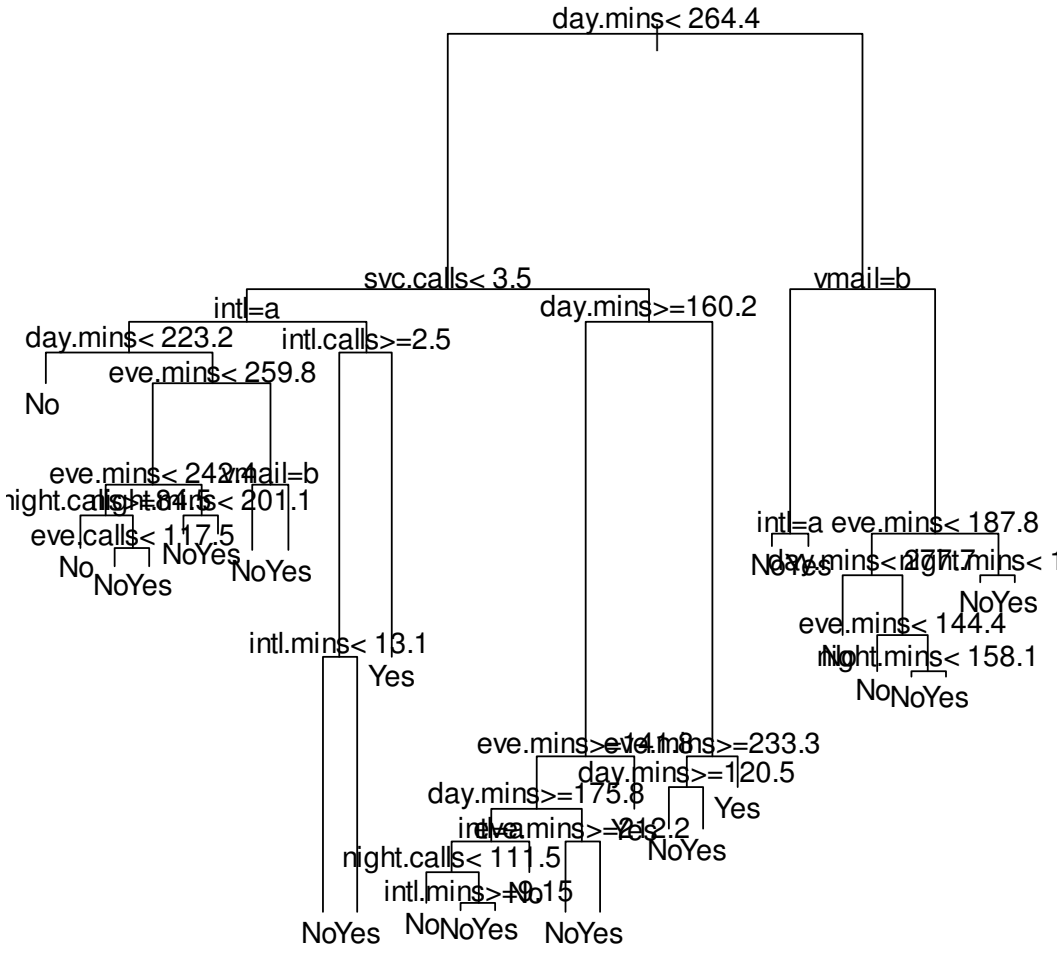
► **Example: Predicting "churn".** For a more complex example, consider the problem of predicting whether a long-distance phone customer will switch to another company (a practice known in the industry as "churning"). The R dataframe `churn` describes 18 monthly characteristics on 3333 customers, including whether they churned or not. Here is the first 6 rows of the data:

```
> head(churn)

  acct.len area intl vmail vmail.msgs day.mins day.calls day.charge eve.mins eve.calls eve.charge
1      128  415   no   yes         25    265.1       110      45.07    197.4        99      16.78
2      107  415   no   yes         26    161.6       123      27.47    195.5       103      16.62
3      137  415   no    no          0    243.4       114      41.38    121.2       110      10.30
4       84  408  yes    no          0    299.4        71      50.90     61.9        88       5.26
5       75  415  yes    no          0    166.7       113      28.34    148.3       122      12.61
6      118  510  yes    no          0    223.4        98      37.98    220.6       101      18.75

  night.mins night.calls night.charge intl.mins intl.calls intl.charge svc.calls churn
1      244.7          91        11.01      10.0          3        2.70         1    No
2      254.4         103        11.45      13.7          3        3.70         1    No
3      162.6         104         7.32      12.2          5        3.29         0    No
4      196.9          89         8.86       6.6          7        1.78         2    No
5      186.9         121         8.41      10.1          3        2.73         3    No
6      203.9         118         9.18       6.3          6        1.70         0    No
```

We want to fit a classification model to predict churn, and use cross-validation to determine the optimal tree size.

Solution. We start by fitting a "default" tree, using all 18 predictors. The default value of c_p that R uses is 0.01. By lowering this somewhat, we can get a tree that is intentionally over-fitted. My choice of $c_p = 0.0001$ is an arbitrary guess:

```
> fit1 <- rpart(churn~.,data=churn,method="class", cp=0.0001)
> text(fit1)
> plot(fit1)
```



Yeah, now that's a mess! Now, let's look at some cross-validated error rates. We can get these by either printing or plotting the c_p values for this tree object. The functions `printcp()` and `plot()` do the work:

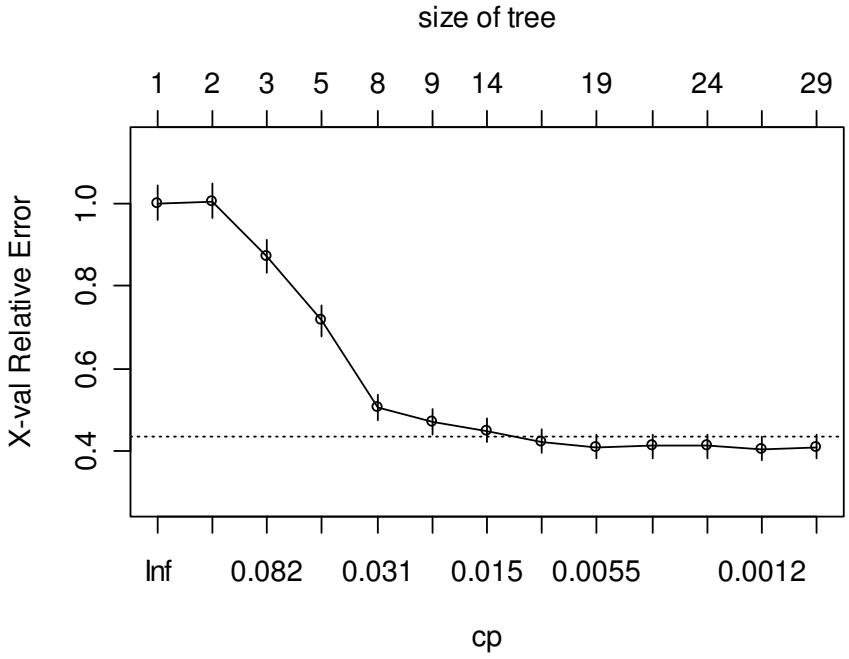
```
> printcp(fit1)

Classification tree:
rpart(formula = churn ~ ., data = churn, method = "class", cp = 1e-04)
```

```
Variables actually used in tree construction:
 [1] day.mins    eve.calls   eve.mins    intl    intl.calls  intl.mins
 [7] night.calls night.mins  svc.calls   vmail

Root node error: 483/3333 = 0.14491

n= 3333

           CP nsplit rel error  xerror     xstd
1  0.08902692      0   1.00000 1.00000 0.042076
2  0.08488613      1   0.91097 1.00414 0.042148
3  0.07867495      2   0.82609 0.87164 0.039707
4  0.05279503      4   0.66874 0.71636 0.036458
5  0.01863354      7   0.47412 0.50518 0.031134
6  0.01759834      8   0.45549 0.47205 0.030174
7  0.01242236     13   0.36232 0.45135 0.029552
8  0.00724638     16   0.32505 0.42443 0.028717
9  0.00414079     18   0.31056 0.41201 0.028321
10 0.00310559     19   0.30642 0.41408 0.028388
11 0.00207039     23   0.29400 0.41408 0.028388
12 0.00069013     25   0.28986 0.40787 0.028187
13 0.00010000     28   0.28778 0.41201 0.028321
```

The cross-validated error info is given in the column labeled `xerror`. Note that there are many different trees (with varying numbers of splits) that produce roughly the same `xerror` value, but the optimum appears to be at 25 splits (using a c_p value of 0.00069). For this number of splits, the absolute cross-validated error rate is given by $0.14491 \times 0.40787 = 5.91\%$.

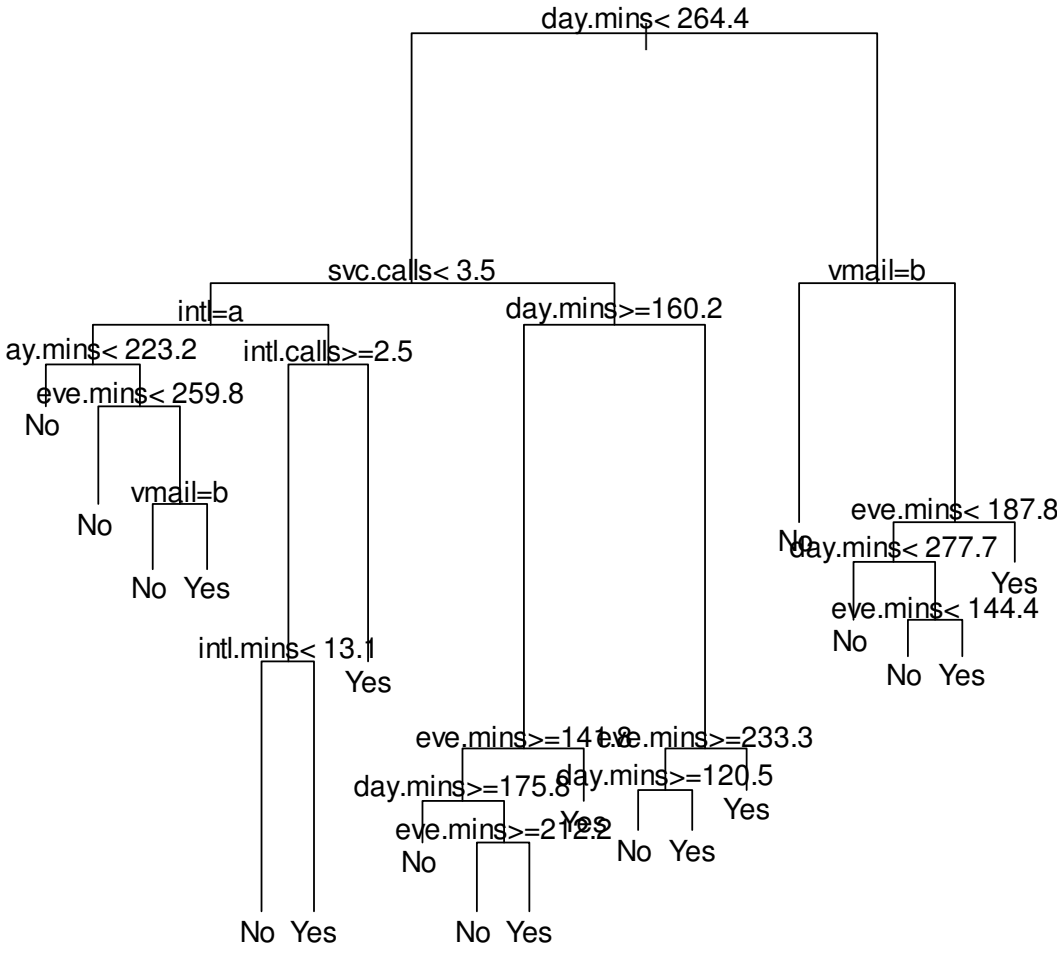
Here is a visual representation of the same information:

```
> plotcp(fit1)
```



The previous picture seems to more strongly suggest that a smaller tree (using $c_p \approx 0.0055$) will do essentially as good a job as the more complex option. So, I now "prune" the tree back to this point:

```
> fit2 <- prune(fit1, cp=0.0055)
> plot(fit2)
> text(fit2)
```



It's somewhat simpler than our first model, though still pretty complicated. It's clear that predicting churn is not a simple matter! ◀