Teaching Statistical Inference via Simulation using R

A CAUSE Webinar

Daniel Kaplan

Macalester College
St. Paul, Minnesota, USA

Oct. 14, 2008

Outline

1. BEFORE THE WEBINAR:
   1.1 Install R.
   1.2 Download the seminar software and data files from the web.
   1.3 Learn a few basic R commands.
2. Inference as simulation.
3. Using R for simulation.
4. Confidence Intervals & Coverage
5. Hypothesis Testing
6. Simulations using Causal Networks to illustrate Confounding and Covariation

BEFORE THE WEBINAR: Objectives

1. Show how computer-based simulations can be used to teach students the logic of statistical inference.
2. Show a good computer notation, based in R, that provides an effective way for students to observe the consequences of randomness.
3. Help participants to see that the power of R is quite accessible to them. The start-up costs of learning the necessary commands are not great.

BEFORE THE WEBINAR: Some Claims

Behind each of the objectives is a claim.

1. Motivating CLAIM: Students can more easily learn the principles of inference when they can see directly the role that randomness plays. The formulas that are commonly used in teaching statistics describe the randomness but in a way that is too abstract for any but a handful of students to understand.
2. Motivating CLAIM: A language-based approach to computing let's students see the structure of statistical inference: how the parts fit together. (For some background, see `http://repositories.cdlib.org/cgi/viewcontent.cgi?article=1003&context=uclastat/cts/tise`.)
3. Motivating CLAIM: There are start-up costs to the language-based approach: a few hours of practice and some frustration when you make mistakes. But the investment pays off.

BEFORE THE WEBINAR: Limitations of the Webinar Format

Presenting this material in a classroom setting requires approximately 4 hours spread around the semester. This gives time to provide enough time for the concepts to sink in, for students to develop fluency with the notation, and to work with students at a humane pace that allows them to make natural mistakes and then correct them.

- We have only 30 minutes for this webinar.
  This means that there is not enough time for you to become fluent with the notation. You will make typographical mistakes and you may make mistakes of a more conceptual nature. You won't have much time to correct them.
- I cannot see what you are typing.
  When you make a mistake, I will not be able to point it out to you and show you what's wrong.

BEFORE THE WEBINAR: Dealing with the limitations

- If you can, practice a bit before the webinar with the commands given after the installation instructions.
- I'm going to show you the basic structure of the notation and give some simple statistical examples of using it to understand inference.
- The slides marked "Supplemental Material" won't be reviewed during the webinar. They provide some background and detail, and contain extension examples of greater richness.
- I suggest you try to follow along on your own computer. But, when you make a mistake, do not get obsessed with correcting it. Let it drop and come back to it later, after the webinar. Then you will have the time to compare your statement to that given on the slides.

BEFORE THE WEBINAR: Install R

- It takes only about 5 minutes to install R. Please do so before the webinar begins. It will not reconfigure your system or do any damage to existing software. It's free and runs on all the major operating systems: Windows, Mac OS X, Linux. The web site is `www.r-project.org`
- Installing R is usually very easy, but not everyone is experienced with using a web browser to download files, finding the files once they have been downloaded to their computer, or installing new software. If you have problems, contact the webinar presenter: Danny Kaplan, `kaplan@macalester.edu`, 651-695-1877. I'll talk you through it.
- If you already have R installed on your system, make sure that it is a relatively recent version (2.6 or more recent). If not, install the newest version.

BEFORE THE WEBINAR: Install R

- To install R, you download one file using your web browser. Links to the current version are given below.
  - **Windows** To install R, execute the downloaded file. You will be prompted with several questions; you can accept all of the default settings. Link: `http://streaming.stat.iastate.edu/CRAN/bin/windows/base/release.htm`
  - **Mac OS X** The file is a "disk image" and will appear in the Finder as such. To install R simply double-click on icon of the multi-package "R.mpkg" contained in the R-2.7.2.dmg disk image. Link: `http://streaming.stat.iastate.edu/CRAN/`

  More generally you can access the latest version of the software through the Download/CRAN link on the main R web page. You will want the "base" distribution, which is distributed as a "binary" file appropriate for your operating system.

BEFORE THE WEBINAR: Install the software and data.

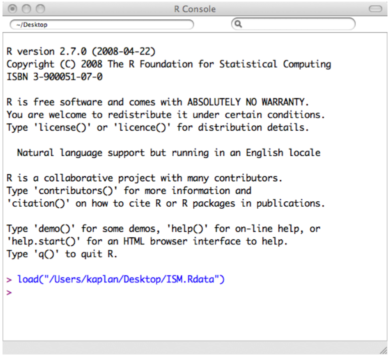
All of the data that we need for this webinar, and some special software in R that makes simulations easy, is contained in a single file available on the web:

- Data/software file:
  `http://www.macalester.edu/~kaplan/ISM/ISM.Rdata`

Use your browser to download this file onto your computer.

BEFORE THE WEBINAR: Starting R

To start R, find the `ISM.Rdata` file that you installed in the previous step. Double-click on it. This should start R and read in the software and data sets. The window should look something like this:



```
R version 2.7.0 (2008-04-22)
Copyright (C) 2008 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.

> load("/Users/kaplan/Desktop/ISM.Rdata")
>
```

BEFORE THE WEBINAR: Checking that things work

Just to make sure that everything is working, try two simple R commands.

In the R window, you type commands at the prompt, the little > character that starts the last line.

First, do this one:

```
> 3 * 7
```

When you press return, R should print a line that looks like this:

```
[1] 21
```

Next, do this one (taking care to spell out the word "five"):

```
> five * 7
```

The response should be

```
[1] 7 7 7 7 7
```

If either of these commands doesn't work, contact the webinar presenter by phone or email: Danny Kaplan (`kaplan@macalester.edu`) 651-695-1877

BEFORE THE WEBINAR: Review some R Notation

Please try out these few statements, just to develop some "finger memory" so that you'll be better able to follow the webinar. These examples are more abstract than the ones in the actual webinar. Don't worry. We'll be using real data in the webinar. Remember, you will type just the statement to the right of the > prompt. When you press return, R will do its work.

1. Do a computation $(3 + 2)$ and view the result.

   ```
   > 3 + 2
   [1] 5
   ```

2. Re-do the computation and store the resulting value in an object named `x`:

   ```
   > x = 3 + 2
   ```

   This is called "assignment" to `x`.

BEFORE THE WEBINAR: Reviewing R Notation

3. Look at the value stored in `x`:

   ```
   > x
   [1] 5
   ```

4. Create a small set of numbers, store it in `y`, and look at the value:

   ```
   > y = 1:5
   > y
   [1] 1 2 3 4 5
   ```

5. Add `x` and `y`, without storing the result:

   ```
   > x + y
   [1]  6  7  8  9 10
   ```

6. Randomly sample from `y` with replacement:

```
> resample(y)
[1] 3 4 4 2 3
```

7. Use the "up arrow" key to recall the previous statement so that you don't have to type it. Modify it so that it looks like this:

```
> resample(y, 40)
 [1] 5 2 5 5 5 3 1 5 5 2 1 3 3 1 1 3 1 5 1 2 5 3 3 2 3
[29] 1 4 4 4 3 3 2 3 2 2 5 2
```

   Note that the two arguments to `resample` are separated by a comma.

8. Randomly sample from `y` without replacement:

```
> resample(y, replace=FALSE)
[1] 5 2 1 3 4
```

   The argument `replace=FALSE` is called a "named argument."

9. Calculate the mean of a $n = 40$ random sample drawn from `y`, without replacement:

```
> mean( resample(y, 40))
[1] 3.15
```

10. Do this again, many times, using the up-arrow to recall the previous statement so that you don't have to type it. You'll see some variability in the result.
11. Last one ... Generate 100 trials of the mean of an $n = 40$ random sample drawn from `y`, store it in an object, and calculate the standard deviation of the result:

```
> s = do(100)*mean( resample(y, 40) )
> s
  [1] 2.650 3.000 2.875 3.400 3.125 3.375 2.675 2.850 2
 [10] 3.075 3.075 2.875 2.850 2.825 2.775 3.125 2.750 3
      ... and so on.
> sd(s)
[1] 0.2437568
```

Editing Commands

The use of language in computer commands makes it easier (I claim) to see the structure of the logic of statistical inference. But, undeniably, it makes it easier to make typographical mistakes. This is a trade-off, but it is a worthwhile trade-off.

R provides an editing capability that makes it easier to build new commands out of old ones. At the heart of this are the arrow keys on your keyboard.

- If you **start with the cursor at the latest prompt**, the up arrow will recall the previous command.
- Successive presses of the up arrow navigate up the history of commands. The down key moves down the history.
- Use the left and right arrows to move within the command. Typing and deleting do the expected thing.

Three minutes of practicing will get you familiar with this.

BEFORE THE WEBINAR: Practice editing within R

1) Within R, enter and evaluate this command

```
> 3 + 2
[1] 5
```

2) Now use the up arrow (don't retype!) to bring back that command and edit it to look like this:

```
> 32 + sqrt(2)
[1] 33.41421
```

3) Use the up arrow again to recall that command and edit it to look like this

```
> x = 32 + sqrt(2)
```

4) Now use the arrow to get all the way to the first command (3+2) and edit it to look like this:

```
> y = do(5) * 3 + sqrt(2)
```

5) Finally, just type y to see the result

```
> y
[1] 4.414214 4.414214 4.414214
[4] 4.414214 4.414214
```

Practice such editing until you feel comfortable with it.

BEFORE THE WEBINAR: Incomplete statements

A very common mistake involves omitting the closing parethesis in a statement around a file name. R handles this gracefully, but you should be aware of what it does.
Try typing this statement, which misses a closing parethesis:

```
> sqrt( 4
```

When you press return, R will recognize that the statement is incomplete and prompt you to complete it with a +:

```
> sqrt( 4
+
```

Just type the closing parenthesis after the + and press ENTER.

Sometimes the mistake will be more involved and R will keep prompting with +. In such a situation, you can bail out by pressing ESC and, perhaps, additional closing parentheses or quotes.

BEFORE THE WEBINAR: Last step.

If the commands are working, you are all set.

- Quit R using the usual menu commands. You will be asked if you want to save the workspace. Say no.
- Keep track of the location of the `ISM.Rdata` file. You will need this on the day of the webinar.
- If you like, you may want to read a short tutorial on using R. This is at `http://www.macalester.edu/~kaplan/ISM/draft-intro.pdf`. Just read Section 1.4.
- Ahead of time on the day of the webinar, start up R with the `ISM.Rdata` file. Then just leave R running until the webinar begins.

Congratulations! You are all set for the webinar. Also, you now have installed on your computer a powerful, professional-level system for statistical computation which you can use for many purposes beyond those of the webinar.

AFTER THE WEBINAR

My hope is that, after the webinar, you will want to bring some of these ideas to your classes.

- I have included many extra slides containing elaborations or further examples — more than we could possibly do in the 30 minutes of this webinar.
- I encourage you to think about using R for the non-simulation computations in your courses. This works well and makes it easier for students to make the transition from descriptive statistics to inference. Some resources that you might find helpful:
  - Daniel Kaplan, *Introduction to Statistical Modeling*
  - John Verzani *Using R for Introductory Statistics*
  - Michael J. Crawley. *Statistics: An Introduction using R.*

Teaching Statistical Inference via Simulation using R

A CAUSE Webinar

Daniel Kaplan

Macalester College
St. Paul, Minnesota, USA

Oct. 14, 2008

Inference as Simulation

- Statistical inference is often presented in terms of formulas and distributions.
- Underlying those formulas is a conception of a *process*. George Cobb has described that process in terms of three components:
  - Randomize
  - Repeat
  - Reject
- Formulas such as $t = (m_1 - m_2)/s_p\sqrt{1/n_1 + 1/n_2})$ present the results of the inference process as worked out by the rules of probability calculus.
- Another way: simulate the process and base the inference on the outcomes of the simulation. Technical terms are bootstrapping and resampling.

Algebra and Computer Notation

For half a century, from the days of FORTRAN, computer languages have provided an algebra-like notation for arithmetic, e.g.,

```
> sqrt( 3^2 + 4^2)
[1] 5
```

Most people have little trouble with the idea of *assignment*: storing a result for re-use.

```
> n = 10
> x = 1/sqrt(n)
> x
[1] 0.3162278
```

Computer Concepts

Admittedly, the computer notation has differences from the algebraic notation. For example, the statement

```
x = x + 1
```

is perfectly sensible to the computer, but not valid algebraically. (It means: give `x` a new value that will be the old value plus 1.)

Randomization

There is no standard algebraic notation for "randomize." And randomization can mean different things: shuffling, sampling with replacement, shuffling within groups, etc.
However, a single, simple operator will do what's needed to understand many forms of statistical inference: random sampling with replacement.

```
> pop = 1:5
> pop
[1] 1 2 3 4 5
> resample(pop)
[1] 4 2 5 2 5
> resample(pop)
[1] 4 5 1 4 5
```

Samples of different sizes:

```
> resample(pop,3)
[1] 1 3 1
> resample(pop,3)
[1] 5 5 2
> resample(pop,10)
[1] 2 3 4 4 2 1 3 3 1 1
```

Computer Concepts

`Resample` can also sample without replacement, e.g., shuffling:

```
> resample(pop,replace=FALSE)
[1] 3 2 4 5 1
```

There is a `shuffle` operator that makes this easier:

```
> shuffle(pop)
[1] 2 5 3 4 1
```

A Simple Notation for “Repeat”

The do notation makes it straightforward to do many of the sorts of repetition used in statistical inference.

```
> do(10) * mean(resample(pop))
 [1] 3.0 2.4 4.0 3.4 2.8 2.8 2.8 2.2 3.2 4.0
> do(10) * max(resample(pop))
 [1] 3 5 4 5 5 5 4 5 5 4
> do(10) * sd(resample(pop))
 [1] 1.6733201 1.4832397 1.5165751 0.8366600 0.8944272
 [6] 1.6431677 1.0954451 0.8366600 1.5165751 0.8366600
```

Computer Concepts

In most computer languages, the process of “repeating” involves somewhat complicated syntax: loops, counters, accumulators, and such. Teaching this to students is not so easy and arguably distracts them from statistical concepts. The `do` notation is less general than a loop, but easier to use for simple tasks involving repetition.

Computing with Real Data

Real data consists of cases and variables, not just simple sets of numbers.

- Read in some data. I name this pop because I want you to think of this as a population.

```
> pop = ISMdata("ten-mile-race.csv")
```

It looks like this

```
     state time  net age sex
1       VA 6060 5978  12   M
2       MD 4515 4457  13   M
3       VA 5026 4928  13   M
  ... and so on ...
8635    ND 9726 9231  75   F
8636    VA 7058 7058  87   F
```

From the Cherry Blossom 10-mile road race held in Washington, DC in 2004. Variables: the age and sex of the runner, the state where the person lived, and the time (in seconds) from the start gun to crossing the finish line; net is the time from the start line to the finish line.

Simple Statistics

- Calculate the mean of a variable.

```
> with( pop, mean(time))
[1] 5813.148
> with( pop, median(age))
[1] 35
```

- Select a random sample of size $n = 100$

```
> mysamp = resample( pop, 100 )
```

- Calculate sample statistics

```
> with(mysamp, mean(time))
[1] 5845.5
> with(mysamp, median(age))
[1] 34.5
```

EXTENSION: Statistical Models

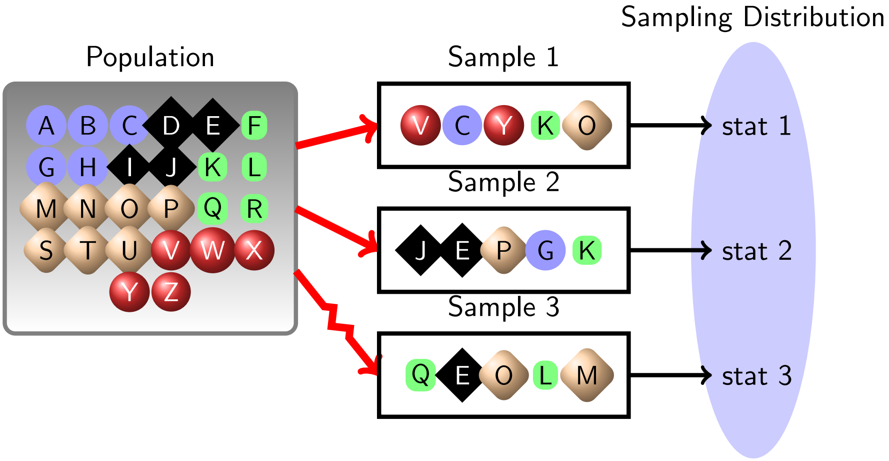
Due to the 30-minute time constraint of this workshop, I'm working with simple statistics such as the sample mean, sd, IQR, etc.
It's a bit richer to work with linear models, as I do in my courses.
R has a particularly strong notation for models. Example: Model running time by age:

```
> with(pop, lm( time ~ age) )
(Intercept)          age
   5547.829        7.199

> with( pop, lm( time ~ age + sex))
(Intercept)          age         sexM
    5592.61        16.49      -775.91

> with( pop, lm( time ~ age * sex))
(Intercept)          age         sexM     age:sexM
   5630.490       15.384     -845.526        1.911
```

The Process behind Sampling Distributions

- Randomize: Draw a new random sample from the population and calculate your sample statistic.
- Repeat: Do this many times and form the distribution of the results: this is the sampling distribution.

The Process behind Sampling Distributions



Simulating this on the Computer

My sample statistic:

```
> with(mysamp, mean(time))
[1] 5845.5
```

Now, draw a new random sample and find the mean time. Three times, as in the diagram:

```
> with( resample(pop, 100), mean(time))
[1] 5851.84
> with( resample(pop, 100), mean(time))
[1] 5775.59
> with( resample(pop, 100), mean(time))
[1] 5886.71
```

Drawing from The Sampling Distribution

The do operator lets you collect many trials. Here are a dozen:

```
> do(12) * with( resample(pop, 100), mean(time))
[1] 5606.31 5996.19 5732.24 5868.49 5607.31 5893.15 5962.5
[8] 5826.24 5692.76 5871.56 5869.97 5684.92
```

500 trials can give a good picture of the sampling distribution:

```
> trials = do(500) * with( resample(pop,100), mean(time) )
```

Describing the Sampling Distribution

Each of the 500 entries in `trials` is a draw from the sampling distribution.

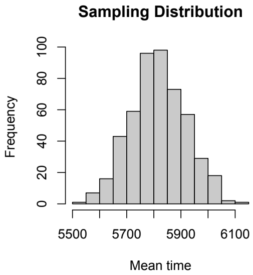
- Plot a histogram

```
> hist(trials)
```

- Describe the distribution:

```
> mean(trials)
[1] 5816.408
> sd(trials)
[1] 103.7016
```

  The standard deviation of the sampling distribution is the standard error.

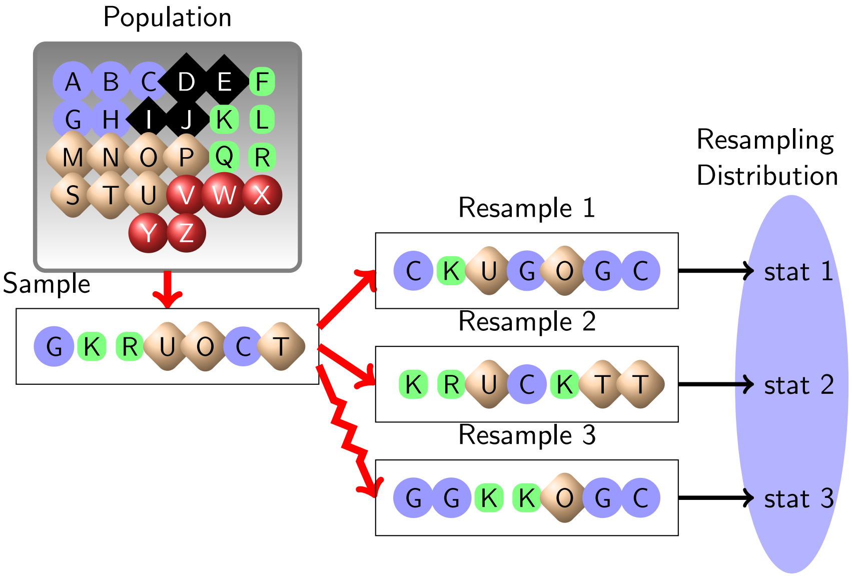


Resampling

Typically, it's impractical to draw repeated samples from the population. We acquire our sample with difficulty; repeating it is not feasible.

Instead of repeating the draws from the population, we draw instead from the sample. This is a matter of copying the entries from the sample rather than going to the field for new data.

Sampling from a sample: RE-sampling.


Population
A B C D E F
G H I J K L
M N O P Q R
S T U V W X
Y Z
Sample
G K R U O C T
Resample 1
C K U G O G C
Resample 2
K R U C K T T
Resample 3
G G K K O G C
Resampling Distribution
stat 1
stat 2
stat 3

The Resampling Operator

The `resampling` operator samples *with replacement*. Some cases may be omitted, some duplicated in the resample.

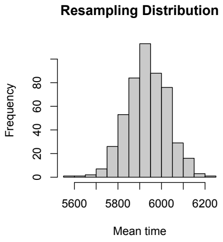
```
> resample(mysamp)
       state time  net age sex
2337      MD 4042 4028  40   M
800       MD 5555 5505  29   M
3897      NJ 6562 6362  55   M
2054      VA 4126 4110  38   M
2054.1    VA 4126 4110  38   M
2202      DC 4063 4050  39   M
3828      MD 6479 6210  54   M
2899      VA 4850 4679  44   M
800.1     MD 5555 5505  29   M
2928      VA 5746 5507  44   M
```

Note that case 2054 appears twice in the resample, as does case 800.

Constructing the Resampling Distribution

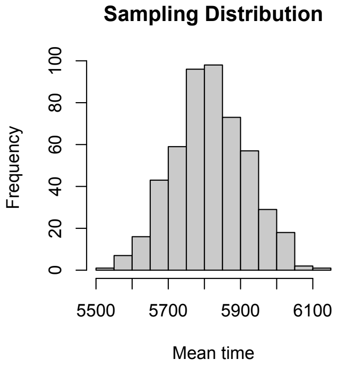
Just like the sampling distribution, but sample from our *original sample*, not the population.

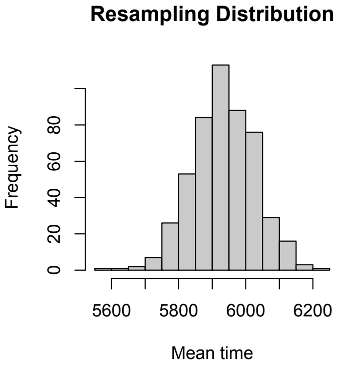
```
> trials = do(500)*with(resample(mysamp), mean(time))
> mean(trials)
[1] 5934.294
> sd(trials)
[1] 93.60788
> hist(trials)
```



The Resampling Distribution is Different!

The resampling distribution is systematically different from the sampling distribution — it has the wrong mean.
But the shape and standard deviation are about right.





Confidence Intervals and the Resampling Distribution

The standard deviation of the resampling distribution approximates the standard deviation of the sampling distribution. So use it as the standard error in constructing confidence intervals.

```
> with(mysamp, mean(time))
[1] 5935.02
> trials = do(500) * with(resample(mysamp), mean(time) )
> sd(trials)
[1] 93.60788
```

Giving $5935 \pm 2 \times 94$, that is, 5748 to 6124.

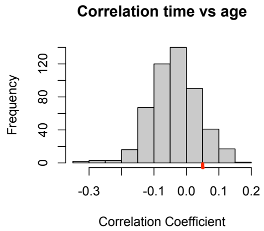
Or, take the coverage interval on the resampling distribution:

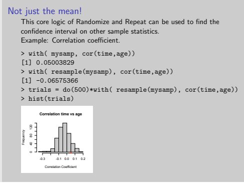
```
> quantile( trials, c(.025, .975))
    2.5%     97.5%
  5753.5   6117.3
```

Not just the mean!

This core logic of Randomize and Repeat can be used to find the confidence interval on other sample statistics.
Example: Correlation coefficient.

```
> with( mysamp, cor(time,age))
[1] 0.05003829
> with( resample(mysamp), cor(time,age))
[1] -0.06575366
> trials = do(500)*with( resample(mysamp), cor(time,age))
> hist(trials)
```

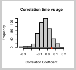




Not just the mean!

This core logic of Randomize and Repeat can be used to find the confidence interval on other sample statistics.
Example: Correlation coefficient.

```
> with( mysamp, cor(time,age))
[1] 0.05003829
> with( resample(mysamp), cor(time,age))
[1] -0.06575366
> trials = do(500)*with( resample(mysamp), cor(time,age))
> hist(trials)
```



Some other sample statistics. Note the common structure to the simulations:

```
>trials=do(500)*with(resample(mysamp),median(age))
>trials=do(500)*with(resample(mysamp),sd(age))
>trials=do(500)*with(resample(mysamp),IQR(age))
>trials=do(500)*with(resample(mysamp),quantile(age,.8))
```

How does the SE depend on *n*?

Do an experiment, varying *n*.

```
> s25 = do(500)*with( resample(pop,25), mean(time))
> sd(s25)
[1] 199.9869
> s100 = do(500)*with( resample(pop,100), mean(time))
> sd(s100)
[1] 106.2490
> s400 = do(500)*with( resample(pop,400), mean(time))
> sd(s400)
[1] 53.0887
> s1600 = do(500)*with( resample(pop,1600), mean(time))
> sd(s1600)
[1] 25.27962
```

Quadrupling the sample size halves the SE.

Inference on Models

Because the same core logic applies broadly, you can discuss more interesting statistics.
Example: What is the time difference between the sexes?

```
> with( pop, lm( time ~ sex))
(Intercept)          sexM
     6156.2        -687.1
```

Men are 687 seconds faster. What's the SE?

```
> trials = do(500) * with(resample(pop), lm( time ~ sex))
> sd(trials)
(Intercept)         sexM
   15.51795     21.50236
```

So, 666 to 709 seconds faster.

Including covariates

Example: What is the time difference between the sexes, taking age into account.

```
> with( pop, lm( time ~ age + sex) )
Coefficients:
(Intercept)          age         sexM
    5592.61        16.49      -775.91
```

Men are faster by 776 seconds. What's the SE?

```
> trials = do(500)*with(resample(pop), lm( time ~ age + se
> sd(trials)
(Intercept)          age        sexM
  39.671063     1.099228   21.617360
```

About 100 seconds of the faster speed is accounted for by the different ages of men and women.

Computer Concepts: `with`

Both `pop` and `mysamp` are data frames, consisting of cases and variables. Think of a data frame as a family, with the family members consisting of the variables.

When you are within a family, you can refer to the members just with their first name. This is what the `with` operator lets you do.

The `with` operator takes two arguments: the data frame and the calculation to do with that frame.

Another style, which might be preferable to some people, involves giving the full name of each variable. This is as if you referred to people always with both their last name (to identify the family) and their first name. This is done with the $ notation, as in `mysamp$time` or `pop$age`. Think of this as specifying both the last name and the first name of a person, using $ to separate them. As you might expect, this notation is helpful when there is more than one "family" involved in a single computation. But this is rarely the case in introductory statistics.

Computer Concepts: `with`

Both `pop` and `mysamp` are data frames, consisting of cases and variables. Think of a data frame as a family, with the family members consisting of the variables.
When you are within a family, you can refer to the members just with their first name. This is what the `with` operator lets you do. The `with` operator takes two arguments: the data frame and the calculation to do with that frame.
Another style, which might be preferable to some people, involves giving the full name of each variable. This is as if you referred to people always with both their last name (to identify the family) and their first name. This is done with the `$` notation, as in `mysamp$time` or `pop$age`. Think of this as specifying both the last name and the first name of a person, using `$` to separate them. As you might expect, this notation is helpful when there is more than one "family" involved in a single computation. But this is rarely the case in introductory statistics.

Computer Concepts

For computer mavens ...

The `do(12)` operator is special. You can ask your local computer scientist how it works (mention "operator overloading" and "lazy evaluation"). For the student, however, what's key is that it provides a simple and transparent way to repeat a command many times.

The `with` operator is also special in that it avoids evaluating its second argument until an environment has been set up with the variables contained in the second argument.

Hypothesis Testing

A hypothesis test creates a world where the null hypothesis is true. and examines, in that hypothetical world, whether the outcome of a study are likely to be different from the actual results that we observed in the actual world.

Steps:

1. Record your actual outcome on the data collected in the actual world.
2. Randomly select a sample in the null hypothesis world.
3. Repeat many times
4. Rejection step: check whether the actual outcome is implausible as an outcome from the null-hypothesis world.

Hypothesis Testing using the Sample Mean

PEDAGOGICAL WARNING: I argue below that the sample mean (or the sample proportion) is a bad place to start. But since it's where people often start, that's what I'll do here.

Setting: Conventional wisdom is that the average age of runners is 35. You collect a sample of size $n = 100$, and compute the sample mean

```
> with(mysamp, mean(age))
[1] 37.38
```

The sample age is 2.38 years larger than the null hypothesis age. Is this sample an implausible trial in the world where the average age of runners is 35? If so, you should doubt the null hypothesis based on these data.

Living in the Null Hypothesis World

1. Create a world in which the null hypothesis is true, by modifying the sample accordingly:

```
> nullworld = with(mysamp, age - 2.38)
> mean(nullworld)
[1] 35
```

2. In that world, randomly sample and collect the results

```
> samps = do(1000)* mean( resample(nullworld) )
```

3. See how different our samples (in the null-world) are from the hypothetical value:

```
> table(abs(samps - 35) > 2.38)
FALSE  TRUE
  939    61
```

61 out of 1000 trials were as far off from the hypothetical value as our actual trial. So the p-value is 0.061.

Comparing to a conventional one-sample t-test

If you like, you can also do a t-test. The reported p-value is 0.06075. (It's accidental that the simulation results were so close.)

```
> with(mysamp, t.test(age,mu=35) )

One Sample t-test

data:  age
t = 1.897, df = 99, p-value = 0.06075
alternative hypothesis: true mean is not equal to 35
95 percent confidence interval:
 34.89054 39.86946
sample estimates:
mean of x
    37.38
```

Now the editorial: Why not to do this?

EDITORIAL

Don't start with sample means and proportions for hypothesis testing!

- It's generally a specialized (and often artificial) setting. Why did I pick 35 for the null hypothesis age?
- It obscures a more important general convention for the null hypothesis: no difference among groups or no relationship between variables.
- The calculations of hypothesis testing on sample means and proportions overlaps with that of confidence intervals. But one involves the null hypothesis world and one does not. Confusing!

True, the formula for a one-sample t-test is pretty simple (so long as you ignore the need to look up a p-value in a table).

A more natural setting for hypothesis testing.

Compare two groups.
EXAMPLE: Are the ages of men and women different? Null hypothesis: No relationship between age and sex.

1. What's the observed difference in the actual sample?
   ```
   > with(mysamp, lm( age ~ sex))
   (Intercept)          sexM
         33.73          6.63
   ```
   The observed age difference is 6.63 years — the 2nd coefficient
2. Simulate the null hypothesis world by randomizing the explanatory variable:
   ```
   > with(mysamp, lm( age ~ resample(sex)))
       (Intercept)  resample(sex)M
            35.895           2.396
   ```
   Notice that randomization is applied only to one of the variables. In the simulation we are breaking the relationship between sex and age.

Repeating the simulation in the null world

Idea: count how many times in the simulated null world, the difference between men and women was as large as that observed in the actual sample.

```
> samps = do(1000)*with(mysamp, lm( age ~ resample(sex)))
> head(samps)
  (Intercept) resample(sex)M
1    34.93750      4.6971154
2    37.81395     -0.7613219
3    38.40426     -1.9325572
   ... and so on
```

Now the count:

```
> table( abs(samps[[2]]) > 6.63)
FALSE  TRUE
  995     5
```

Another example

The logic of hypothesis testing is the same for comparing groups as for regression. Example: Running time modeled by age.

1. **Observe** the actual sample:

   ```
   > with( mysamp, lm( time ~ age))
   (Intercept)          age
          6145           -8
   ```

   Running time goes down by 8 seconds per year in our sample.

2. **Simulate** the null hypothesis world and repeat.

   ```
   > samps = do(1000)*with( mysamp,
   +  lm( time ~ resample(age)))
   ```

3. **Count** how often the simulation produces "big" results:

   ```
   > table( abs(samps[[2]]) > 8 )
   FALSE  TRUE
     644   356
   ```

   About 36% of the time, so p=0.36.

Compare to the Regression Report

The point of the simulations here is to demonstrate the logic of hypothesis testing. Once students understand this logic, they can use conventional tests to do the calculations.

```
> mod = with( mysamp, lm( time ~ age ) )
> summary(mod)

            Estimate Std. Error t value Pr(>|t|)
(Intercept) 6144.534    342.333  17.949   <2e-16
age           -8.000      8.687  -0.921    0.359
```

ELABORATION: ANOVA and R^2

ANOVA fits into this same framework, but instead of checking the size of coefficients, check the size of R^2.
Do running times differ among the states of origin?

1. **Observe** in the actual sample:

```
>  with(mysamp, r.squared( time ~ state) )
[1] 0.0602297
```

2. **Simulate** the null hypothesis world:

```
> samps=do(1000)*
  with(mysamp, r.squared(time~shuffle(state)))
```

3. **Count**

```
> table( samps > 0.0602297)
FALSE  TRUE
  339   661
```

The p-value is 0.66.

ELABORATION: ANOVA (cont)

Compare the results of the simulation to the standard ANOVA table.

```
> mod = with(mysamp, lm(time ~ state))
> anova(mod)
Analysis of Variance Table

Response: time
          Df    Sum Sq  Mean Sq F value Pr(>F)
state      8   7000731   875091   0.729 0.6655
Residuals 91 109233128  1200364
```

For the link between two-way ANOVA and ANCOVA and R^2, see *Introduction to Statistical Modeling*.

With or without replacement?

When randomizing an explanatory variable, it makes sense to sample *without* replacement, that is, to shuffle. Generally, it doesn't make much difference which you do, so for introductory students, I just stick with resampling *with* replacement.
This ANOVA example is one situation where it does make a difference. Some of the states in the sample get dropped when resampling without replacement.

Coverage and Confidence Intervals

- The interpretation of confidence intervals is difficult. They don't really describe the sampling distribution.
- But when properly constructed, they have a useful relationship with the population parameter.
- **Your task**: Construct a 68% coverage interval for a model of your choosing and a sample size of your choosing. Then compare that interval to the population parameter and see if your interval covered it.

Did your interval cover the population parameter?

1. Pick your own sample

   ```
   > newsamp = resample( pop, 150, replace=FALSE )
   ```

2. Compute your own sample statistic and the 68% confidence interval

   ```
   > with( newsamp, IQR(age))
   [1] 13
   > samps = do(500)*with( resample(newsamp), IQR(age))
   > sd(samps)
   [1] 1.717770
   ```

   The 68% interval is $13 \pm 1SE = 13 \pm 1.72$ or 11.28 to 14.72.

3. Check if the population parameter is in the interval.

   ```
   > with(pop, IQR(age))
   [1] 16
   ```

No! I lost! But I don't feel bad since I should win only 68% of the time.

ELABORATION: Doing this with models

Examples of models:

- `mod=with(newsamp,lm(time~age))`
- `mod=with(newsamp,lm(time~sex*age))`

You can have students do intervals “by hand,” but here we’ll just use software. Remember: it’s a 68% confidence interval.

```
> mod = with(mysamp, lm( time ~ age))
> intervals(mod, level=0.68)
                 lower        upper
(Intercept) 5802.36245 6486.7056551
age          -16.68233    0.6826484
```

Now compute the population parameter:

```
> with(pop, lm( time ~ age))
(Intercept)          age
   5547.829        7.199
```

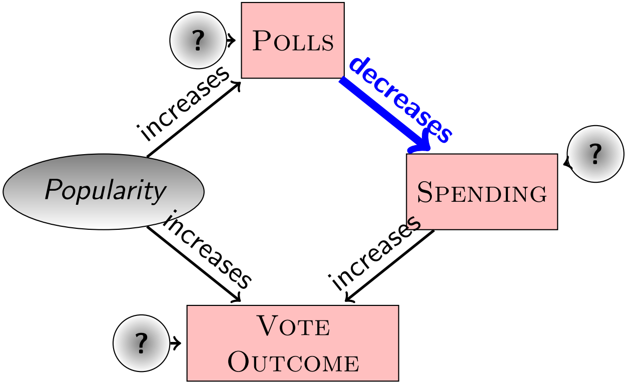
Again, for this sample, the population parameters fall outside of the interval.

Simulating Causal Connections

- There is a strong (and correct) disposition to teach statistics with “authentic data.”
- Since inference is about how to reason from data to the real world, it helps to have examples where we know what the real world is. That’s why teachers often use data from familiar settings, e.g., heights.
- But for teaching process (selecting covariates, designing experiments, etc.), it helps to be able to simulate the real world.

Simulation Example: Campaign Spending

Does spending by incumbent politicians increase their fraction of the vote? Studies have claimed not: there is a negative correlation between incumbent spending and vote percentage.
But, the reason that incumbents spend a lot is that they are in hotly contested elections. That's why the vote percentage goes down when incumbents spend a lot.



Running the Simulation

```
> campaign.spending
Causal Network with  4  vars:
========================================
popularity is exogenous
polls <== popularity
spending <== polls
vote <== popularity & spending
> run.sim(campaign.spending)
  popularity    polls  spending     vote
1   43.95159 48.09180 38.976502 37.46474
2   81.62916 88.35420  2.815538 70.69485
3   79.69063 81.30378 12.343475 55.90033
4   77.35416 75.77626 14.712796 55.08223
5   76.67314 76.16270 26.461983 61.39772
```

Running the Simulation

Collect the data and build models:

```
> spend = run.sim(campaign.spending, 1000)
> mod1 = with(spend, lm( vote ~ spending))
> summary(mod1)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 67.27230    0.64925  103.62   <2e-16 ***
spending    -0.33736    0.01171  -28.81   <2e-16 ***
```

Spending is significantly negatively associated with votes. But ...
Is popularity confounding the relationship between votes and spending?

Another model of the same data shows the opposite.

```
> mod2 = with(spend, lm( vote ~ spending + polls) )
> summary(mod2)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.43022    1.83616   0.234    0.815
spending     0.25407    0.01753  14.493   <2e-16
polls        0.74139    0.01983  37.390   <2e-16
```

Which is right?

Do an Experiment!

We can't control spending, but we can donate money to the campaigns and see what the results are:

```
> donations = resample( c(0,10), 1000)
> exper = run.sim(campaign.spending, spending=donations,
    inject=TRUE,1000)
> mod3 = with( exper, lm( vote ~ donations) )
> summary(mod3)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 49.66413    0.51296  96.818  < 2e-16
donations    0.32163    0.07269   4.425 1.07e-05
```

The experiment says that spending increases votes.

Summary

- The formula-based approach to test statistics hides the core logic of inference: Randomize, Repeat, Reject.
- Computer simulations let students see the core logic in action.
- Since the core logic is the same for a wide range of statistics, inference can easily be extended to models.
- Simulations based on causal networks let students observe confounding, test out experimental design, and explore other statistical concepts.
- The text for a complete introductory-level course, based on modeling and simulation, is available from the presenter. See `http://www.macalester.edu/~kaplan/ISM`