Title:
Web Application Teaching Tools for Statistics Using R and Shiny

Journal Issue:
Technology Innovations in Statistics Education, 9(1)

Author:
DOI, JIMMY, California Polytechnic State University, San Luis Obispo
POTTER, GAIL, California Polytechnic State University, San Luis Obispo and The EMMES Corporation (Rockville, MD)
WONG, JIMMY, California Polytechnic State University, San Luis Obispo and FDA/Center for Drug Evaluation and Research
ALCARAZ, IRVIN, California Polytechnic State University, San Luis Obispo and OpenX
CHI, PETER, California Polytechnic State University, San Luis Obispo and Ursinus College

Publication Date:
2016

Acknowledgements:
The authors wish to thank two anonymous referees for their incredibly helpful suggestions on this paper. We are grateful to Dean Phil Bailey of the Cal Poly College of Science and Math and to Roger Oberg of RStudio for their support in helping to establish the Shiny Server at Cal Poly. We are also indebted to the staff of Cal Poly College of Science and Math Computing Support, especially to Brian Zelenke and Chip Dupre, for setting up and maintaining our Shiny Server. Finally we thank Mark Schilling and Ann Watkins for their feedback on our apps and on a draft of this paper.

Author Bio:
Department of StatisticsProfessor

Department of StatisticsAssistant Professor

Department of StatisticsUndergratuate Student

Department of StatisticsUndergratuate Student

Department of StatisticsAssistant Professor

Keywords:
Statistics Education, Introductory Statistics, Technology Implementation, Web-Based Applications, R, Shiny

Abstract:
Technology plays a critical role in supporting statistics education, and student comprehension is improved when simulations accompanied by dynamic visualizations are employed. Many web-based teaching tool applets programmed in Java/Javascript are publicly available (e.g., www.rossmanchance.com, www.socr.ucla.edu). These provide a user-friendly interface which is accessible and appealing to students in introductory statistics courses. However, not all statistics educators are fluent in Java/Javascript and may not be able to tailor these apps or develop their own. Shiny, a web application framework for R created by RStudio, facilitates applet development for educators who are familiar with R. We illustrate the utility, convenience, and versatility of Shiny through our collection of 17 freely available apps covering a range of topics and levels (found at www.statistics.calpoly.edu/shiny). Our Shiny source code is publicly available so that anyone may tailor our apps as desired. We provide feedback on how our apps have been used in statistics classes including some challenges that were encountered. We also discuss feasibility on building, launching, and deploying Shiny apps. A brief tutorial on installing and using Shiny is provided in the appendix. Some teaching materials based on our Shiny apps are also included in the appendix.

Web Application Teaching Tools for Statistics Using R and Shiny

1. Introduction

Computer technology can greatly enhance statistics education by illustrating fundamental concepts such as randomness, sampling, and variability through computer simulations and visualizations (Chance & Rossman, 2006; Chance, Ben-Zvi, Garfield & Medina, 2007; delMas, Garfield & Chance, 1999; Garfield & Ben-Zvi, 2008; Harraway, 2012; McDaniel & Green, 2012; Pratt, Davies & Connor, 2011; Shaltayev, Hodges & Hasbrouck, 2010; Zieffler, Park, Garfield, delMas & Bjornsdottir, 2012). Studies have shown that such technological enhancements to statistics instruction significantly improve student performance, as measured by pretest and posttest analyses (Hagtvedt, Jones & Jones, 2007; Lane & Tang, 2000; Lunsford, Rowell & Goodson-Espy, 2006). The use of technology to introduce and reinforce these essential concepts has also been promoted in the *Guidelines for Assessment and Instruction in Statistics Education* (GAISE) for introductory statistics at the college level (Franklin & Garfield, 2006; ASA, 2005a) as well as for teaching statistics at the secondary and primary levels (ASA, 2005b).

Many instructors use web-based applications, mainly in the form of Java/Javascript applets, to facilitate technology-assisted instruction. There are many outstanding applet teaching tools found online (e.g., the *Rossman/Chance Applet Collection*; Rossman & Chance, 2004) and the *Statistics Online Computational Resource* (www.socr.ucla.edu)). There are many advantages in using applet teaching tools as they can be interactive, dynamic, user-friendly, visually appealing, and publicly accessible via the web. Students can use these applications to better understand statistical concepts via a point-and-click user-interface that does not require any code compilation by the user.

Despite the large collection of existing applet teaching tools found on the web, eventually an instructor can come across a problem in finding an existing applet to perfectly suit his/her needs. If an existing applet is close to the desired functionality, one can opt to customize the applet. However, such customization requires access to all of the original source code and in many situations this code is not easily available. Moreover, even if it is available, customization still requires fluency in the source code language. Additionally, if the desired functionality of the teaching tool application is novel (e.g., based on newly proposed research), existing applets would most likely be unsuitable.

In these situations the instructor is left to consider building his/her own teaching tool applet. But this is not a trivial task. This can require knowledge in Java, Javascript, HTML, CSS, and PHP languages. The time required to learn the necessary languages can be a sufficient obstacle in keeping an instructor from creating applets.

An alternative method to create web-based teaching tool applications is provided by Shiny (Chang, Cheng, Allaire, Xie & McPherson, 2015), a recent technology created by RStudio. Shiny is a web application framework for R (R Core Team, 2015) that only requires knowledge in the R programming language. It is not uncommon for instructors to build their own teaching tools via scripts written in R and, as we will show, it is not difficult to convert existing R scripts into Shiny applications, known simply as 'Shiny apps'. With Shiny, instructors can build a teaching tool that is interactive, dynamic, user-friendly, visually appealing, and, with similar functionality to Java/Javascript applets; the only requirement is some familiarity in R.

This paper illustrates the versatility of Shiny through a collection of web application teaching tools that we have developed via the Shiny package in R. We start with a motivating example in Section 2 to show what Shiny can do and how it can be used in the classroom. In Section 3 we describe our *Shiny App Teaching Tools Collection* (total of 18 apps). We provide links to our Shiny apps site and our GitHub Gist site containing all Shiny source code for each app so that anyone may tailor our apps as desired. In Section 4 we provide feedback on how apps from our collection have been used in the classroom. In Section 5 we discuss the feasibility of building, launching, deploying, and accessing Shiny apps. Section 6 contains helpful Shiny app resources and Section 7 includes concluding remarks. We also provide a brief tutorial on how to install and get started in Shiny in Appendix A. Some teaching materials based on our Shiny apps are included in Appendix B.

2. Motivating Example

One popular class activity to help students understand chance behavior is to observe the runs of consecutive heads or tails in a sequence of coin flips. When asked to write down a simulated sequence of 100 tosses of a fair coin, most students are hesitant to create runs of heads or tails exceeding four. Students are often surprised to find that the longest run of heads or tails turns out to be much higher.

It is far too time consuming (and boring!) to manually flip a coin hundreds of times and record the outcomes for a class demonstration. It is far more reasonable to use a computer to simulate this task. Let's consider how this might be done by examining a traditional way to use R to how it can be done using Shiny.

Several years ago one of the co-authors was teaching an introductory statistics course for liberal arts majors and for a class demonstration wanted to simulate hundreds of coin tosses and display the runs of heads or tails of a particular length. At the time no Java/Javascript applets or any other simulation resources were found online to accomplish these tasks so the instructor created two R functions for the demonstration. Both functions can be found at our Shiny apps website (www.statistics.calpoly.edu/shiny). The first function, called

`flip.gen()`, simulates the outcomes of a fair coin flipped a given number of times and also keeps track of the run length of heads or tails. As an example, here are the first 9 outcomes from 200 tosses of a fair coin generated by `flip.gen(200)`:

`coin`	1	0	0	1	1	1	1	0	0
`run`	1	2	2	4	4	4	4	2	2

Here, `coin` represents the outcome ($1 = \text{heads}, 0 = \text{tails}$) and `run` represents the length of the corresponding run.

The second function, called `plot.flips()`, graphically displays the outcomes generated by `flip.gen()` and marks any runs having a length of at least some specified value. The length of the longest observed run is also displayed. As an example, suppose the outcomes we previously generated with `flip.gen(200)` were stored in `results`. If we wanted to generate a plot of those outcomes and highlight run lengths of 4 or more, we would submit the function `plot.flips(4,results)`[1]. This plot is shown in Figure 1.

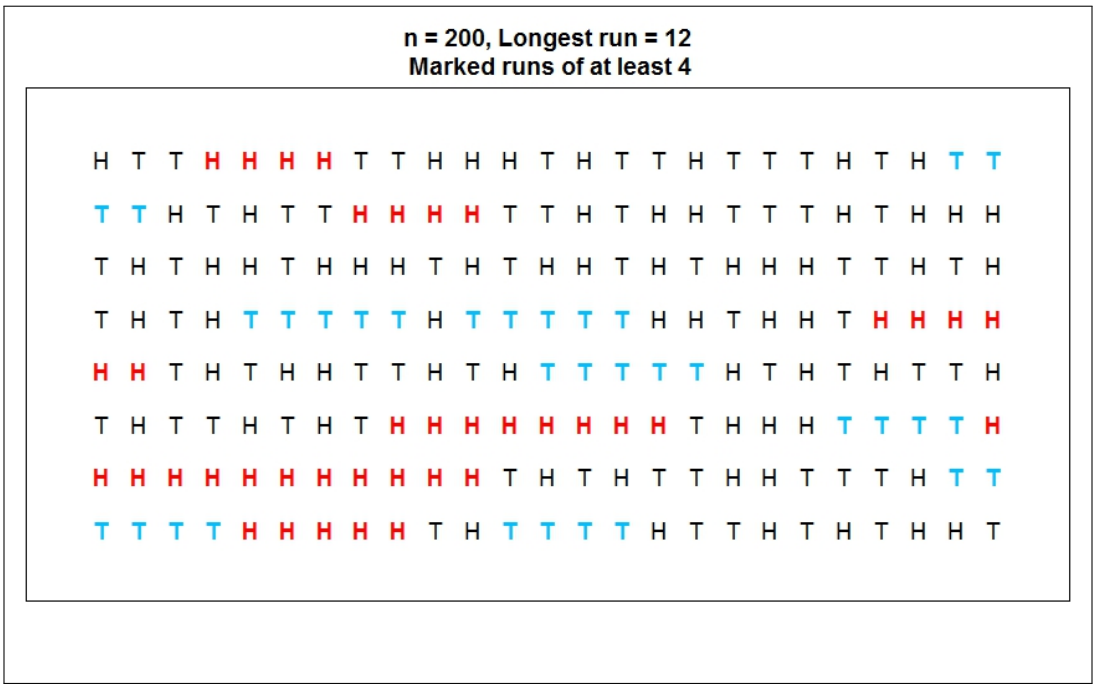

Figure 1: Output from `plot.flips()` based on 200 simulated tosses of a fair coin. Run lengths of four or more are marked in color.

[1]The function `plot.flips()` actually accepts 4 total arguments but we only show 2 here for the sake of simplicity.

For the in-class simulation the two R functions were used in the following manner:

(1) Display 20 trials: `plot.flips(4,flip.gen(20))`

(2) Repeat to observe run length variability: `plot.flips(4,flip.gen(20))`, `plot.flips(4,flip.gen(20))`, ...

(3) For a given set of randomized outcomes, update display with new minimum run length parameter: `plot.flips(5,results)`, `plot.flips(6,results)`, ...

- Step above assumes the given outcomes are stored in `results`

(4) Repeat previous steps using different number of trials (50, 100, 200, 400)

Because the audience needed to wait for the instructor to submit new commands at the R console before the display was updated, the presentation became rather choppy. Submitting commands at the console can be a nuisance and, especially for introductory statistics students, the delays can detract from the content of the presentation.

This scenario is what could be described as a *traditional way* to use R functions in the classroom: (1) load/submit functions from the R console, (2) draw attention to the newly generated display/output, (3) update the script by changing a function parameter to another value, and repeat the process.

As opposed to the choppy nature of this traditional approach, a corresponding Shiny app (using interactive web elements) can allow for a much more fluid presentation. The *Longest Run of Heads or Tails* Shiny app is a web application that produces the same results as shown in Figure 1 but allows for a more dynamic user experience. The app uses the same two R functions previously described. In the app a graphical slider object is used to select the total number of trials to simulate. Another slider object controls the minimum run length which determines what particular runs to mark in color. As shown in Figure 2 both of these slider objects appear in the left panel of the app.[2]

[2] Note that the app features a third slider object that controls font size of the displayed outcomes. Also a checkbox object allows the user the option to display the predicted approximate length of the longest run and an approximate 95% prediction interval for the length of the longest run. Details on these two estimators can be found in Schilling (1990). See Schilling (2012) for a more recent and related article.

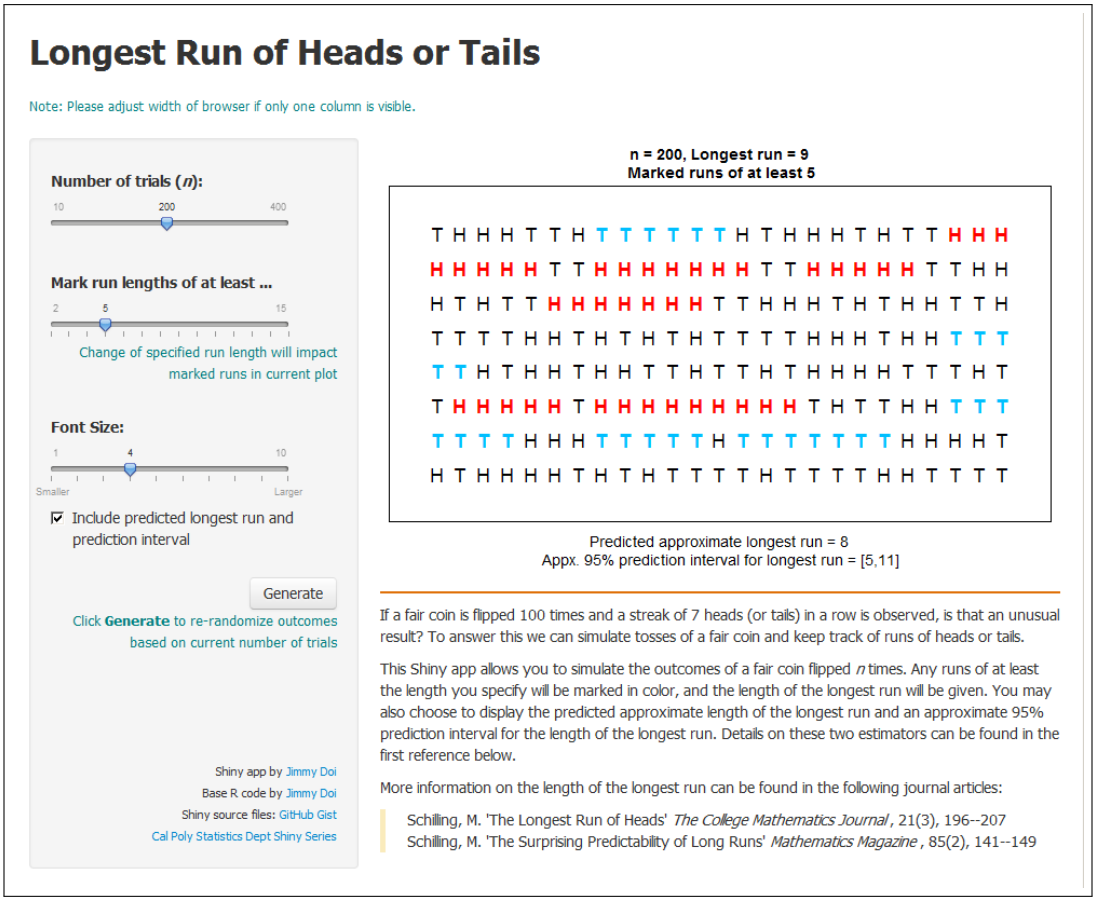

Figure 2: Longest Run of Heads or Tails app.

Here are some features of this Shiny app:

- An adjustment of the trials slider *instantly* updates the display with a re-randomization of outcomes. One can then quickly show simulations corresponding to different trials (e.g., 20, 50, 100, 200, 400).

- An adjustment of the run length slider *instantly* updates marked runs **without** forcing a re-randomization[3]. Thus one can conveniently highlight various run lengths using the same outcomes in the current plot.

- By clicking on the "Generate" button one can *instantly* re-randomize outcomes based on the currently selected number of trials.

[3] This feature is facilitated by what is known as a *reactive expression* in Shiny. For more on reactive programming see Section A.3 of Appendix A.

Presenting via the Shiny app offers some advantages over the traditional presentation method. When using this app the R console need not ever be shown as R works completely in the background and interactively with Shiny. Thus the awkward pauses caused by code submission at the command line from the traditional method are no longer an issue. All adjustments in the app are done by moving sliders or clicking buttons and corresponding updates to the output plot are virtually instantaneous. This leads to a much more fluid and dynamic presentation.

Another important advantage of the Shiny app is that it can be deployed on a server so that it can be accessed online via a web browser. Thus, students can experiment with the app on their own without having to know anything about R programming. All they would need to know is how to launch a web browser.

Finally, it is important to note that this example was spawned by the instructor's need for a demonstration that did not currently exist online. Shiny provided a framework where existing R code could be adapted to produce the dynamic web-based application as shown. As such, Shiny can facilitate the development of new teaching tools in a very feasible manner. This is especially helpful when teaching concepts that are not in the standard curriculum or are based on recent research.

The app described here offers only a limited glimpse into the possibilities with Shiny. In the next section we introduce our collection of Shiny app teaching tools covering a variety of statistical topics ranging from introductory to advanced levels. We hope this collection will provide a greater understanding how Shiny can be used as a teaching tool and inspire instructors to generate their own apps.

3. Shiny App Teaching Tools Collection

At the time of manuscript submission we had completed a total of 18 Shiny app teaching tools. The apps can be accessed from our Shiny apps website at www.statistics.calpoly.edu/shiny. All relevant files required to create each app can be accessed and downloaded from our GitHub Gist directory at gist.github.com/calpolystat. Our complete list of apps is shown in Table 1 (ordered by topic). To our knowledge, most of our apps are unique in that there are no equivalent Java/Javascript applets or other web applications found online.

As can be seen from Table 1 there is a wide variety of apps to select from making them applicable for a broad audience. The *Longest Run of Heads or Tails app* (first described in Section 2) is based on the simple concept of coin tossing which even elementary school students should find understandable. The *Random Variable Generation* app produces random deviates through the probability integral transform and the accept-reject algorithm, topics that are well suited for mathematical statistics students. In addition, an example

of virtually every Shiny widget and application layout can be found within our collection of apps, making it quite broad with regard to not only subject matter but also Shiny app components/features. The development of additional apps will be an ongoing effort and we will update our Shiny website and GitHub Gist directory as new apps are completed.

Topic	Shiny App Name
Inference	• *Benford's Law: Sequences* • *Benford's Law: Current Demographic Data* • *Length/Coverage Optimal Confidence Intervals* • *Performance of the Wilcoxon-Mann-Whitney vs. t-test* • *t-test with diagnostics* • *Testing Violation of the Constant Variance Condition for ANOVA*
Probability and Randomness	• *Chaos Game: Two Dimensions* • *Chaos Game: Three Dimensions* • *Longest Run of Heads or Tails* • *Gambler's Ruin*
Distribution Theory and Estimation	• *Maximum Likelihood Estimation for the Binomial Distribution* • *Probability Distribution Viewer* • *Random Variable Generation* • *Sampling Distributions of Various Statistics*
Regression	• *Correlation and Regression Game* • *Multiple Regression Visualization*
Special Topics	• *Heaped Distribution Estimation* • *Hierarchical Models*

Table 1: Shiny apps listed by topic.

To offer a glimpse into the variety of apps we have developed we will now provide a brief summary of four selected apps.

3.1. Testing Violation of the Constant Variance Condition for ANOVA

The ANOVA F-test, used to test for a difference among group means, requires the conditions of normality (or large sample size), independence, and constant variance. A common rule of thumb for the constant variance condition is that the ratio of the largest to smallest standard deviation is less than or equal to two. This app implements a user-guided simulation study to assess the consequences of non-constant variance on the Type I error rate of the ANOVA F-test. The app enables the user to visualize data with different standard deviations, reinforces the concepts of sampling distribution, null distribution, and Type I error, and allows the user to uncover a rule of thumb for the constant variance condition.

A screenshot of this app is shown in Figure 3.

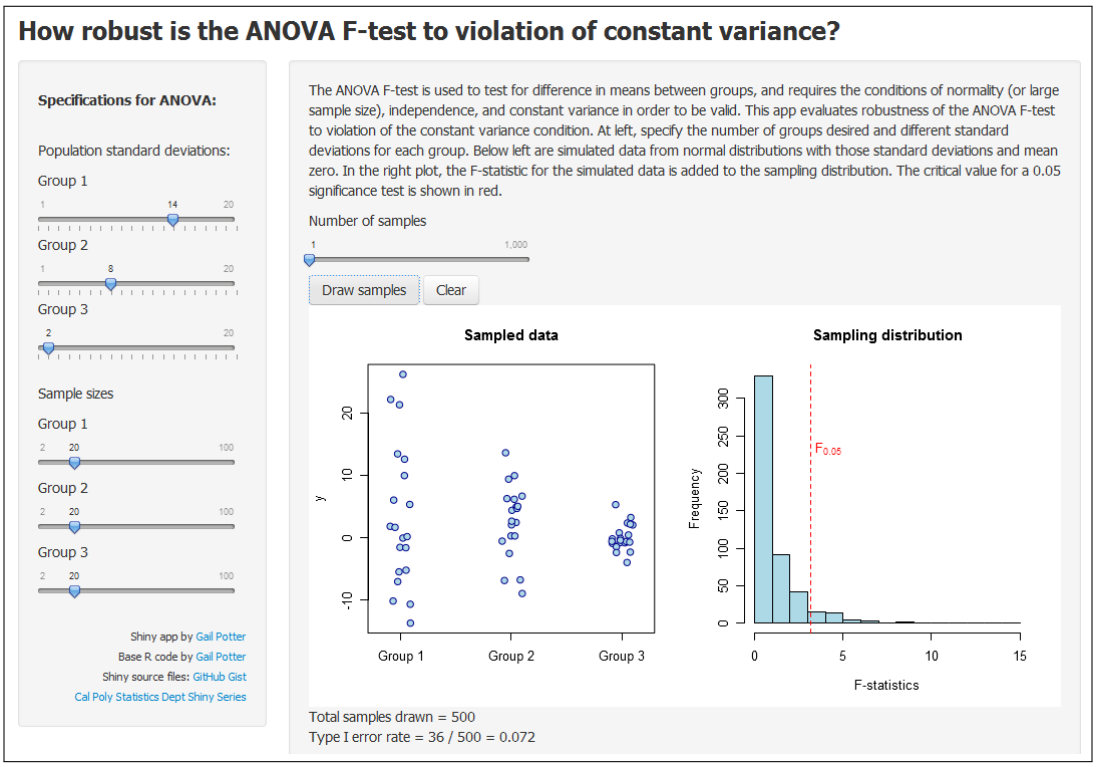

Figure 3: Testing Violation of the Constant Variance Condition for ANOVA app.

In the app the user specifies standard deviations for three hypothetical populations and sample sizes to be drawn from each of the populations. When the user presses the "Draw samples" button, data will be simulated from normal distributions (with mean zero, specified standard deviations) and displayed in dot plots in the left graph. The ANOVA F-statistic for the simulated data is plotted in the graph at right, and the critical value for a 0.05-level hypothesis test is shown in red. As more samples are drawn (with the option to draw up to 1,000 samples at a time), more F-statistics are plotted in the sampling distribution on the right. The Type I error rate is estimated as the proportion of samples for which the null hypothesis was rejected, and this estimate is displayed in the app. Below the graphs (not included in the accompanying screenshot) is guidance for a suggested series of simulation studies allowing the user to compare different specifications systematically and uncover the rule of thumb for the constant variance condition.

3.2. Performance of Wilcoxon-Mann-Whitney Test versus t-test

The goal of this app is to compare the performance of a nonparametric to a parametric test for the difference in two population means. Specifically, performance is measured in the app either by Type I error rate or power, and the two respective tests for comparison are the Wilcoxon-Mann-Whitney (WMW) test and the two-sample t-test. For the respective test conditions to be satisfied, the two-sample t-test requires either the two population distributions to be normal or for the samples to be sufficiently large, while the WMW test requires the two population distributions to have the same shape. Users have the option to produce different scenarios and conclude the better test either through a lower Type I error rate (if the two population means are the same) or a higher power (if they are not).

A screenshot of this app is shown in Figure 4.

Figure 4: Performance of Wilcoxon-Mann-Whitney Test versus t-test app.

When users first launch the app, they are presented with the goal of the study. Then, a game demonstrates the challenge of identifying the population distributions of sample data. Following the first two introductory tabs, users can proceed to comparing performance. They have the option to choose a tab corresponding to their choice of the population distributions. Within each tab, either a single comparison or comparisons over a range can be conducted. The settings available for users to adjust are sample sizes, population means, significance level, number of simulations, and range of comparison values. In addition, visualizations are implemented to communicate results to users. For a single comparison, the outputs are distributions of the test statistics and gauges. For comparisons over a range, the output illustrates the performance of the two tests in each comparison.

3.3. Probability Distribution Viewer

Probability distributions, p-values, and percentiles are fundamental topics taught to introductory statistics students. Often, students encounter these topics through static images in textbooks, but frequently do not have access to a dynamic and interactive tool they can use for exploration. For example, in the case of p-values, introductory students are frequently shown how to use tables to obtain a range of possible p-values associated with their test statistic along with a guiding image, but this is often difficult for students to understand. The goal of this app is to provide the student with an intuitive, simple, and comprehensive visualization of the three aforementioned topics. At the moment, many continuous distributions (Beta, Cauchy, Chi-Squared, Exponential, F, Gamma, Logistic, Log Normal, Normal, Student's t, Uniform, and Weibull) are available in the Shiny app. Support for discrete distributions may be added in the future.

A screenshot of this app is shown in Figure 5.

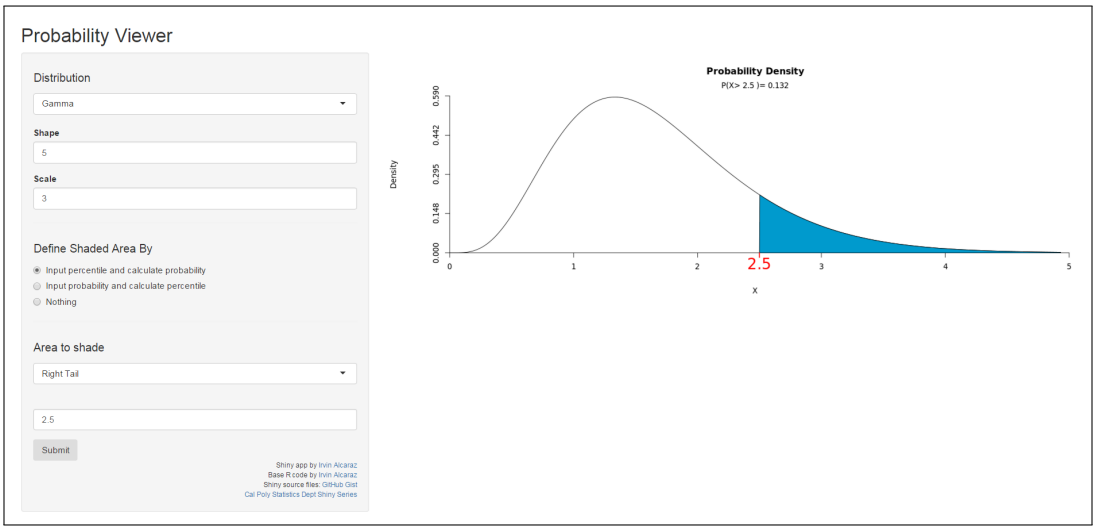

Figure 5: Probability Distribution Viewer app.

When the app first renders, by default the user is shown the standard normal distribution. Users may vary the both the distribution and corresponding parameters to choose the distribution of their choice from the options at the top under "Distribution". This enables the student to see how the shape of their selected distribution changes as parameter values change. The probability viewer app easily allows the student to select between two types of inputs: probabilities and percentiles. The student can also select between different shaded tail visualizations for the specified percentiles or probabilities. After inputting all required values, a graph appears with the appropriate distribution, the percentile, the probability, and the appropriate shading.

3.4. Gambler's Ruin

The Gambler's Ruin is a well-known problem that can be used to illustrate a variety of probability concepts (e.g., conditional probability, Markov chains, and for simply visualizing a stochastic process). Two players are playing a game against each other, betting the same amount on each turn (here, we use \$1). On each turn of the game, Player A has a fixed probability p of winning \$1 from Player B, where $0 < p < 1$. The probability that Player B will win \$1 from Player A is $1 - p$. Player A and Player B start with some initial fortunes (which may or may not be equal to each other), and the game continues until one player has all of the money.

A screenshot of this app is shown in Figure 6.

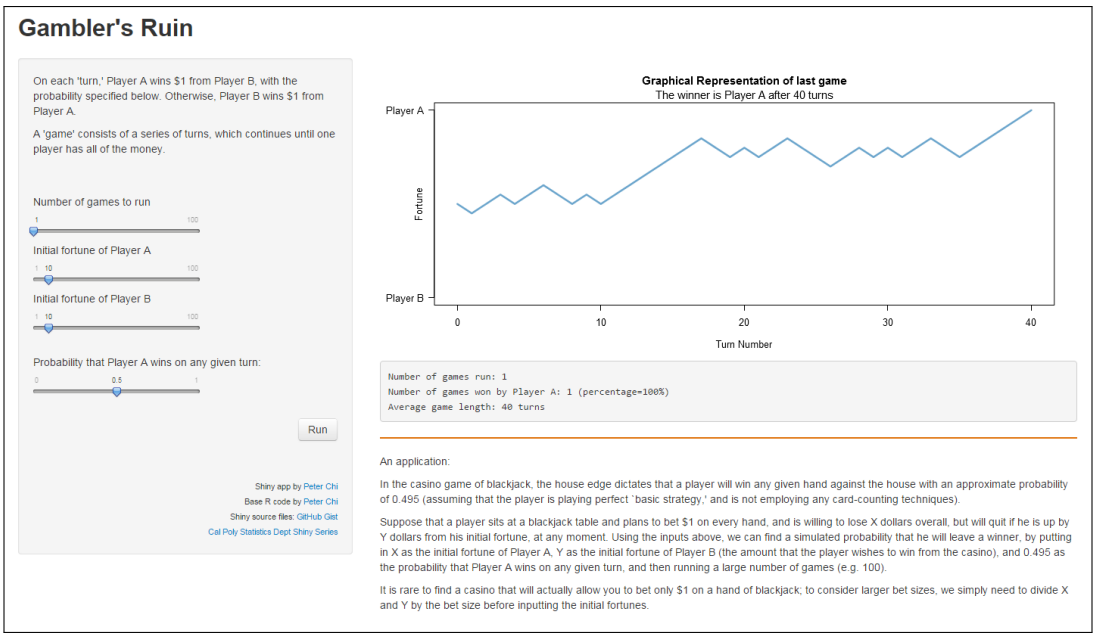

Figure 6: Gambler's Ruin app.

This app shows a graphical representation of one iteration of the Gambler's Ruin, and also can simulate many runs under a variety of settings that may be manipulated to obtain simulated estimates of the average length of a game and the probability that Player A will win under those settings. In a mathematical statistics class, the simulated estimates from this app could be used to corroborate analytic solutions.

4. Feedback on Using Shiny Apps in the Classroom

In this section we share feedback from various instructors (not all of whom are from our university) who have used or plan to use our Shiny apps in class. We also provide access to some of the related teaching materials (e.g., homework assignments using the apps). We hope this information will help instructors see how Shiny can be used in their own classes.

4.1. Longest Run of Heads or Tails App

The following is based on feedback provided by two faculty at another university who used the *Longest Run* app in an introductory probability and statistics course they co-taught for liberal arts majors.

In the instructors' lesson titled *Randomness, Runs, and Coincidences*, students were first shown two sets of sequences of 200 coin tosses: one sequence based on a fair coin toss simulation and the other was fake. The fake sequence was constructed so that there was never a run of more than four heads or four tails. The simulated sequence of a fair coin contained multiple runs of six and a maximum run of seven. The students were asked to determine which sequence was faked and most chose incorrectly.

The *Longest Run* app was then displayed in class to show the simulated sequence that most students dismissed as being 'fake' was not a fluke occurrence. Given that the class did not meet in a computing lab, students were invited to access the app via a web browser on their smartphones to see the results for themselves. Many complied and were surprised to see the unexpected results.

One of the instructors provided the following impression about the app:

> "I felt that the app was quite effective in reinforcing a main message from the lesson, which is that 'surprising' patterns (such as long runs) are in fact quite common in sequences of outcomes of independent events."

The instructor went on to say:

> "I'm afraid I didn't make the use of the app very student centered, as it was not built into the lesson ... however, we absolutely could and I think should incorporate the app into the written lesson and the homework that is at the end of it. It is so easy to use, so transparent in what it shows, and does exactly what we want to do in that part of the lesson."

4.2. Testing Violation of the Constant Variance Condition for ANOVA App

This app was used in a calculus-based second course in statistics, mainly taken by statistics and math majors. The app was first demonstrated in class to illustrate what different simulated data sets look like when they have the same or different standard deviations as well as the sampling distribution of the F-statistic. Students then explored simulations with the app in a homework problem. See Section B.1 of Appendix B for more details about this assignment.

This exercise helped students solidify their understanding of variability and sampling distributions while students explored the robustness of the F-test to the constant variance condition. The homework highlighted the fact that the violation of this condition may be small or large, with larger violations having a larger impact on test validity. The app also allowed students to uncover the importance of sample size in robustness of the test.

The instructor who used this app felt that the students achieved a better understanding of power and Type I error when compared to students from other classes where the app was not used. The instructor also felt the app helped the students better understand the constant variance condition of ANOVA.

4.3. Sampling Distributions of Various Statistics App

We describe the use of this app in two different classes: introductory statistics for engineers (taught at our university) and introductory biostatistics (taught at a liberal arts college).

Course: Introductory Statistics for Engineers

This app was used in a calculus-based introductory course in statistics for engineering majors. The app was used to demonstrate the concept of a sampling distribution. During lecture, each student manually sampled a data set of three American incomes (from papers in an envelope) and computed the mean income. A histogram of all the mean incomes was created on the board. Next, the app was used to draw a larger number of samples, thereby generalizing the histogram and better approximating the sampling distribution. The entire exercise was repeated with samples of size 10, again using the app to generalize the sampling distribution. Finally, the app alone was used to illustrate the sampling distribution with samples of size 1000. After this demonstration, students were assigned a homework problem that required the use of the app. See Section B.2 of Appendix B for more details about this assignment. The instructor who used this app felt the students had a better understanding of sampling distributions in general compared to students who had not used the app.

Course: Introductory Biostatistics

This app was also used in a second semester course of a non-calculus-based introductory biostatistics sequence. A laboratory exercise was built around this app, with the aim of demonstrating that the sample size requirement for the Central Limit Theorem depends on the underlying distribution of the data generating process.

Assuming the underlying distribution has finite variance, some textbooks suggest a rule of thumb for the minimum size required for the sample mean to attain normality is 30. With this lab students explored, by a qualitative examination of simulated data, what actual sample sizes appear to lead to normal distributions of the sample mean for different underlying distributions. See Section B.3 of Appendix B for more details about this laboratory exercise. The instructor who used this app felt the students had a firm understanding of the fact that the minimum sample size of 30 rule of thumb fails in many situations.

4.4. Random Variable Generation App

One of the co-authors plans to use the *Random Variable Generation* app for a class to be taught later in the year. The following is the proposed use for the app.

Random variable generation is an advanced topic in Probability/Mathematical Statistics. As such, it can be very informative to visualize the process through a step-by-step simulation, and students can gain insight that they might not have otherwise. For example, a student may be very capable of proving mathematically that the Probability Integral Transform does in fact produce random variates from a desired distribution, but may still have little intuition behind its mechanics. Our app provides exactly this insight.

The app could work in combination with a standard homework problem of showing a proof of the Probability Integral Transform. The student could then use the app to produce random variates from a few different distributions that the app allows for and comment on its properties. In this way, a student gains not only a theoretical understanding of the Probability Integral Transform, but also a practical understanding of it.

4.5. Challenges and Limitations

To allow students access to our Shiny apps we needed to deploy all apps on a server. When we had completed about ten apps we deployed them at the cloud-based server ShinyApps.io. This was a convenient option as our account was free and the account limitations were initially not too restrictive. However the restrictions for the free account at ShinyApps.io eventually changed. One change was the total hours of app usage (across all apps) per month was lowered to 25 hours. The previous setting was such that our students had no

problems accessing our apps and using them as much as they desired. But when the 25 hours restriction was set it did not take long for our students to use up this allotted time within a month. Another important change was the maximum number of deployed apps per free account was lowered to five. With these changes we realized we could not solely rely on ShinyApps.io for app deployment. This led to our decision to install Shiny Server Pro on a Linux server on our campus (see Section 5.2 for more details).

As far as the Shiny apps themselves, though they can mimic the behavior of certain Java/Javascript applets, there are some disadvantages to note. First, applets can perform more efficiently and faster than corresponding Shiny apps. Second, Shiny does not currently offer the same dynamic animation capabilities that applets can provide. One example is the *Least Squares Regression* applet at www.rossmanchance.com/applets/RegShuffle.htm. In this applet the user can overlay a line on the scatterplot and adjust the slope and intercept very fluidly by manipulating anchor points on the line with the mouse. To our knowledge there is no way to emulate this same feature in Shiny.

5. Feasibility

In this section we discuss the feasibility of building, launching, deploying, and accessing Shiny apps.

5.1. Building Shiny Apps

Shiny allows the instructor to build web-based teaching tool applications that can have very similar functionality to Java/Javascript applets. Unlike applets that may require knowledge in Java, JavaScript, PHP, CSS, and HTML, Shiny only requires familiarity with the R programming language.

For instructors who already have R scripts used for in-class demonstration purposes, the process to convert the code into a Shiny app can be straightforward. As described in Section A.2 of Appendix A, each Shiny app is usually comprised of two components: a user-interface script (`ui.r`) and a server script (`server.r`)[4]. It is usually the case that `server.r` is mainly comprised of computational components found in an R script. So if one already has a working R script, much of the work to build the corresponding Shiny app has already been done. In Section A.2 we show how a simple R script is converted to a corresponding Shiny app.

[4]As of version 0.10.2, Shiny allows for single-file applications where the components of `server.r` and `ui.r` can be stored in one file called `app.r`.

Of course if one wants to build a complex app with multiple user interface components then the required Shiny code can also be quite complex. Our recommendation for anyone learning Shiny for the first time is to start with a very simple project and slowly build on that foundation.

Our Experience

The initial step for each of the co-authors in learning Shiny was to go through the Shiny Tutorial provided by RStudio (shiny.rstudio.com/tutorial). The seven lessons of the tutorial took about two hours to complete. This tutorial provided a solid foundation to create apps.

One of the first apps we created was the *Longest Run of Heads or Tails* app described in Section 2. The co-author who created this app had previously built the R script containing the required functions to perform the coin toss simulations and to generate the corresponding plot as shown in Figure 1. Those functions were copied (without modification) into the app's server script `server.r` and only a handful of additional Shiny functions were required to complete this file. As a result this step required relatively little time.

The step that required the most time in building this app was the creation of the corresponding user interface script `ui.r`. However this is not to suggest that this is necessarily a difficult process. The layout and appearance of the app is defined in `ui.r`. As shown in Figure 2 the layout of the *Longest Run* app is not complicated. There is a title, a left panel containing text and various widgets (three sliders, one checkbox, one button), and a right panel containing a plot and some exposition. All of these components must be defined in `ui.r`. We slowly built this file in stages by adding a component, test launching the app, and then repeating the process to eventually include all necessary components. The *Longest Run* app required approximately seven hours of work in total. Some projects in our Shiny apps teaching tool collection required less time and some required more. The time commitment depends on the computational complexity of the app and how elaborate the desired user interface components are.

Overall we believe it is quite feasible for anyone with some familiarity with R to be able to create Shiny apps. One does not need to be at the level of a master R developer. In fact, two of the co-authors were undergraduate statistics students in our department as we developed our collection of Shiny apps. We recommend new users to start with the RStudio tutorial and then slowly build the first app in stages. Many Shiny app teaching tools found online, including those at our Shiny apps website, provide complete access to corresponding `server.r` and `ui.r` files. As such it is easy to learn from and customize an existing app.

5.2. Launching, Deploying, and Accessing Shiny Apps

Launching Apps for Demonstrations

Once a Shiny app has been completed, showing the app for a presentation is a very simple process. We encountered no problems in demonstrating Shiny apps in the classroom. When the app is launched it will appear either in a web browser or in an app window. If the app files are stored on a local machine, by default a Shiny app is launched on a randomly selected port of the computer and so no active internet connection is required in this mode. More details on launching a Shiny app are found in Section A.4 of Appendix A.

Deploying Apps on a Server

Shiny apps can be made publicly available on the internet through app deployment on a server (see Section A.5 of Appendix A for more details on deployment). With a deployed app students will have the opportunity to experiment with the app at their leisure and thereby gain a better understanding of the relevant material. Their exposure would not be limited to a brief demonstration of the app shown in class.

One way to deploy an app is through RStudio's server ShinyApps.io where the user can choose one of several accounts based on a tiered pricing structure. ShinyApps.io offers a free account option however it is extremely limited (maximum of five apps deployed and 25 hours of total app usage per month). The fee based accounts offer greater flexibility however they can be cost prohibitive.

Instructors at an institution with access to a Linux server could consider installing a local Shiny server (Open Source or Professional Edition). The Open Source Edition is free, while the Professional Edition requires an annual license fee (but see *Academic Pricing Policy* below). Users can download binary installers for Ubuntu/Debian and Red Hat/CentOS.

Academic Pricing Policy

According to the RStudio Academic Pricing website (www.rstudio.com/pricing/academic-pricing) qualified academic institutions can access RStudio commercial products (such as Shiny Server Professional Edition) at a discounted rate. If the purpose of the commercial product is for research then the institution can obtain the software at a 50% discount. If the purpose is for teaching then the institution can obtain the software for **free** (course syllabus must be submitted to RStudio as verification).

At our university we were able to obtain Shiny Server Pro for free through the Academic Pricing Policy. Our IT staff installed the software on a Linux server housed in our college. The RStudio site claims that "installation is straightforward for anyone familiar with Linux" and our IT staff confirmed this to be the case thanks to the very helpful Shiny Server Pro installation guide (rstudio.github.io/shiny-server/latest).

For more details about Shiny Server and the difference between the Open Source and Professional Editions see www.rstudio.com/products/shiny/shiny-server.

Accessing Deployed Apps

Once an app has been deployed on a server, accessing the app is very simple because all that is needed is a web browser to access the URL address where the app is located. Shiny works well on all standard computing platform environments including mobile devices. Recall in Section 4.1 that students successfully used the *Longest Run* app on their smartphones during a lecture.

6. Shiny App Resources

Here are some very helpful resources for learning more about Shiny.

1. A great starting point: The Shiny Tutorial at RStudio (shiny.rstudio.com/tutorial)
2. Many in-depth Shiny related articles can be found at shiny.rstudio.com/articles. This would be a good resource after completing the tutorial. Here are some examples:
    - Style your apps with CSS (shiny.rstudio.com/articles/css.html)
    - Shiny Cheat Sheet (www.rstudio.com/resources/cheatsheets)
3. Many articles on Shiny can also be found at R-bloggers (www.r-bloggers.com/?s=shiny)
4. Beeley (2013) is a tutorial for Shiny available in paperback and e-book formats.
5. The "Shiny – Web Framework for R" Google Group is an online forum for Shiny related issues. Questions posted at this Google Group are quickly answered, often by the developers of Shiny (groups.google.com/forum/#!forum/shiny-discuss).
6. Many useful examples of apps and Shiny widgets can be found at the official Shiny Gallery by RStudio (shiny.rstudio.com/gallery).
7. A collection of sophisticated apps can be found at the Shiny User Showcase (www.rstudio.com/products/shiny/shiny-user-showcase).

8. A large gallery (over 150) of various Shiny apps is at www.showmeshiny.com.

9. Scenarios Network for Alaska + Arctic Planning (SNAP) offers an interesting array of Shiny apps using environmental/climate data. Examples can be found at:

   - shiny.snap.uaf.edu/sea_ice_winds
   - shiny.snap.uaf.edu/sea_ice_coverage
   - shiny.snap.uaf.edu/temp_wind_events

7. Summary

There has been a large body of evidence to suggest that web application teaching tools help statistics students learn material more effectively. Many of these applications have been implemented as Java/Javascript applets. Shiny is a technology that allows an instructor to create web application teaching tools. The construction of applets can require fluency in Java, Javascript, HTML, CSS, and PHP languages, but with Shiny all that is needed is some familiarity with R.

In-class demonstrations with Shiny can yield a much more fluid and dynamic presentation than what one may typically experience by using the standard R console approach. As described in the presentation scenario of Section 2, using Shiny eliminates R interruptions due to script-changing and, in fact, the console need not ever be shown.

So far we have built a suite of 18 apps in our *Shiny App Teaching Tools Collection*. Anyone interested in accessing the source code for the apps can find them from our GitHub Gist site. Our apps have been successfully used for in-class demonstrations and for lab/homework exercises. Feedback from instructors who have used our apps has been very positive.

For anyone with some familiarity in R we believe it is very feasible to build, launch, and deploy Shiny apps. Those without access to a local server might try the free account at ShinyApps.io but please note the significant account restrictions. For instructors with access to a Linux-based server we highly recommend that they take advantage (as we did) of the RStudio Academic Pricing Policy.

We do want to note again that there are many outstanding applet teaching tools found online (e.g., the *Rossman/Chance Applet Collection* and the *Statistics Online Computational Resource*). Our *Shiny App Teaching Tools Collection* augments this publicly available set of accessible, interactive web-based statistics teaching tools. We also provide complete source code for our apps so that users may easily tailor them to their own needs. Our app gallery may stimulate ideas on how to to present statistical concepts to students, and may even inspire instructors to create a new app of their own design.

References

American Statistical Association. (2005a). *Guidelines for Assessment and Instruction in Statistics Education (GAISE): College report*. Alexandria, VA: American Statistical Association. Retrieved from www.amstat.org/education/gaise/GaiseCollege_Full.pdf.

American Statistical Association. (2005b). *Guidelines for Assessment and Instruction in Statistics Education (GAISE): Pre-K-12 Report*. Alexandria, VA: American Statistical Association. Retrieved from www.amstat.org/education/gaise/gaiseprek-12_full.pdf.

Beeley, C. (2013), *Web Application Development with R Using Shiny*, Packt Publishing.

Chance, B. & Rossman, A. (2006). Using Simulation to Teach and Learn Statistics. In A. Rossman & B. Chance (Eds.), *Proceedings of the Seventh International Conference on Teaching Statistics*. [CD-ROM]. Voorburg, The Netherlands: International Statistical Institute.

Chance, B., Ben-Zvi, D., Garfield, J. & Medina, E. (2007) "The Role of Technology in Improving Student Learning of Statistics," *Technology Innovations in Statistics Education*, 1(1). Retrieved from www.escholarship.org/uc/item/8sd2t4rr.

Chang, W., Cheng, J., Allaire, JJ, Xie, Y. & McPherson, J. (2015). shiny: Web Application Framework for R. R package version 0.11.1. Retrieved Feb. 23, 2015. Available at CRAN.R-project.org/package=shiny.

delMas, R., Garfield, J. & Chance, B. (1999), "A Model of Classroom Research in Action: Developing Simulation Activities to Improve Students' Statistical Reasoning," *Journal of Statistics Education*, 7(3).

Franklin, C. & Garfield, J. (2006). The GAISE project: Developing statistics education guidelines for pre K-12 and college courses. In G. Burrill (Ed.), *Thinking and reasoning with data and chance: 2006 NCTM yearbook* (p. 435–475). Reston, VA: National Council of Teachers of Mathematics.

Garfield, J. & Ben-Zvi, D. (2008), *Developing Students' Statistical Reasoning: Connecting Research and Teaching Practice*, Kluwer Academic Publishers.

Hagtvedt, R., Jones, G. T. & Jones, K. (2007), "Pedagogical Simulation of Sampling Distributions and the Central Limit Theorem," *Teaching Statistics*, 29, 94–97.

Harraway (2012), "Learning Statistics Using Motivational Videos, Real Data and Free Software," *Technology Innovations in Statistics Education*, 6(1). Retrieved from www.escholarship.org/uc/item/1fn7k2x3.

Lane, D. & Tang, Z. (2000), "Effectiveness of Simulation Training on Transfer of Statistical Concepts," *Journal of Educational Computing Research*, 22, 383–396.

Lunsford, M., Rowell, G. & Goodson-Espy, T. (2006), "Classroom Research: Assessment of Student Understanding of Sampling Distributions of Means and the Central Limit Theorem in Post-Calculus Probability and Statistics Classes," *Journal of Statistics Education*, 14(3).

McDaniel, S. & Green, L. (2012) "Using Applets and Video Instruction to Foster Students' Understanding of Sampling Variability," *Technology Innovations in Statistics Education*, 6(1). Retrieved from www.escholarship.org/uc/item/1nh4n607.

Pratt, D., Davies, N. & Connor, D. (2011). "The role of technology in teaching and learning statistics", In C. Batanero, G. Burril & C. Reading (Eds.), *Teaching statistics in school mathematics-challenges for teaching and teacher education* (pp. 97-108). New ICMI Study Series, vol. 15. Heidelberg, New York: Springer

R Core Team (2015). "R: A Language and Environment for Statistical Computing," *R Foundation for Statistical Computing*, Vienna, Austria. Available at www.R-project.org.

Rossman, A. & Chance, B. (2004), *The Rossmann/Chance Applet Collection*. Available at www.rossmanchance.com/applets.

RStudio (2015). RStudio: Integrated Development Environment for R (Version 0.98.1102) [Computer software]. Boston, MA. Retrieved Feb. 23, 2015. Available at www.rstudio.com.

Schilling, M. (1990) "The Longest Run of Heads," *The College Mathematics Journal*, 21(3), 196–207.

Schilling, M. (2012) "The Surprising Predictability of Long Runs," *Mathematics Magazine*, 85(2), 141–149.

Shaltayev, D., Hodges, H. & Hasbrouck, R. (2010) "VISA: Reducing Technological Impact on Student Learning in an Introductory Statistics Course," *Technology Innovations in Statistics Education*, 4(1). Retrieved from www.escholarship.org/uc/item/1gh2x5v5.

Zieffler, A., Park, J., Garfield, J., delMas, R. & Bjornsdottir, A. (2012) "The Statistics Teaching Inventory: A Survey on Statistics Teachers' Classroom Practices and Beliefs," *Journal of Statistics Education*, 20(1).

Appendix

A. Shiny Basics

A.1. Getting Started

The version of Shiny that we used is 0.11.1 which requires R version 3.0.0 or higher. R version updates are available at cran.r-project.org.

At the R console submit the following commands to install Shiny:

```
install.packages("shiny")
library(shiny)
```

To confirm successful installation, submit the following command to launch one of the built-in Shiny example apps:

```
runExample("01_hello")
```

This built-in app is a simple histogram example that we will discuss later.

Although not required, the freely available RStudio Desktop program (RStudio, 2015) can greatly facilitate work with Shiny. This software can be downloaded from www.rstudio.com. As shown later in this section, RStudio has some features that are quite convenient for Shiny app building, launching, and deployment.

We will now describe the anatomy of a Shiny app and how to launch/deploy an app. Although we will provide a basic overview of these topics, to learn more details we recommend the excellent Shiny tutorial lessons found at shiny.rstudio.com/tutorial. Currently there are a total of seven lessons and, as stated on the site, each one takes about 20 minutes and teaches one new Shiny skill. Another entry point to Shiny is via R Markdown. With R Markdown one can create dynamic documents comprised of interactive components based on Shiny. A very helpful resource that shows how one can learn Shiny using R Markdown is found at rmarkdown.rstudio.com/authoring_shiny.html.

A.2. Anatomy of a Shiny App

Each Shiny app is usually comprised of two components: a user-interface script and a server script. The layout and appearance of the app is defined in the user-interface script. Instructions for the computer to build the app are stored in the server script. The user-interface script is stored in a file called `ui.r` and the server script is stored in a filed called `server.r`.[a] The name of the directory containing these two files becomes the name of app. This can be important later when the app is launched with `runApp()` or when the app is deployed on a Shiny server. These topics will be covered later in this section.

The dynamic and interactive nature of a Shiny app is made possible through the interplay that occurs between `server.r` and `ui.r`. For example, if an app contains "widgets" for the user to specify input values (e.g., sliders, text input boxes, etc.), these widgets would usually be defined in `ui.r`. The user specified input values are then sent to `server.r` and operations based on these values can be performed (e.g., construction of a graph). The results of those computations are then sent *back* to `ui.r` where the app displays the outcome (e.g., display the graph).

Histogram App Example

As an illustration, let us return to the histogram Shiny app from Section A.1. We start by examining the contents of the corresponding `ui.r` file.

ui.r

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Histogram Shiny App"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("numBin", "Number of bins:",
                  min = 1, max = 50, value = 15)),
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

ui.r

[a] As of version 0.10.2, Shiny allows for single-file applications where the components of `server.r` and `ui.r` can be stored in one file called `app.r`.

Note the following features of `ui.r`:

- `library(shiny)` loads functions from the Shiny package.
- All user interface elements that we define are wrapped in the call `shinyUI(fluidPage())`.
- The first element is a title: `titlePanel("Histogram Shiny App")`.
- The appearance of the app will be described in `sidebarLayout()`. It will be comprised of 2 panels:
  1. The first panel will be shown on the left side as defined by `sidebarPanel()`.
     - ▷ This side panel will contain a widget in the form of a slider: `sliderInput("numBin","Number of bins:",min=1,max=50,value=15)`.
     - ▷ The value of this widget will be stored in the variable `numBin`. This argument will be used later in `server.r`.
  2. The second panel will be shown on the right side as defined by `mainPanel()`.
     - ▷ With `plotOutput("distPlot")`, a plot called `distPlot` will be displayed in the main panel. As will be seen later, `distPlot` will be created by computations (based on `numBin`) that occur in `server.r`.

Figure A.1 reveals how the elements of the histogram Shiny app correspond to the components of `ui.r`. The app uses only one widget (slider) and a simple layout. There are many other types of widgets (e.g., numeric input, radio buttons, check boxes) and layout options (e.g., tabsets, navbar, navlistPanel). These examples (including code) can be found at shiny.rstudio.com/gallery.

The code required to generate the histogram from the R console is shown below as `histogram.r`. The server script code for the Shiny app is shown below as `server.r`.

histogram.r

```
bin.num <- 15
x <- faithful[, 2]
bins <- seq(min(x),max(x),
   length.out = bin.num+1)
hist(x, breaks = bins,
     col = "darkgray",
     border = "white")
```

histogram.r

server.r

```
shinyServer(function(input,output){
 output$distPlot <- renderPlot({
   x <- faithful[, 2]
   bins <- seq(min(x), max(x),
      length.out = input$numBin+1)
   hist(x, breaks = bins,
        col = "darkgray",
        border = "white")
 })
})
```

server.r

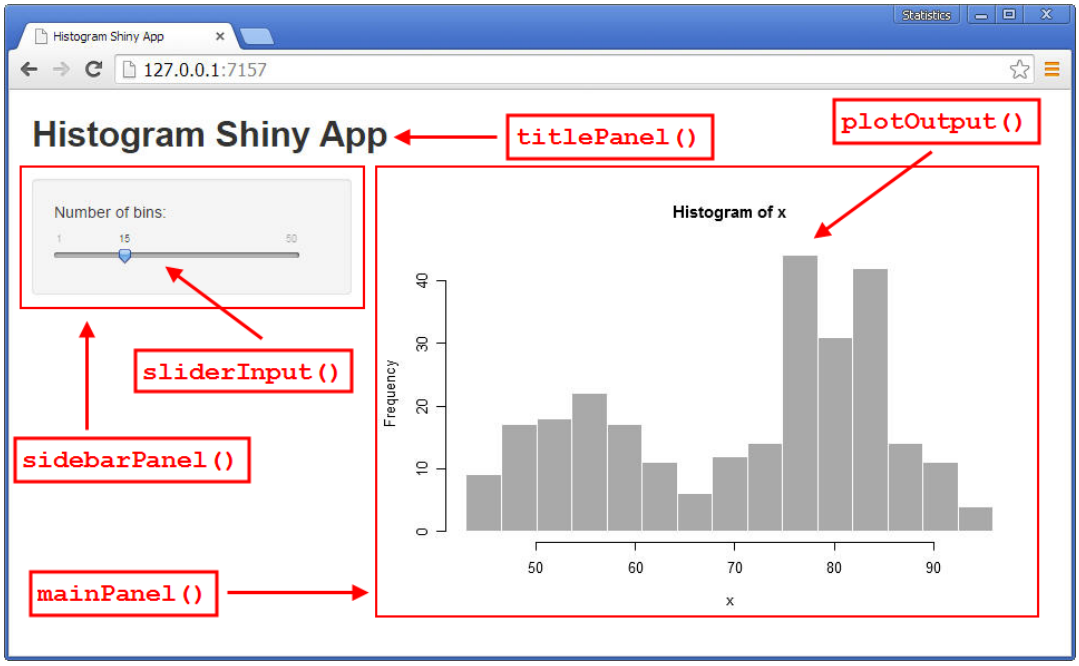

Figure A.1: Correspondence of Shiny app elements to components of `ui.r`

Note the following features of `server.r`:

- All computational commands are wrapped in the call `shinyServer(function(input, output){})`.
- `output$distPlot` indicates that an *output* object will be created and its name will be `distPlot`. The type of object is a *plot* as it is assigned the result of `renderPlot()`.
- The arguments of `renderPlot()` specify what plot to create. Note that `histogram.r` is used almost **entirely verbatim** within `renderPlot()`. The only difference is that what was previously defined to be `bin.num` has been replaced by the user specified input value `numBin` corresponding to the slider widget from `ui.r`.
- The plot `distPlot` is finally shipped back to `ui.r` as the argument of `plotOutput()` which ultimately reveals the image in the app.

A.3. Reactive Programming

As a brief introduction to reactive programming in Shiny let us revisit the *Longest Run* app from Section 2. Recall this app generates a plot containing the outcomes of simulated coin tosses and marks any runs having a length of at least some specified value. Three key features of this app are: (1) an adjustment of the trials slider updates the display with a re-randomization of coin toss outcomes, (2) an adjustment of the run length slider updates marked runs in the plot **without** forcing a re-randomization, and (3) a click of the "Generate" button re-randomizes outcomes. The prevention of re-randomization in (2) is vital as it allows one to highlight runs of different lengths for the *same* set of outcomes. Thus, what we desire is for the app to re-randomize only when the trials slider is updated or when the "Generate" button is clicked, but **not** when the run length slider is updated.

The standard behavior in Shiny is for functions to be evaluated when *any* widget is updated. This default mode would not be helpful for the *Longest Run* app as we would not be able to prevent re-randomization as stated in feature (2). It is, however, possible for an app to have a function evaluated **only** when specific widgets are updated and this can be done via so-called *reactive expressions*. This feature allows for greater control and flexibility.

Recall from Section 2 that the function that performs the randomization of all coin toss outcomes is `flip.gen()`. As found in the `server.r` code[b] for the *Longest Run* app, we used the variables corresponding to the trials slider and "Generate" button widgets as arguments for `flip.gen()`. We then embedded that entire function call into the reactive expression `reactive()`. Given that the only arguments of `flip.gen()` correspond to the trials slider and button widgets, and because the entire function is wrapped within `reactive()`, this allows for re-randomization to occur **only** when those two widgets are updated.

This is only one example of how reactive expressions work. Other examples can be found in our *Shiny App Teaching Tools Collection* as virtually all of our apps use some form of reactive programming. A general overview on this topic can be found at shiny.rstudio.com/articles/reactivity-overview.html.

A.4. Launching a Shiny App

As one builds an app by editing the corresponding `server.r` and `ui.r` files, the user will want to launch the app to see how it looks. This can be done by submitting the `runApp()` command, assuming that the app files are in the current working directory. Or one can directly specify in `runApp()` the full directory location of the folder containing the app files. After submitting `runApp()` the app should launch in a web browser.

[b] Code can be found at our GitHub Gist website (gist.github.com/calpolystat).

While the app is running note that the R console remains in a busy state and becomes inaccessible (indicated by the "Listening" message from the console)[c]. Once the app is stopped (Escape key or Stop button in the console) R will return to an interactive mode. One way to build an app is to use a standard text editor to make changes to `server.r` and `ui.r` and then manually submit `runApp()` in the R console, but this is an extremely inconvenient process. This is because, once subsequent changes are made to `server.r` or `ui.r`, the app needs to be manually stopped and then relaunched with another `runApp()` submission. This could become a very tedious cycle.

A more convenient mode of editing an app can be done with RStudio. With this software, as one edits either `server.r` or `ui.r`, a quick way to launch the app is by pressing the "Run App" button in the editor panel (keyboard shortcut CONTROL+SHIFT+ENTER). See Figure A.2. By default this launches the app in an app window as shown in Figure A.3. After additional changes are made to `server.r` or `ui.r`, pressing CONTROL+SHIFT+ENTER will *automatically refresh* the app window. So there is no need to ever manually close/re-open the app window. Using RStudio the app may be launched in the default app window, in a web browser, or in the Viewer Pane. The app launch mode can be set via a pull-down menu next to the "Run App" button.

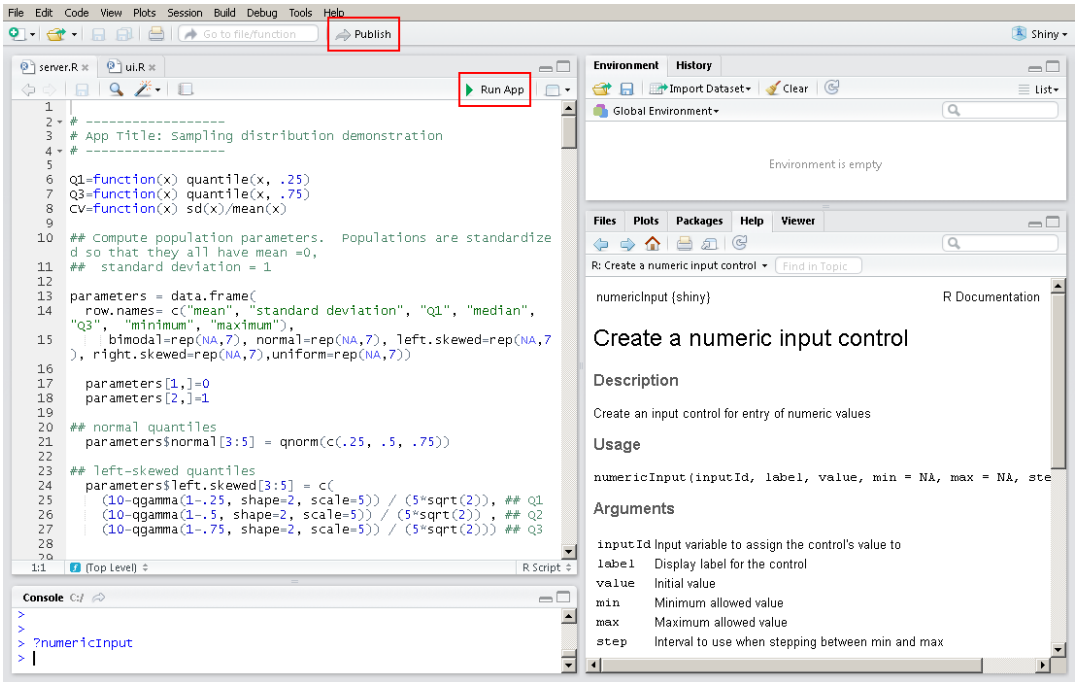

Figure A.2: RStudio environment for editing `server.r` and `ui.r`. Note the location of the "Run App" and "Publish" buttons.

[c]It is possible to run the app in a separate process and keep the console accessible. See shiny.rstudio.com/articles/running.html for more details.

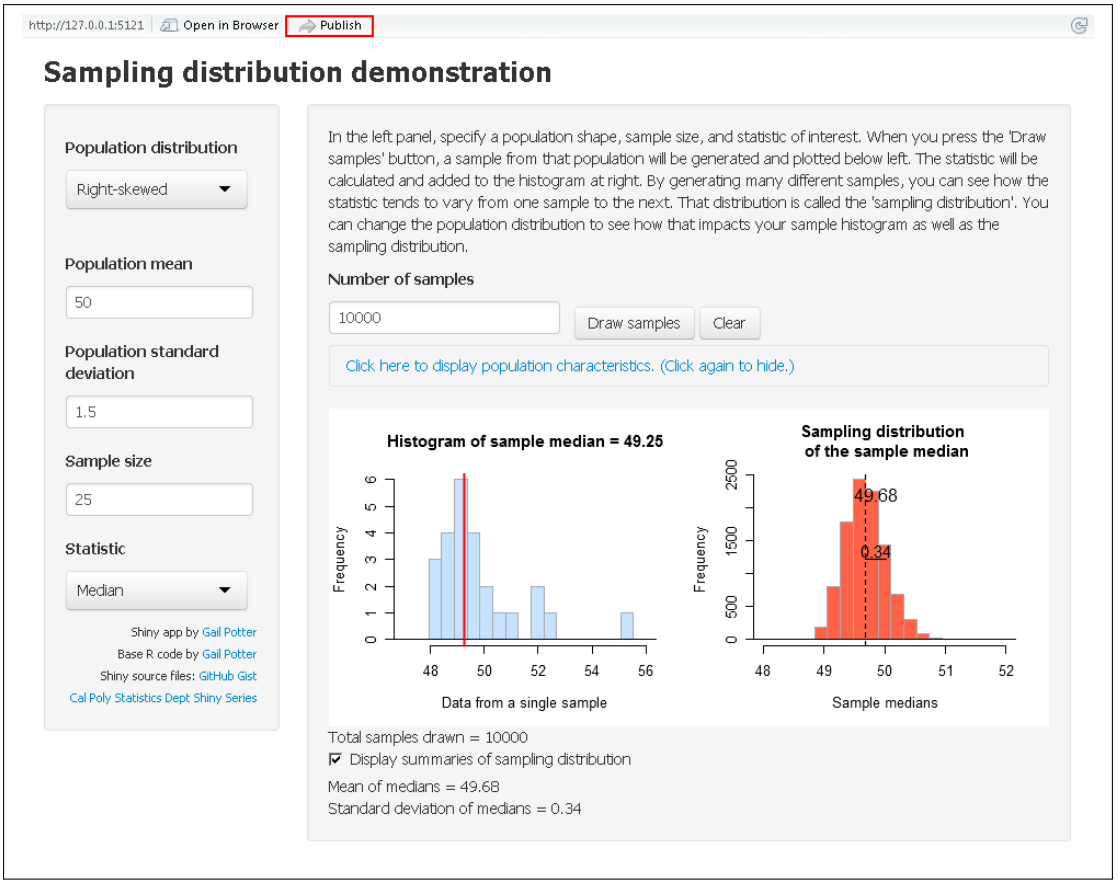

Figure A.3: Shiny app launched in the default app window. Note the location of the "Publish" button.

A.5. Deploying/Publishing a Shiny App

Once an app has been finalized, the user may want to deploy/publish it by posting it on a Shiny server. By doing so anyone can access the app with a web browser. Users with access to a Linux server can install a local Shiny Server. For more details see www.rstudio.com/products/shiny/shiny-server. Also, for those at academic institutions, see www.rstudio.com/pricing/academic-pricing for information on the RStudio Academic Pricing Policy.

Users without access to a server can deploy apps to the cloud service ShinyApps.io offered by RStudio. Apps can be uploaded for free with a restriction on the number of deployed apps and number of active use hours. Other plans that require a fee offer less restrictions. Visit shiny.rstudio.com/articles/shinyapps.html for more details on creating a free ShinyApps.io account and configuration instructions for app deployment.

Once the configuration process is completed publishing an app to ShinyApps.io is quite simple. To do so, assuming the app files are in the current working directory, submit the `deployApp()` command from the `shinyapps` package at the R console as shown here:

```
library(shinyapps)
deployApp()
```

Users can also directly specify in `deployApp()` the full directory location of the folder containing the app files.

Alternatively one can publish an app by clicking the "Publish" button that is found in RStudio (see Figure A.2) or in the default app window (see Figure A.3). Clicking this button yields the dialogue box shown in Figure A.4. Note that the name of the folder containing the app becomes the name of the app.

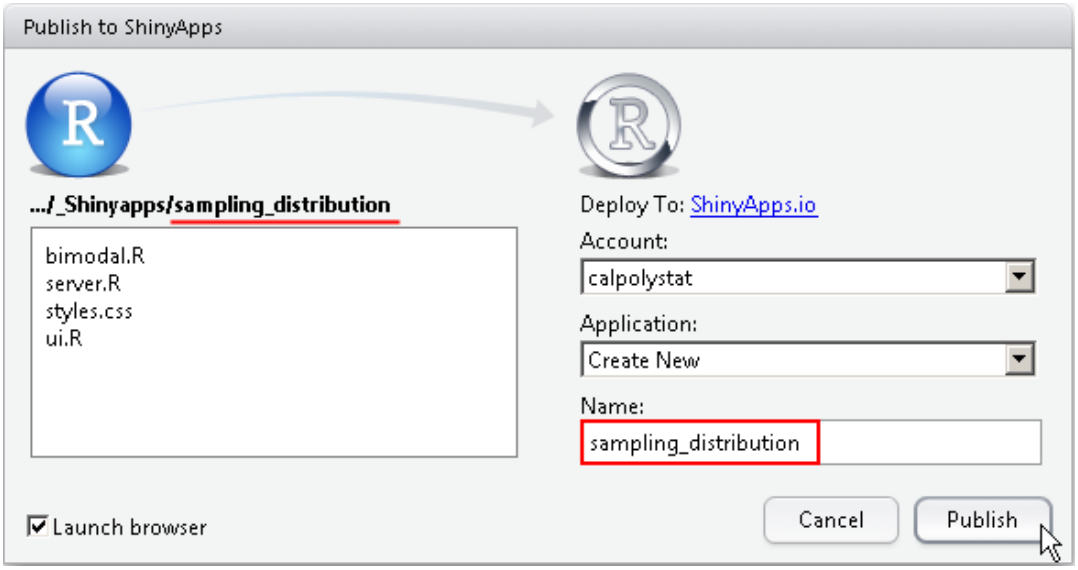

Figure A.4: Publishing a Shiny app to ShinyApps.io. The name of the folder containing the app files becomes the name of the app.

B. Teaching materials

B.1. Homework Assignment using Testing Violation of the Constant Variance Condition for ANOVA App

You are going to use this app to investigate performance of the ANOVA F-test when the condition of constant variance fails. For your first simulation, use the default values for population standard deviations, sample sizes, and population means.

1. Simulate 1000 data sets and record the estimated Type I error rate in the table below. Next, perform simulations for the other combinations of sample size and standard deviations. The Type I error rates are approximations which will improve if you do more simulations.

	Standard deviations		
Sample sizes	$6, 6, 6$	$4, 6, 8$	$1, 6, 11$
$20, 20, 20$			
$10, 20, 30$			
$5, 10, 20$			

2. Does violation of constant variance cause the Type I error rate to be different than what we'd expect? If so, is the effect different for different sample sizes?

3. Is the Type I error rate higher or lower than what we expect?

4. Is there any drawback to having a Type I error rate lower than expected? Explain.

B.2. Homework Assignment using Sampling Distributions of Various Statistics App

1. Use this app to explore the sampling distribution of the sample mean of American income. Suppose income is right skewed with mean \$53K and standard deviation \$35K.

   (a) Simulate samples of size 10 and study the histograms of the samples as well as the sampling distribution. What is the shape of the sample histograms? What is the shape of the sampling distribution of the sample mean?

   (b) Simulate samples of size 1000 and study the histograms of the samples as well as the sampling distribution. What is the shape of the sample histograms? What is the shape of the sampling distribution?

   (c) Did the shapes of the sample histograms and sampling distributions change from (a) to (b)? If so, explain how. Explain in plain English why they changed or didn't change.

2. Use the same app to explore the sampling distribution of the sample max. Assume a uniform population with mean 0, standard deviation 1. With samples of size 10, what is the distribution of the sample max? Is this an unbiased estimator for the population max? (NOTE: you can select "Display summaries of sampling distribution" to see the mean of your simulated sample statistics.) Why or why not?

B.3. Laboratory Exercise using Sampling Distributions of Various Statistics App

In this lab, you will explore the effect of sample size on the validity of the Central Limit Theorem using the *Sampling Distributions of Various Statistics* App. First, we will begin with a left-skewed population distribution. Select that from the drop-down menu, and then do the following:

- Change the sample size to 100
- Change the number of samples to 1 and click "Draw samples" just to observe what one iteration looks like
- Do a few more, and then ultimately change the number of samples so that you get at least 1000 total
- Decrease the sample size and repeat the above. Continue to decrease the sample size until the sampling distribution no longer looks like a normal distribution.

Comment on what you observed, specifically the smallest sample size at which the sampling distribution still resembles a normal distribution. Take a screen shot of the histograms at this sample size and include it with your assignment.

Then, repeat this for a Bimodal population distribution, and a Normal population distribution (are your results here surprising?)