Statistical Simulations

Amelia McNamara

November 2016

1 Introduction

Bill asked me to spend time this month thinking about how statistical simulations might be incorporated into CODAP. When I think of "statistical simulations" I consider two main categories:

1. Simulations from a model to generate data under that model
2. Simulations from data to generate new samples, either
   (a) randomized or
   (b) bootstrapped

When I say "simulations from a model" I am mostly referencing the types of things possible in the TinkerPlots Sampler. Generating data from flipping a coin, drawing from a theoretical distribution, picking a card, picking a ball, etc. In this sense, our 'model' is a coin that comes up 50/50 heads/tails (or something else, as defined by the user). We are generating data from that model. To me, this type of work is more *probability* than *statistics*. I suppose data could be generated from more complex models– the equation for a line with some quantification of variability could be used to generate data consistent with that model– but it's not clear what the purpose would be.

On the other hand, "simulations from data" feel much more statistical to me, so that is my personal interest and what I will focus on.

My argument is that randomization and simulation should be integrated throughout CODAP, and possible to apply to any situation: running a model, taking a mean, making a plot, etc. Since many statistics educators are turning toward simulation for their courses (Diez et al., 2014), this would make it a more valuable pedagogical tool. Ideally, the interface would feel similar for randomization, the bootstrap, and any simulations from models that the user was doing. However, there would need to be sufficient differentiation that learners could understand the distinction between the methods.

Typically, we use randomization to generate a null sampling distribution (or null plots!) while we use the bootstrap to generate a sampling distribution centered around the point estimate (or plots showing potential sampling variability). For a more in-depth explanation of these topics, see Appendix A and B, or read Hesterberg (2015).

One thing that is interesting about these techniques is that they are inherently pretty interactive. It is hard to describe them with words, and much more intuitive to see them at work. I've given

a few talks related to these techniques, including 'Do you know Nothing when you see it?' at OpenVisConf (`https://www.youtube.com/watch?v=hps9r7JZQP8`). For a demonstration of what randomization looks like, start the video around 4:20, and for a demonstration of the bootstrap start at 6:40.

I like these topics because they are so generally applicable to statistical problems, and don't rely on distributional assumptions. Even in 1997, Rolf Biehler was calling for the integration of these methods (Biehler, 1997). My hypothesis is that they are more intuitive to learners, although there is very little research on this yet.

One of the few papers is Pfannkuch et al. (2014), an observational pilot study that outlines problem areas, like "When random allocation to two groups is misunderstood, inadequate concepts are formed" (Pfannkuch et al., 2014). Maxine Pfannkuch, Chris Wild, and Matt Regan close by saying "verbalization of structure and concepts in graph displays and word-only contexts" presents problems for students and more work is needed on how to "develop a habit of mind in students."

One way to develop this habit of mind is to reinforce it with many repetitions, so I believe bootstrap and randomization methods should be integrated throughout tools like CODAP. This means making it possible to generate many "samples" from existing data and denote them as different than the original data, while allowing students to combine parameter estimates. It also means integrating the methods into any other aspects of the system, for example plots or statistical modeling (i.e. fitting a linear regression model). Because the plotting methods are generally less well-known than the corresponding numeric methods, I'll address them in the main text.

2 Graphical methods

2.1 Randomized plots

Just as we can use randomization to generate a null distribution, a similar approach can be used with plots. Wickham, et al proposed an idea of "graphical inference," where a real plot is set among an array of null plots (Wickham et al., 2010). The idea is that if your eye can pick the real data out, that is essentially like the $p < 0.05$ cutoff that is commonly used in statistics. To see me talking about this, begin watching `https://www.youtube.com/watch?v=hps9r7JZQP8` around 9:30.

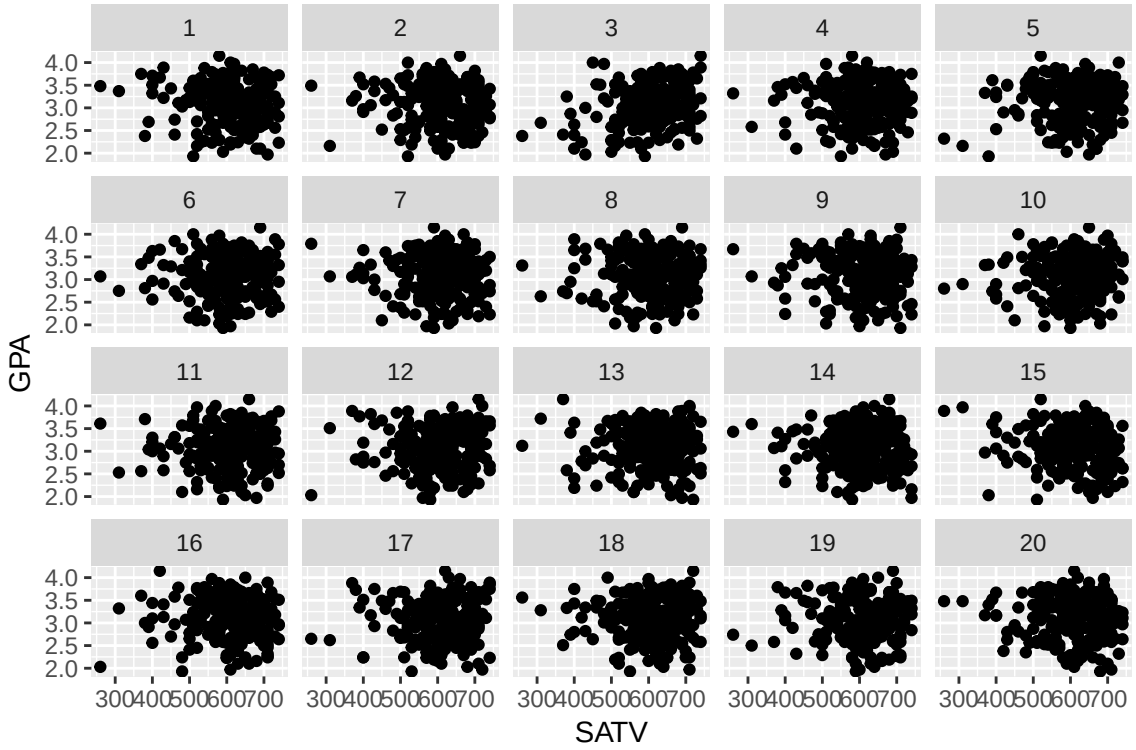
For this document, we'll be thinking about some data on college students' first year GPA. Let's see if we can pick the real relationship between SAT verbal score and first year college GPA out of a set of randomized plots.

```
library(mosaic)
require(Stat2Data)
data(FirstYearGPA)
head(FirstYearGPA)
```

```
##    GPA HSGPA SATV SATM Male   HU   SS FirstGen White CollegeBound
## 1 3.06  3.83  680  770    1  3.0  9.0        1     1            1
## 2 4.15  4.00  740  720    0  9.0  3.0        0     1            1
## 3 3.41  3.70  640  570    0 16.0 13.0        0     0            1
## 4 3.21  3.51  740  700    0 22.0  0.0        0     1            1
## 5 3.48  3.83  610  610    0 30.5  1.5        0     1            1
## 6 2.95  3.25  600  570    0 18.0  3.0        0     1            1
```

```
set.seed(123)
library(nullabor)
ggplot(lineup(null_permute('SATV'), FirstYearGPA)) + geom_point(aes(x=SATV, y=GPA)) + facet_wrap(~.sample)


## decrypt("Be2q dYgY ci Hvxcgcvi z")
```



So, this is an array of null plots (those with the relationship between SAT verbal score and GPA broken) with the real plot placed randomly among the others. After making our guesses, we can check which plot is the true data. We run the decryption,

```
decrypt("Be2q dYgY ci Hvxcgcvi z")


## [1] "True data in position 3"
```

This one is pretty hard to see, which reflects how little practical significance there is to the very small slope value.

The lineup can also be used to produce a number of plots from a specified model, among which a plot of real data is hidden. If the real data looks sufficiently like the modeled data, you can conclude the model is good.

2.2 Bootstrapped plots

Just as the bootstrap can be used to generate a sampling distribution, it can be used in visualization to create "hypothetical outcome plots," or HOPs. This technique takes bootstrapped samples

from the data and remakes the plot many times, to give viewers a sense of the other possible plots they might have seen. Jessica Hullman's OpenVisConf talk focused specifically on HOPs, `https://www.youtube.com/watch?v=pTVAn4oLvbc`. For a quick look at how they work, see 7:00 in the video or scroll through Hullman (2016) on Medium. HOPs do not translate well to paper, but essentially are a series of plots made using bootstrapped data to show possible sampling variation.

Hullman has done research that suggests HOPs are more intuitive than other methods of encoding uncertainty, like error bars (Hullman et al., 2015). They also allow viewers to more easily see correlation between variables, as shown in Figure 1.

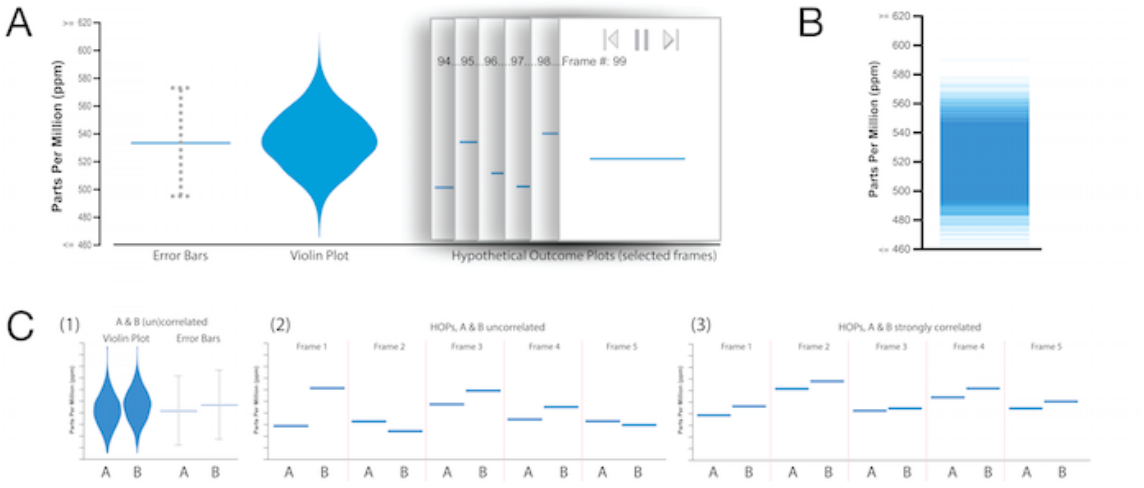


Figure 1: A visual representation of how HOPs help readers better understand correlated groups , from Hullman (2016).

Other methods of making plots to show sampling variation are described in Wild et al. (2011b), including boxplots with memory over repeated sampling for a 'making the call' activity (Wild et al., 2011a). An animated version of the making the call activity can be found in Wild and Chang (2012), and this page was superseded by Wild et al. (2015).

One consideration for displaying variation is how scary it can be. In the recent US presidential election, the New York Times used random jitter on their election prediction dials, with the jitter programmed to be from the 25^{th} to the 75^{th} percentile. People reacted very badly to this online, calling for the journalists behind the graphic to be fired because it wasn't 'true' (see Ford, P. (2016)). So, it seems more important than ever to help people understand variability, but it must be done carefully.

3 Integrating simulation throughout CODAP

In my dissertation, I argue tools should have "Support for randomization throughout" and be "Interactive at every level" (McNamara, 2015, 2016). These two things have rarely coexisted, so it will take some careful thought about how to reconcile them.

3.1 Differentiating data

One important consideration is how the simulated data would be kept separate from the original data. In a scenario where users create derived datasets from their original data, animation could show exactly how the relationship is being broken (randomization) or the data is being drawn (bootstrap). The simulations could all be accompanied by their visual analogues, to allow users to see the variation between samples as the simulation ran.

However, it is not clear that one would want to maintain all 5000 derived data sets, in particular because of how much space that would consume (both in memory and screen real estate). In my `R` examples in Appendix A and B I just saved a particular value (e.g. the slope coefficient) at each step. However, this is an abstraction that might not make immediate sense to learners. So, it seems important that they be shown at least some of the derived datasets in their entirety.

3.2 How quickly to run simulations

Many tools that show simulations have buttons to run the simulation once, 10 times, or 100 times. For example, the Rossman-Chance applet on random assignment of babies (`http://www.rossmanchance.com/applets/randomBabies/RandomBabies.html`) provides a checkbox for users to choose whether they want an animation or not. Then, it allows users to input how many babies they want in each sample and how many samples they want per click. A screenshot of this applet is shown in Figure 2

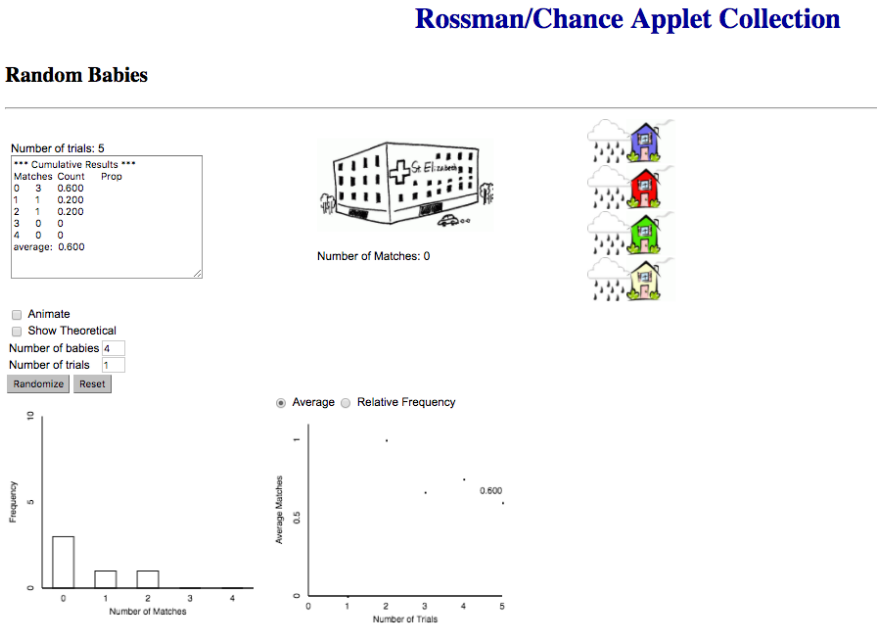


Figure 2: A screenshot of the random babies applet from Rossman/Chance.

Rock 'n Poll (an interactive essay on polling) provides similar functionality (`http://rocknpoll.graphics/`). At first, the interface only allows you to "Survey 1 Pollandian," but after you have clicked to get individual samples a few times, the "Lazy Button" appears to allow you to quickly

generate 10 samples. A screenshot of this essay is shown in Figure 3

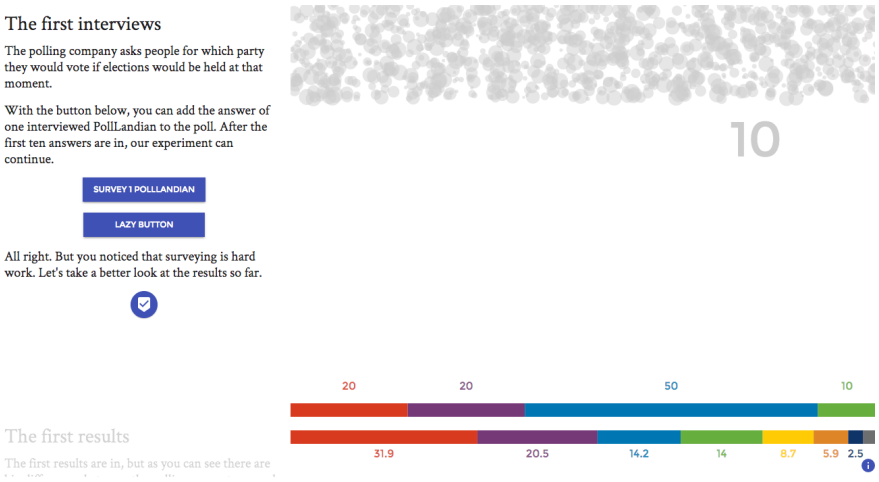


Figure 3: A screenshot of Rock'n Poll.

So, it seems that any interface that can take many samples should start with only allowing users to take one sample, and animating that process very specifically. But, in order for these methods to be powerful, many samples need to be taken. At that point, some approach needs to be taken to conserve computational resources.

One aspect of this is accelerating how quickly the samples are generated (moving from animation to near-instantaneously-updated results). In TinkerPlots, a slider on the Sampler allowed users to choose how quickly they wanted the simulation to run, as shown in Figure 4. This idea could be applied in CODAP.

The other element is how to manage the numerous datasets that would typically be generated. In the R examples given in Appendix A and B, I kept only the regression coefficients derived from the simulated data at each step. Rather than keeping 1,000 separate datasets, I kept 1,000 sets of slope-intercept pairs from the models I ran. This type of setup requires a certain level of computational forethought, but is probably the only reasonable way to implement this type of feature. Perhaps users would see the data at each sample, but the system would discard all but the most recent dataset.

In a Fathom-like interface, an empty dataset (F) could be linked to a model object (G) and that model object linked in turn to another empty dataset (H). As randomized data filled F, a model would be fit in G and then its slope and intercept would be fed into H. As the samples flew by, the coefficients in G would flicker and H would gain more rows.

3.3 Interface choices

In order for randomization and the bootstrap to be integrated into the interface, there must be a way to make them work. While there could be a wizard walking users through specifying a simulation, a more direct interaction might make the operation feel more concrete. Here are a few interface ideas for randomization, as jumping-off points:

1. To create a simulated dataset from existing data, a user would drag out a particular variable

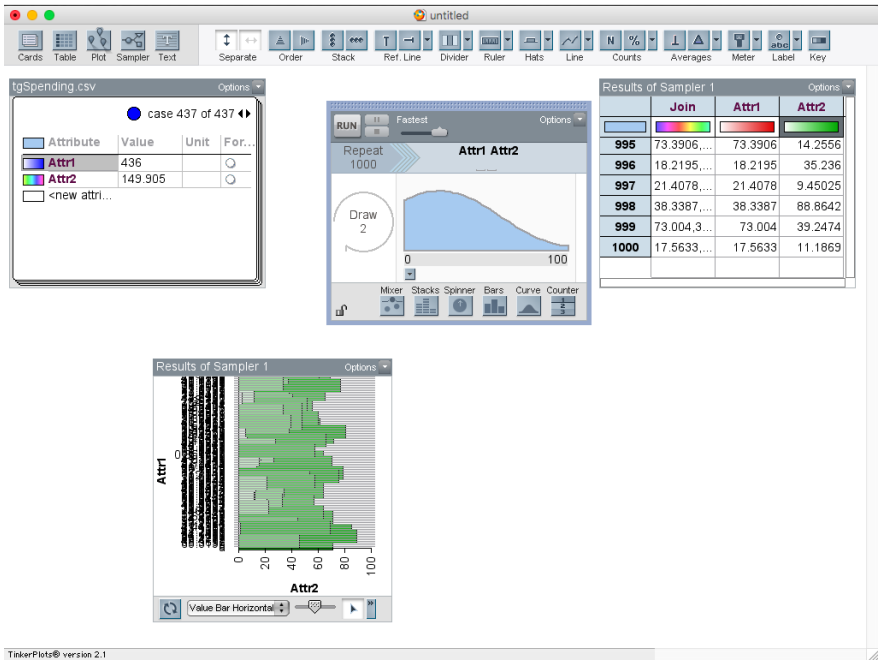


Figure 4: A screenshot of TinkerPlots showing how the speed of the Sampler can be adjusted.

they were interested in, using the top of the column to pull it from the original data to a new data set. Then, they could drag a particular value from another column into the top row of the new dataset, indicating that the relationship between X and Y should be broken.

2. Using the idea of a "randomize" button, a user could click a dropdown on the column they wished to randomize (say, X), and the rectangular data window would duplicate itself before the X column visibly shuffled.

3. Perhaps a particular gesture would be associated with randomization, say a three-finger touch and rotate over a column.

4. An element on the toolbar labeled "randomize" could be pulled out and dropped on to any element in the document. If dragged onto a column, it would randomize it. If dragged onto the axis of a plot, it would randomize the points on that axis.

For the bootstrap, perhaps one of these would work:

1. Pulling one row of the data out of a table would generate an entire bootstrapped sample of the same size as the original data.

2. A similar gesture could be used to indicate that resampling should be used. On a plot, perhaps a five-fingered touch and drag would generate an array of bootstrapped plots.

3. An icon on the data table would indicate bootstrapping. When this was clicked, a new data set would appear and its table header would be a different color than the original data. If the button was clicked on a bootstrap table, another table would appear on top of it (like a card stack, reminiscent of the TinkerPlots data interface) and the original would be obscured.

4. A button in the toolbar with an icon of a lumpy distribution could be dragged out and any quantity dragged to this box would get bootstrapped. If it was a single variable, a sampling distribution for the mean would be generated. If a plot was dragged there, the plot would animate as a Hypothetical Outcome Plot.

In any of these scenarios, if a variable is used for a simulation that simulation should take place across any dependent pieces of analysis. For example, if a randomization is applied in a data table with a dependent plot, that randomization should also be performed on the plot.

4 Concluding thoughts and more questions

Because of the difficulty many people have with understanding sources of variability, it seems important to include generalizable methods like randomization and the bootstrap in intuitive, interactive, and graphical tools. Randomization can help us see a null distribution, bootstrap methods show sampling variability, and even data generated from a priori models can help us see how variation shows up in the real world.

One general statistical issue is the term "sampling distribution," which is at once vague and a bit threatening. "The distribution of the parameter" is almost as bad. I certainly would not want to impose this language on CODAP, so the question remains as to how to describe what we are doing. Distinguishing between real and simulated data seems paramount, as well as making it very explicit what steps are taking place when data is randomized or a bootstrap sample is drawn.

Since this isn't a feature in other interactive software, there are many questions that would need to answered before it could be implemented. A few of these questions are,

1. Does it make sense for CODAP to support randomization and the bootstrap?
2. What other ways could the interface support these methods?
3. Does it make sense to support the other type of simulation, from a priori models?
4. How can we better show sampling variation?
5. What should happen with derived data that is simulated in some way? How is it distinguished from real data?

References

Baumer, B. and McNamara, A. (2016). Lab: Randomization and the bootstrap. http://www.science.smith.edu/~amcnamara/sds291/labs/lab_randomization.html.

Biehler, R. (1997). Software for learning and for doing statistics. *International Statistical Review*, 65(2):167–189.

Diez, D. M., Barr, C. D., and Çetinkaya Rundel, M. (2014). *Introductory Statistics with Randomization and Simulation*. OpenIntro.

Ford, P. (2016). @ftrain: "There's a lot to process right now but don't think we won't be coming back to those NYT election probability gauges with random jitter.". https://twitter.com/ftrain/status/797210858942251009.

Hesterberg, T. (2015). What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. *The American Statistician*, 69(4).

Hullman, J. (2016). Hypothetical outcome plots: Experiencing the uncertain. https://medium.com/hci-design-at-uw/hypothetical-outcomes-plots-experiencing-the-uncertain-b9ea60d7c740#.rlyunor9v.

Hullman, J., Resnick, P., and Adar, E. (2015). Hypothetical outcome plots outperform error bars and violin plots for inferences about reliability of variable ordering. *PLOS ONE*, 10(11).

McNamara, A. (2015). *Bridging the Gap Between Tools for Learning and for Doing Statistics*. PhD thesis, University of California, Los Angeles.

McNamara, A. (2016). Key attributes of a modern statistical computing tool. submitted.

Pfannkuch, M., Wild, C., and Regan, M. (2014). *Using Tools for Learning Mathematics and Statistics*, chapter Students' difficulties in practicing computer-supported data analysis: Some hypothetical generalizations from results of two exploratory studies. Springer.

Wickham, H., Cook, D., Hofmann, H., and Buja, A. (2010). Graphical inference for infovis. *IEEE Transactions on Visualization and Computer Graphics*, 16(6).

Wild, C. and Chang, K. H. (2012). Bootstrap animations. https://www.stat.auckland.ac.nz/~wild/BootAnim/.

Wild, C., Grolemund, G., Stevenson, B., Potter, S., Chang, K. H., and Li, J. (2015). VIT: Visual inference tools. https://www.stat.auckland.ac.nz/~wild/VIT/index.html.

Wild, C., Horton, N. J., Pfannkuch, M., and Regan, M. (2011a). Guidelines for "how to make the call". http://new.censusatschool.org.nz/wp-content/uploads/2012/09/Shifts-diagram.11.02.041.pdf.

Wild, C. J., Pfannkuch, M., Regan, M., and Horton, N. J. (2011b). Towards more accessible conceptions of statistical inference. *Journal of the Royal Statistical Society*, 174(2):247–295.

A Randomization

[The material in this appendix is based in part on a lab from the multiple regression course I am currently teaching (Baumer and McNamara, 2016).]

Randomization is a non-parametric way to build up a sampling distribution under the null hypothesis. If we have two variables we believe are correlated, for example, we might be testing the following hypothesis:

$$H_0 : \rho = 0$$
$$H_A : \rho \neq 0$$

The traditional way to test the hypothesis would be to compute a test statistic from a formula and then compare the value of the test statistic to a corresponding theoretical distribution (like the normal distribution, student's t-distribution, or the F-distribution). If our test statistic is sufficiently "weird" in the distribution (typically, if $p < 0.05$), we reject the null. This approach relies on a lot of assumptions.

In contrast, to test the same hypothesis with randomization, we would compute our true (observed) correlation and save that as our test statistic. Then, we would take one of the two variables (say, x), and randomly permute the values. This would break any existing relationship between the x and y. We could then compute the correlation between our original y and our permuted version of x. This value could be added to a null distribution. By repeating this process hundreds or thousands of times, we could build a null distribution that did not depend on any assumptions. Then, we compare our observed value to the values from randomization. If it is sufficiently "weird" in the distribution, we reject the null.

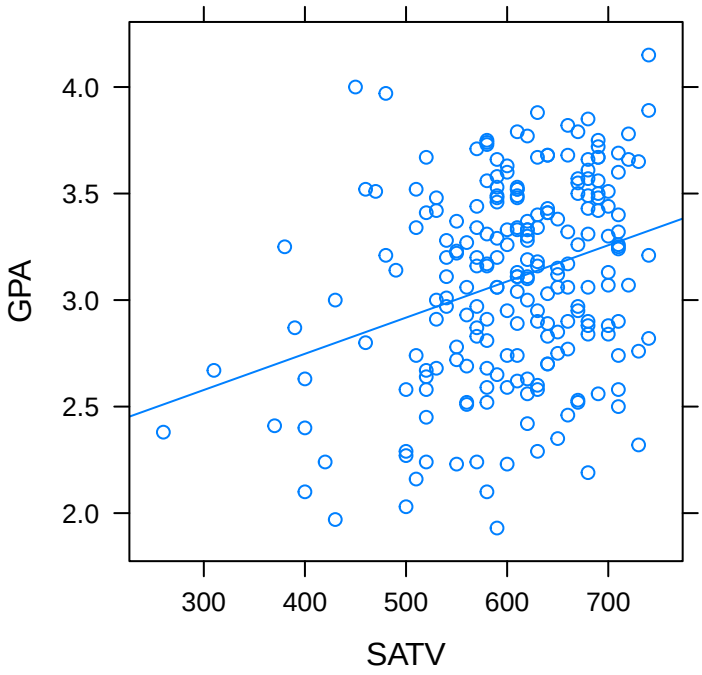
Let's examine this with some code from `R` to make it more concrete. We'll be thinking about some data on college students' first year GPA.

```
library(mosaic)
require(Stat2Data)
data(FirstYearGPA)
head(FirstYearGPA)
```

```
##    GPA HSGPA SATV SATM Male   HU   SS FirstGen White CollegeBound
## 1 3.06  3.83  680  770    1  3.0  9.0        1     1             1
## 2 4.15  4.00  740  720    0  9.0  3.0        0     1             1
## 3 3.41  3.70  640  570    0 16.0 13.0        0     0             1
## 4 3.21  3.51  740  700    0 22.0  0.0        0     1             1
## 5 3.48  3.83  610  610    0 30.5  1.5        0     1             1
## 6 2.95  3.25  600  570    0 18.0  3.0        0     1             1
```

We think there might be a relationship between first year GPA and SAT verbal scores. So, we can make a plot of the relationship and build a model

```
xyplot(GPA~SATV, data=FirstYearGPA, type=c("p", "r"))
```



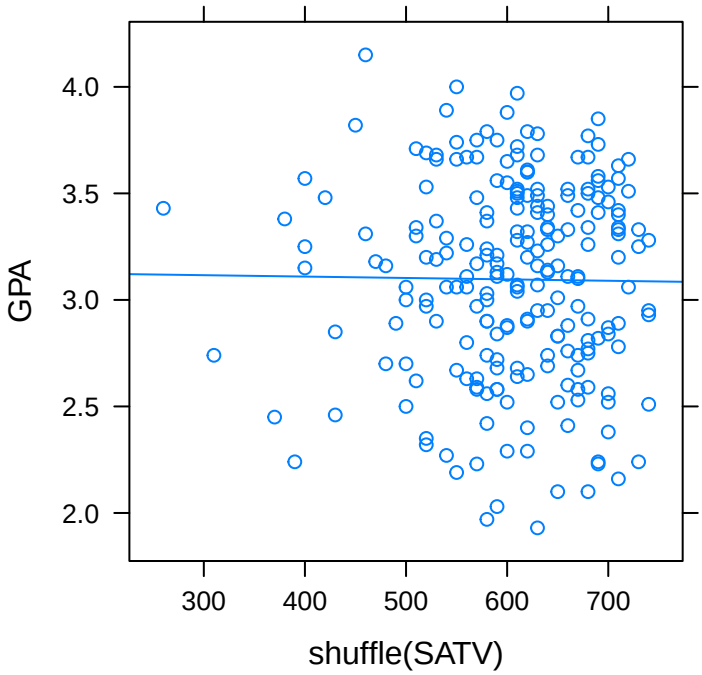

```
m1 = lm(GPA~SATV, data=FirstYearGPA)
summary(m1)
```

```
##
## Call:
## lm(formula = GPA ~ SATV, data = FirstYearGPA)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.14057 -0.32756  0.03134  0.34259  1.16723
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.0684133  0.2204478   9.383  < 2e-16 ***
## SATV        0.0016986  0.0003609   4.706  4.5e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4444 on 217 degrees of freedom
## Multiple R-squared:  0.09261,Adjusted R-squared:  0.08842
## F-statistic: 22.15 on 1 and 217 DF,  p-value: 4.499e-06
```

The equation for the line would be $\widehat{GPA} = 2.07 + 0.002 \cdot SATV$. Using the standard equation for a test statistic ($\frac{\hat{\beta}_1}{SE_{\hat{beta}_1}}$) and the assumption that the data follow a t-distribution, the summary gives us the p-value 4.5×10^{-6}. So, we would reject our null hypothesis and conclude that there is a relationship between GPA and SAT verbal scores.

The randomization approach is not as built in, so we need to use a little more code. First, let's try randomizing the SATV variable and making the same plot as above, so we can see what the relationship would look like if we broke the relationship just once.

```
set.seed(13)
xyplot(GPA~shuffle(SATV), data=FirstYearGPA, type=c("p", "r"))
```

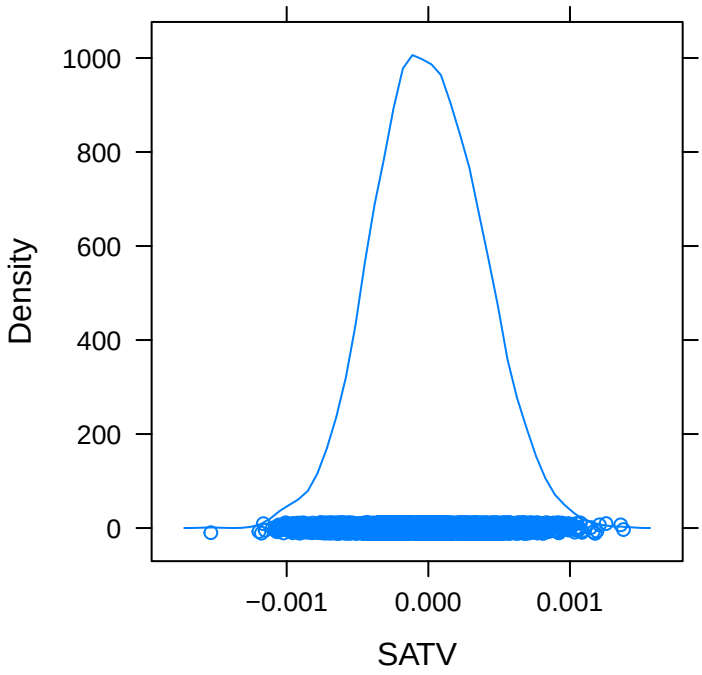


Already, this is a much less extreme visual relationship than we saw before. But we want to do this many times and store the slope coefficient so we can study it over many randomizations.

```
rtest = do(5000) * coef(lm(GPA ~ shuffle(SATV), data=FirstYearGPA))
```

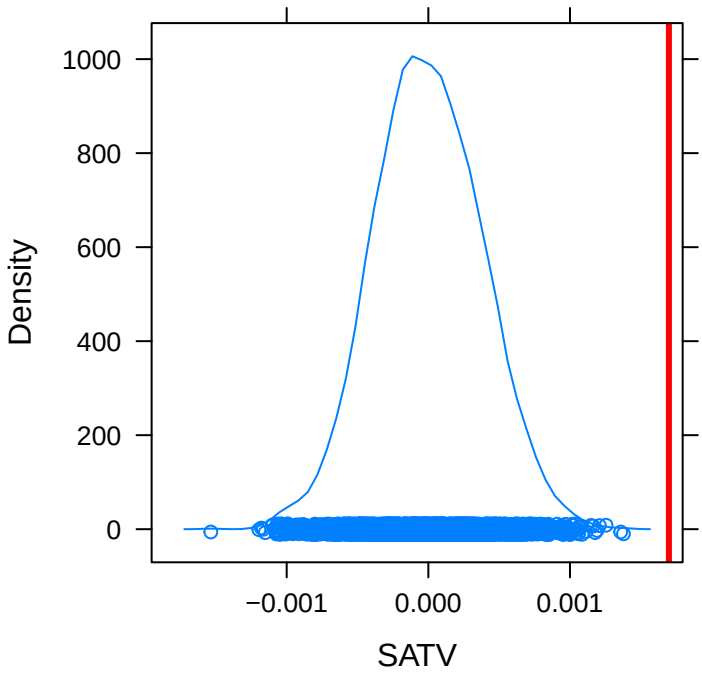
Now we can look at the sampling distribution for the slope

```
p1 = densityplot(~SATV, data=rtest)
p1
```



This is reasonably symmetric, but centered around 0 (as we would expect the null distribution to be). We can use our observed coefficient as a test statistic and see how extreme it is in the context of this distribution

```
ladd(panel.abline(v=coef(m1)["SATV"], col="red", lwd=3), plot=p1)
```



So, our observed data was more extreme than any of the randomized values we found in 5000 samples! This provides evidence we should reject the null hypothesis.

B Bootstrap

[The material in this appendix is based in part on a lab from the multiple regression course I am currently teaching (Baumer and McNamara, 2016).]

The corresponding technique is the bootstrap. Bootstrapping builds a sampling distribution of a parameter (not of the null). It allows us to ask questions about the standard error of the parameter that, once again, we would usually rely on parametric statistics to do.

For example, we might have a coefficient for the slope in a regression model and we want to construct a confidence interval to see what other reasonable slope values we might have observed. Typically, we would use

$$\hat{\beta}_1 \pm SE_{\hat{\beta}_1} \cdot \text{critical value}$$

The critical value is something that depends on our level of confidence and the distribution we are using. If we assume our data follows a t-distribution, we would pick the critical value of t based on the degrees of freedom in our model and our desired level of confidence (for example 80% confidence, 95% confidence, 99% confidence).

Again, the idea of using the bootstrap to simulate data and build up a sampling distribution is precisely to avoid having to make these kind of distributional assumptions.

The bootstrap focuses on taking samples with replacement from existing data to generate new samples. For example, if we had a dataset with 10 values, we might select the following rows in our first bootstrapped sample

```
set.seed(132)
sample(10,replace = TRUE)
```

```
##  [1]  7 10  3  5  5  3  9  2  3  8
```

Notice how rows 5 and 3 appear several times in our new dataset, while rows 1, 4, and 6 do not appear at all. With our new sample, we can compute the value of the $\hat{\beta}_1$ and add that to our sampling distribution. By doing this many times, we will build up a distribution, and again, we can use this to answer the same sorts of questions as we did with the parametric approach above. For example, we could calculate the values that contained 95% of our bootstrapped slope estimates, and use that as a confidence interval. Interestingly, this interval might not be symmetric.

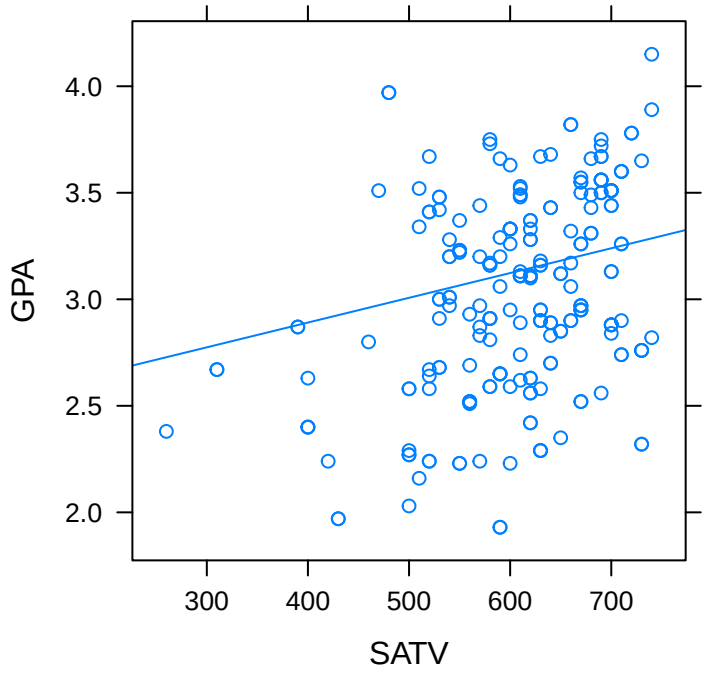
Again, let's use the example of first year GPA and SAT verbal scores. We want to see what some other reasonable values for the slope of the line would be. To do this, we can take a sample with replacement from the data and use that to compute a new line.

```
m2 = lm(GPA~SATV, data=resample(FirstYearGPA))
summary(m2)
```

```
##
```

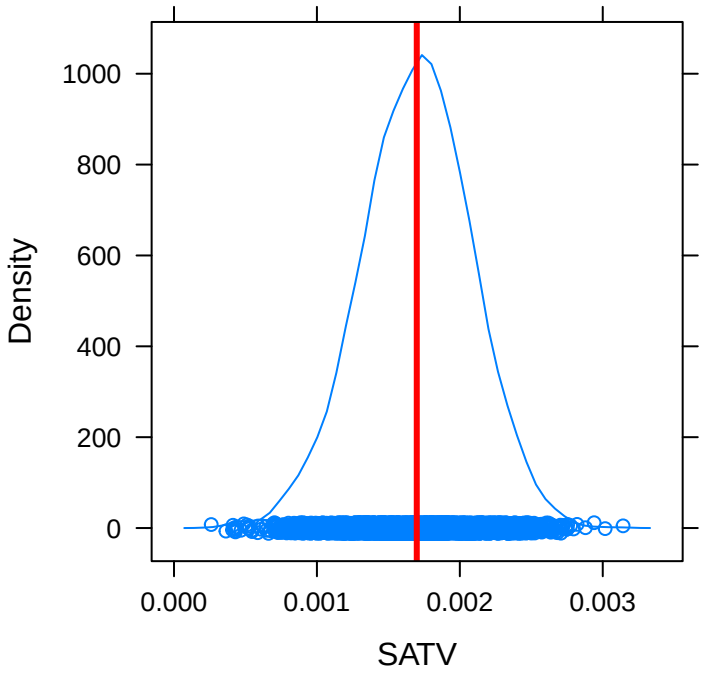
```
## Call:
## lm(formula = GPA ~ SATV, data = resample(FirstYearGPA))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.18162 -0.36189  0.06893  0.36690  1.05109
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.4259443  0.2478924   9.786  < 2e-16 ***
## SATV        0.0011622  0.0004039   2.878  0.00441 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4564 on 217 degrees of freedom
## Multiple R-squared:  0.03676,Adjusted R-squared:  0.03232
## F-statistic: 8.281 on 1 and 217 DF,  p-value: 0.004407


plotModel(m2)
```



Notice that this line is pretty similar to the one we saw in the original data. This makes sense. Again, we can repeat the procedure many times to build up a sampling distribution.

```
bootstrap = do(5000) * coef(lm(GPA~SATV, data=resample(FirstYearGPA)))
p2 = densityplot(~SATV, data=bootstrap)
ladd(panel.abline(v=coef(m1)["SATV"], col="red", lwd=3), plot=p2)
```



This distribution is centered around our original parameter estimate. We can calculate where the middle 95% of the distribution is to create a bootstrapped confidence interval.

```
qdata(~SATV, p=c(0.025, 0.975), data=bootstrap)
```

```
##           quantile     p
## 2.5%  0.0009162567 0.025
## 97.5% 0.0024319270 0.975
```

Since 0 is not contained within the interval, we can reject the null hypothesis.